Astra Trident 24.06ドキュメント

Astra Trident

NetApp
November 13, 2024

目次

Astra Trident 24.06ドキュメント

リリースノート

新機能

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

> `tridentctl`インストーラのzipファイルに含まれているLinux用のバイナリは、テスト済みでサポートされているバージョンです。zipファイルの一部で提供されるバイナリ `/extras`は、テストまたはサポートされていないことに注意して `macos`ください。

24.06の新機能

キノウカクチョウ

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

Kubernetes

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

の修正

- 古いトランザクションが原因でAstra Tridentのインストールが失敗する問題を修正。
- kutes()からの警告メッセージを無視するtridentctlを修正しました"問題#892"。
- Astra Tridentコントローラの優先度が（）に変更されました `SecurityContextConstraint` `0`"問題#887"。
- ONTAPドライバでは、20MiB未満のボリュームサイズを使用できるようになりました（"問題[#885"）。
- Astra Tridentを修正し、ONTAP-SANドライバのサイズ変更処理中にFlexVolが縮小されないようにしました。
- NFS v4.1でのANFボリュームのインポートエラーを修正。

非推奨

- EOLのWindows Server 2019のサポートが削除されました。

24.02の変更点

キノウカクチョウ

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

Kubernetes

- Kubernetes 1.29のサポートを追加。

の修正

- ACPが有効になっていない場合に表示されるACPの警告メッセージを修正しました（"問題#866"）。
- クローンがスナップショットに関連付けられている場合、ONTAPドライバのスナップショット削除中にクローンスプリットを実行する前に10秒の遅延が追加されました。

非推奨

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

23.10の変更点

の修正

- 新しい要求サイズがONTAP - NASおよびONTAP - NAS - FlexGroupストレージドライバの合計ボリュームサイズよりも小さい場合、ボリュームの拡張を修正しました（"問題#834"）。
- ONTAP - NASおよびONTAP - NAS - FlexGroupストレージドライバのインポート時にボリュームの使用可能なサイズのみを表示する固定ボリュームサイズ（"問題#722"）。
- ONTAP-NAS-EconomyのFlexVol名変換が修正されました。
- ノードのリブート時にWindowsノードでAstra Tridentの初期化問題が発生する問題が修正されました。

キノウカクチョウ

Kubernetes

Kubernetes 1.28のサポートを追加。

Astra Trident

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

Astra Controlの高度な機能

Astra Trident 23.10では、Astra Controlのライセンスユーザがastra Control Provisionerと呼ばれる新しいソフトウェアコンポーネントを利用できます。このプロビジョニングツールでは、Astra Tridentだけではサポートされない、高度な管理機能とストレージプロビジョニング機能のスーパーセットを利用できます。23.10リリースでは、次の機能があります。

- ONTAP NAS経済性に優れたドライバベースのストレージバックエンドで、アプリケーションのバックアップとリストアを実現
- Kerberos 5暗号化によるストレージバックエンドのセキュリティの強化
- スナップショットを使用したデータリカバリ
- SnapMirrorの機能拡張

"Astra Control Provisionerの詳細をご確認ください。"

23.07.1の変更点

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

23.07の変更点

の修正

Kubernetes

- 古いポッドがterminating状態で停止するのを無視するためのTridentアップグレードを修正しました"問題#740"()。
- 「transient-toleration-toleration-pod Trident」定義（）に公差を追加しました"問題#795"。

Astra Trident

- ノードステージング操作中にゴーストiSCSIデバイスを識別して修正するためのLUN属性を取得するときに、LUNシリアル番号が照会されるようにするためのONTAP ZAPI要求を修正しました。
- ストレージドライバコード()のエラー処理を修正しました"問題#816"。
- use-rest = trueを指定してONTAPドライバを使用すると、クォータのサイズが修正されました。
- ONTAP-SAN-EconomyでLUNクローンを固定作成
- パブリッシュ情報フィールドをからに `devicePath`戻し `rawDevicePath`ます。フィールドに値を入力して回復するためのロジックが追加されました（場合によっては） `devicePath`。

キノウカクチョウ

Kubernetes

- 事前プロビジョニングされたSnapshotのインポートのサポートが追加されました。
- 最小化された配備とデーモン設定Linuxパーミッション"問題#817"()

Astra Trident

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

実験的強化

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

ドキュメント

組織とフォーマットの多くの改善が行われました。

非推奨

Kubernetes

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

23.04の変更点

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

の修正

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

キノウカクチョウ

- Kubernetes：
  - Kubernetes 1.27のサポートを追加。
  - LUKSボリュームのインポートのサポートが追加されました。
  - ReadWriteOncePod PVCアクセスモードのサポートが追加されました。
  - ノードの正常でないシャットダウン時にONTAP-SAN-*ボリュームで強制的に接続解除がサポートされるようになりました。
  - すべてのontap-san-*ボリュームでノード単位のigroupを使用するようになりました。LUNはigroupにマッピングされるだけで、それらのノードにアクティブにパブリッシュされるため、セキュリティ体制が強化されます。既存のボリュームは、アクティブなワークロードに影響を与えずに安全であるとTridentが判断した場合に、必要に応じて新しいigroupスキームに切り替えられます（"問題#758"）。
  - Tridentで管理されていないigroupをONTAP-SAN-*バックエンドからクリーンアップし、Tridentのセキュリティを強化
- ストレージドライバontap-nas-economyとontap-nas-flexgroupに、Amazon FSxによるSMBボリュームのサポートが追加されました。
- ontap-nas、ontap-nas-economy、ontap-nas-flexgroupストレージドライバでSMB共有のサポートが追加されました。

- arm64ノードのサポートを追加しました"問題#732"()。
- 最初にAPIサーバを非アクティブ化することにより、Tridentのシャットダウン手順が改善されました（"問題#811"）。
- Windowsおよびarm64ホストのクロスプラットフォームビルドサポートをMakefileに追加しました。build.mdを参照してください。

非推奨

- Kubernetes：** ONTAP SANドライバおよびONTAP SANエコノミードライバの構成時に、バックエンドスコープigroupが作成されなくなりました（"問題#758"）。

23.01.1の変更点

の修正

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

23.01の変更点

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

の修正

- Kubernetes：Helm()を使用してTridentのインストールを修正するポッドセキュリティポリシーの作成を除外するオプションが追加されました"問題#783、#794"。

キノウカクチョウ

Kubernetes

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

Astra Trident

- ONTAP 対応のAmazon FSXを使用したSMBボリュームのONTAP NASストレージドライバへのサポートが追加されました。

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

非推奨

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

22.10の変更点

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

Astra Tridentに関する重要な情報22.10

- TridentでKubernetes 1.25がサポートされるようになりました。Kubernetes 1.25にアップグレードする前に、Astra Tridentを22.10にアップグレードする必要があります。
- SAN環境では、Astra Tridentでマルチパス構成の使用が厳密に適用されるようになりました。これには、multipath.confファイルの推奨値が `find_multipaths: no`含まれています。

  マルチパス以外の構成を使用するか、multipath.confファイルにまたは `find_multipaths: smart`の値を使用する `find_multipaths: yes`と、マウントが失敗します。Tridentでは、21.07リリース以降での使用を推奨してい `find_multipaths: no`ます。

の修正

- 22.07.0のアップグレード中にフィールドを使用して作成されたONTAPバックエンドに固有の問題が修正されました `credentials`（"問題#759"）。
- **Docker**：一部の環境（および"問題#760"）でDockerボリュームプラグインが起動しない問題を修正しました"問題#548"。
- レポートノードに属するデータLIFのサブセットのみが公開されるように、ONTAP SANバックエンド固有の修正されたSLM問題 。
- ボリュームの接続時にiSCSI LUNの不要なスキャンが発生するというパフォーマンス問題 の問題が修正されました。
- Astra Trident iSCSIワークフロー内で詳細な再試行を削除し、失敗の時間を短縮。外部の再試行間隔も短縮
- 対応するマルチパスデバイスがすでにフラッシュされている場合にiSCSIデバイスのフラッシュ時にエラーが返される修正問題 。

キノウカクチョウ

- Kubernetes：
  - Kubernetes 1.25のサポートを追加。Kubernetes 1.25にアップグレードする前に、Astra Tridentを22.10にアップグレードする必要があります。
  - Trident Deployment and DemonSet用に別々のServiceAccount、ClusterRole、ClusterRoleBindingを追加して、今後の権限の強化を可能にしました。
  - のサポートが追加されました"ネームスペース間ボリューム共有"。
- すべてのTrident `ontap-*`ストレージドライバがONTAP REST APIで動作するようになりました。
- Kubernetes 1.25をサポートするために、(`bundle_post_1_25.yaml`を使用せずに新しい演算子yaml）を`PodSecurityPolicy`追加しました。
- および `ontap-san-economy`ストレージドライバ用に `ontap-san`追加されました"LUKS暗号化ボリュームをサポートします"。
- Windows Server 2019ノードのサポートが追加されました。
- ストレージドライバを使用して `azure-netapp-files`追加され"WindowsノードでのSMBボリュームのサポート"ます。
- ONTAP ドライバの自動MetroCluster スイッチオーバー検出機能が一般提供されるようになりました。

非推奨

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

22.07の変更点

の修正

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

キノウカクチョウ

- CSIイメージのデフォルトレジストリとして、k8s .gcr.ioからregistry.k8s .ioに移行します
- ONTAP SANボリュームでは、ノード単位のigroupが使用され、LUNがigroupにマッピングされると同時に、これらのノードにアクティブに公開されてセキュリティ体制が強化されます。既存のボリュームは、アクティブなワークロードに影響を与えずに安全であるとAstra Tridentが判断したときに、必要に応じて新しいigroupスキームに切り替えられます。
- TridentのインストールにResourceQuotaが含まれ、PriorityClassの消費がデフォルトで制限されたときにTrident DemonSetがスケジュールされるようになりました。

- Azure NetApp Filesドライバにネットワーク機能のサポートが追加されました。("GitHub問題#717")
- ONTAP ドライバにTech Previewの自動MetroCluster スイッチオーバー検出機能を追加。("GitHub問題#228")

非推奨

- **Kubernetes**：サポートされている最小Kubernetesを1.19に更新。
- バックエンド構成では、単一の構成で複数の認証タイプを使用できなくなりました。

削除します

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

ドキュメント

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

22.04の変更点

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

> 以前のTridentリリースからアップグレードし、Azure NetApp Filesを使用する場合、`location`configパラメータは必須の単一フィールドになりました。

の修正

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

キノウカクチョウ

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

削除します

E シリーズドライバ（ 20.07 以降無効）が削除されました。

22.01.1の変更点

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

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。
- リソース名の重複を許可し、操作を単一の場所に制限するためのAzure NetApp Filesドライバを修正しました。
- リソース名の重複を許可し、操作を単一の場所に制限するためのAzure NetApp Filesドライバを修正しました。

キノウカクチョウ

- Kubernetes の機能拡張：
  - Kubernetes 1.23のサポートを追加。
  - Trident Operator または Helm 経由でインストールした場合、 Trident ポッドのスケジュールオプションを追加します。("GitHub問題#651")

- GCP ドライバでリージョン間のボリュームを許可します。("GitHub問題#633")
- Azure NetApp Filesボリュームに「unixPermissions」オプションがサポートされるようになりました。("GitHub問題#666")

非推奨

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

21.10.1の変更点

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

の修正

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

21.10.0の変更点

の修正

- XFS ボリュームのクローンをソースボリュームと同じノードにマウントできない固定問題（ GitHub 問題 514 ）
- 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：
  - および `ontap-nas-flexgroup`ドライバを使用してスナップショットを作成する場合、ボリュームの使用済みスペースを最小restoreSizeとして返し `ontap-nas`ます(GitHub Issue 645)。
  - ボリュームのサイズ変更後にエラーが記録される問題が修正されまし `Failed to expand filesystem`た(GitHub Issue 560)。
  - ポッドが状態で動かなくなる問題を修正 `Terminating`(GitHub Issue 572)。
  - FlexVolがスナップショットLUNでいっぱいになる場合がある問題を修正しまし `ontap-san-economy`た(GitHub Issue 533)。
  - 異なるイメージを持つ固定カスタム YAML インストーラ問題（ GitHub 問題 613 ）。
  - Snapshot サイズの計算方法を固定（ GitHub 問題 611 ）。
  - 問題は修正され、 Astra Trident のすべてのインストーラが OpenShift としてプレーン Kubernetes を識別できるようになりました（ GitHub 問題 639 ）。
  - Kubernetes API サーバにアクセスできない場合に、 Trident オペレータが更新を停止するよう修正しました（ GitHub 問題 599 ）。

キノウカクチョウ

- GCP-CVS Performanceボリュームのオプションのサポートが追加されました `unixPermissions`。
- 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のサポートを追加。
  - 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
  - 画像コマンドに演算子画像を追加 `tridentctl`(GitHub Issue 570)。

実験的な機能強化

- ドライバでのボリュームレプリケーションのサポートが追加されました `ontap-san`。
- 、 `ontap-san`、および `ontap-nas-economy`ドライバの`* tech preview * REST`サポートを追加 `ontap-nas-flexgroup`。

既知の問題

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

- Astra TridentがインストールされているKubernetesクラスタを1.24から1.25以降にアップグレードする場合は `helm upgrade`、クラスタをアップグレードする前に、値.yamlをに `true`設定するかコマンドに追加するよう `--set excludePodSecurityPolicy=true`に更新する必要があります。 `excludePodSecurityPolicy`
- Astra Tridentでは 、StorageClassで指定された(`fsType=""`がないボリュームに対して `fsType`空白が適用されるようになりました `fsType`。Tridentでは、Kubernetes 1.17以降を使用する場合、NFSボリュームに空のを指定できます `fsType`。iSCSIボリュームの場合、セキュリティコンテキストの使用を適用するときは、StorageClassで `fsGroup`を設定する必要があります `fsType`。
- 複数のAstra Tridentインスタンスでバックエンドを使用する場合は、各バックエンド構成ファイルの値がONTAPバックエンドに対して異なるか、SolidFireバックエンドに対して異なる値を使用する `TenantName`必要があります `storagePrefix`。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。 `storagePrefix` `TenantName`同じバックエンドに作成されたボリュームで名前の競合が発生する可能性があります。
- Astra Tridentをインストールし（またはTrident Operatorを使用）、を使用して `tridentctl`Astra Tridentを管理する場合は `tridentctl`、環境変数が設定されていることを確認する必要があります `KUBECONFIG`。これは、対象となるKubernetesクラスタを指定するために必要 `tridentctl`です。複数のKubernetes環境を使用する場合は、ファイルが正確にソースされていることを確認する必要があり `KUBECONFIG`ます。
- iSCSI PVS のオンラインスペース再生を実行するには、作業者ノード上の基盤となる OS がボリュームにマウントオプションを渡す必要があります。これはRHEL/RedHat CoreOSインスタンスに当てはまります `discard` "マウントオプション"。オンラインブロック破棄をサポートするには、discard mountOptionが^]に含まれていることを確認してください。[`StorageClass`
- 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。

- Astra TridentがオフラインのときにAstra Tridentベースのオブジェクトが Kubernetesから削除された場合、 `StorageClass`Astra Tridentはオンラインに戻っても対応するストレージクラスをデータベースから削除しません。これらのストレージクラスは、またはREST APIを使用して削除して `tridentctl`ください。
- 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。またはREST APIを使用してボリュームを削除してください `tridentctl`。
- FlexGroup では、プロビジョニング要求ごとに一意のアグリゲートセットがないかぎり、同時に複数の ONTAP をプロビジョニングすることはできません。
- IPv6経由のAstra Tridentを使用する場合は、バックエンド定義でとを `dataLIF`角かっこで指定する必要があります `managementLIF`。たとえば、 `[fd20:8b1e:b258:2000:f816:3eff:feec:0]`です。

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

- OpenShift 4.5でドライバを使用する場合 `solidfire-san`は、基盤となるワーカーノードがMD5をCHAP認証アルゴリズムとして使用していることを確認します。Element 12.7では、FIPS準拠のセキュアなCHAPアルゴリズムSHA1、SHA-256、およびSHA3-256が提供されています。

詳細情報

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

以前のバージョンのドキュメント

Astra Trident 24.06を実行していない場合は、以前のリリースのドキュメントをベースに提供してい "Astra Tridentのサポートライフサイクル"ます。

- "Astra Trident 24.02"
- "Astra Trident 23.10"
- "Astra Trident 23.07"
- "Astra Trident 23.04"
- "Astra Trident 23.01"
- "Astra Trident 22.10"
- "Astra Trident 22.07"
- "Astra Trident 22.04"
- "Astra Trident 22.01"
- "Astra Trident 21.10"

開始する

Astra Trident の詳細をご確認ください

Astra Trident の詳細をご確認ください

Astra Tridentは、の一部としてNetAppが管理している、フルサポートのオープンソースプロジェクト"Astra 製品ファミリー"です。Container Storage Interface（CSI）などの業界標準のインターフェイスを使用して、コンテナ化されたアプリケーションの永続性要求を満たすように設計されています。

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、Astra Tridentを基盤に構築された永続的コンテナストレージをプロビジョニング、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などの高度なアプリケーション対応データ管理機能も提供し、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に対応します。

詳細については、をご覧ください "Astraをご利用いただくか、無償トライアルにご登録ください"。

Astra Tridentとは

Astra Tridentでは、パブリッククラウドやオンプレミスにあるONTAP（AFF、NetApp FAS、Select、Cloud、Amazon FSx for NetApp ONTAP）、Elementソフトウェア（NetApp HCI、SolidFire）、Azure NetApp Filesサービス、Cloud Volumes Service on Google Cloud

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

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

Kubernetesを初めて使用する場合は、について理解しておく必要があります"Kubernetesの概念とツール"。

Astra TridentのTest Driveプログラム

Test Driveプログラムを利用するには、すぐに使用できるラボイメージを使用して、「コンテナ化されたワークロード向けの永続的ストレージの簡単な導入とクローニング」へのアクセスをリクエストしてください"ネットアップのテスト用ドライブ"。このテストドライブは、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。これは、Astra Tridentについて理解を深め、その機能を確認するための優れた方法です。

もう1つのオプションは、"kubeadm インストールガイド"Kubernetesが提供するです。

> これらの手順を使用して構築したKubernetesクラスタは、本番環境では使用しないでください。本番環境向けクラスタ向けに、ディストリビューションから提供されている本番環境導入ガイドを使用します。

KubernetesとNetApp製品の統合

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

NetApp ONTAP 対応の Amazon FSX

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

Azure NetApp Files

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

Cloud Volumes ONTAP

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

Cloud Volumes Service for Google Cloud

"NetApp Cloud Volumes Service for Google Cloud"は、NFSおよびSMB経由でNASボリュームを提供するクラウドネイティブのファイルサービスで、オールフラッシュのパフォーマンスを実現します。

Element ソフトウェア

"要素"ストレージ管理者は、パフォーマンスを保証し、シンプルで合理的なストレージ設置面積を実現することで、ワークロードを統合できます。

NetApp HCI

"NetApp HCI"日常業務を自動化し、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。

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

NetApp ONTAP

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

ONTAP システムには、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。Astra Tridentは、これらのONTAP導入モデルをサポートしています。

詳細情報

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

Astra Tridentのアーキテクチャ

Astra Tridentは、単一のコントローラポッドと、クラスタ内の各ワーカーノード上のノードポッドとして実行されます。Astra Tridentボリュームをマウントするすべてのホストでノードポッドが実行されている必要があります。

コントローラポッドとノードポッドについて

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

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

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

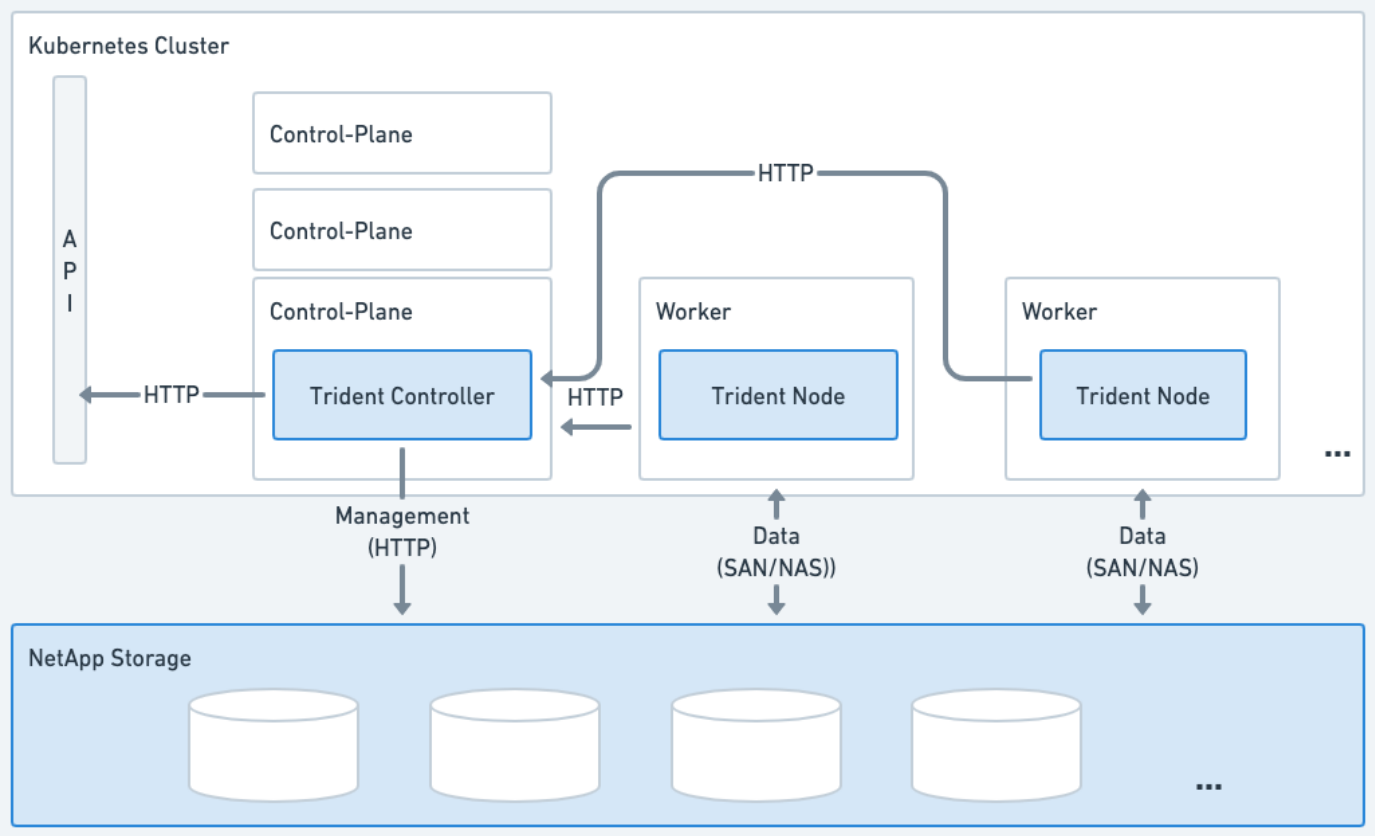


図 **1. Kubernetes**クラスタに導入される**Astra Trident**

Tridentコントローラポッド

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

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

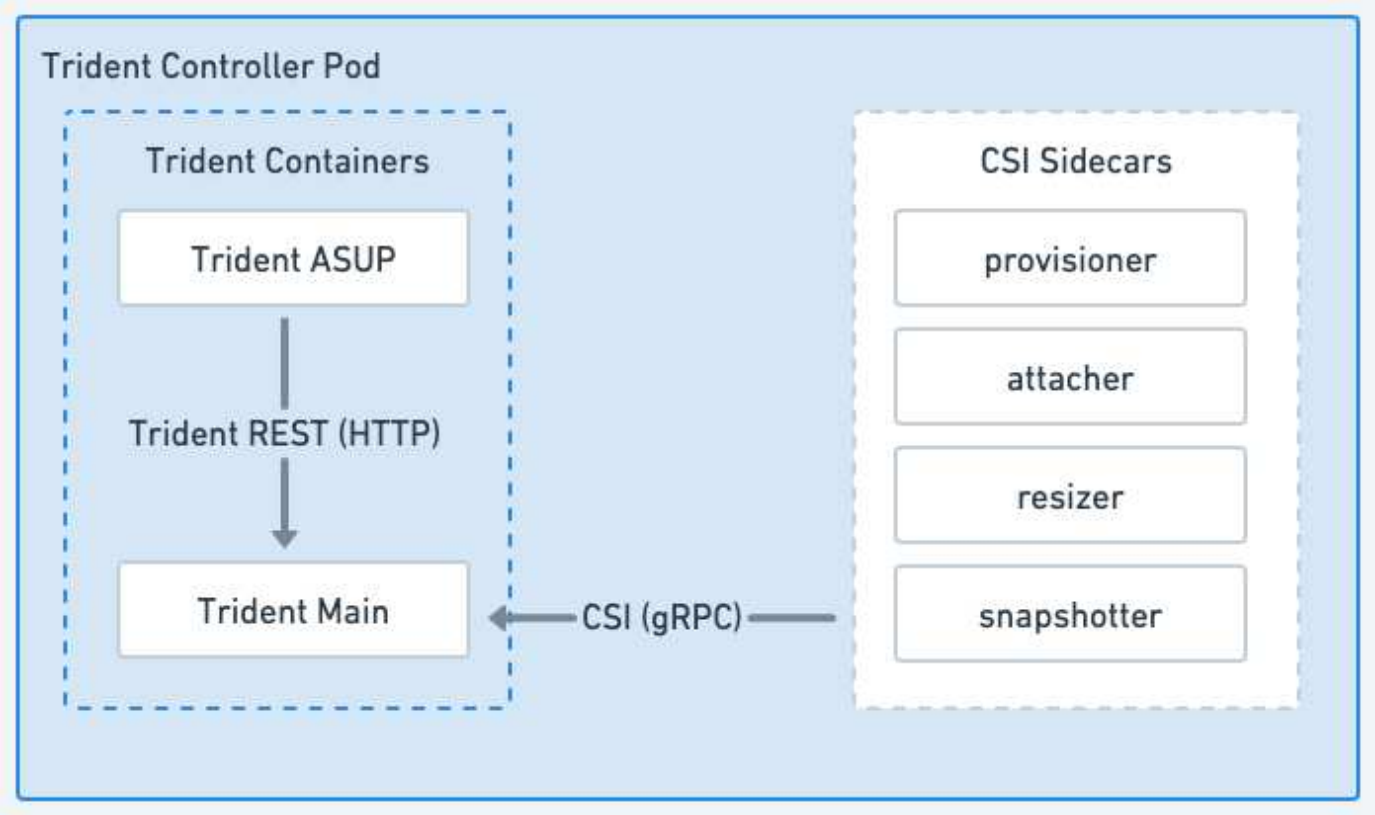


図 **2. Trident**コントローラポッドの図

Tridentノードポッド

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

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

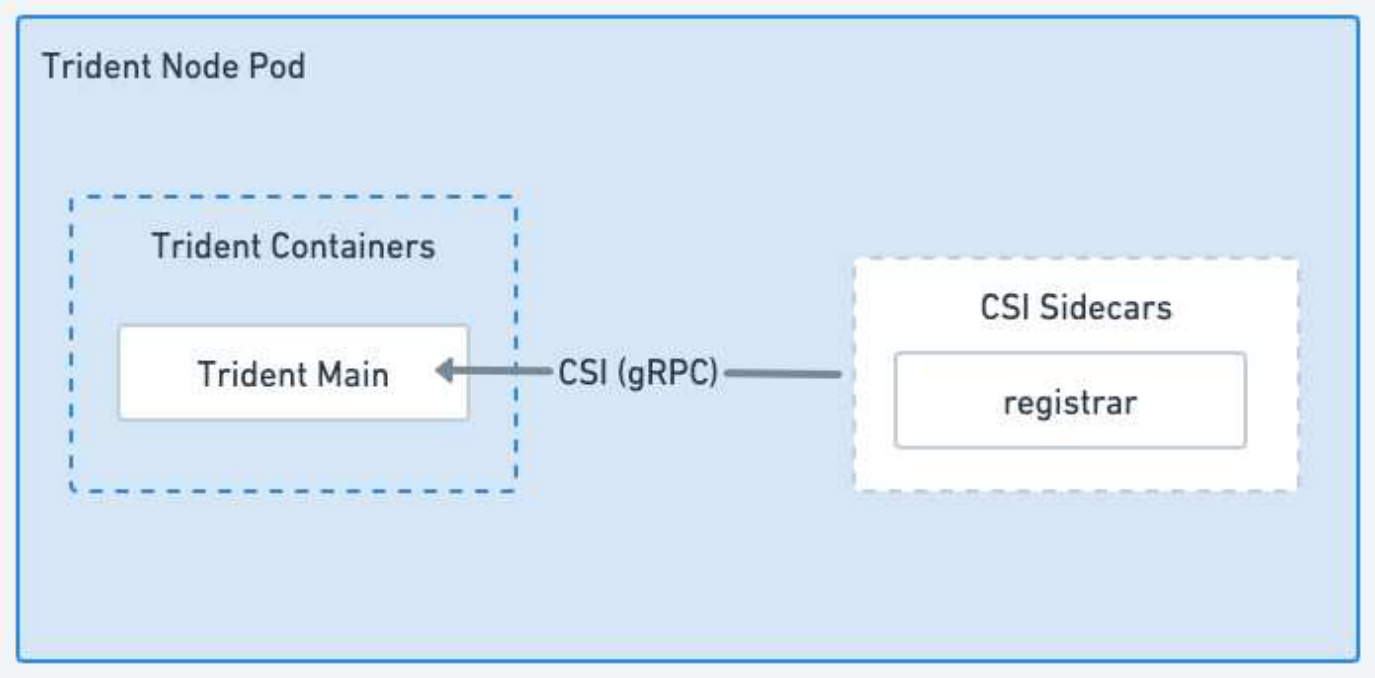


図 **3. Trident**ノードのポッド図

サポートされる **Kubernetes** クラスタアーキテクチャ

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	サポート対象	デフォルトのインストールです
単一マスター、コンピューティング	はい	はい
複数のマスター、コンピューティング	はい	はい
マスター、 `etcd`コンピューティング	はい	はい
マスター、インフラ、コンピューティング	はい	はい

概念

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、保留状態のボリュームのストレージクラスに関連付けられたストレージプールの中からストレージプールを選択する必要があります。

ストレージクラスの関連付け

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

と、 `additionalStoragePools`の `excludeStoragePools`リストの両方が `storagePools`必要です。ストレージクラスを作成すると、 Trident はバックエンドごとに提供される属性とプールを、ストレージクラスから要求された属性とプールと比較します。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、Astra Tridentは、リストに表示されているすべてのストレージプールをそのセットに追加します。プール `additionalStoragePools`の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同様です。このリストを使用して、ストレージクラスでのストレージプールの使用を無効にしたり削除したりする必要があり `excludeStoragePools`ます。Astra Trident では、新しいバックエンドを追加するたびに同様のプロセスが実行され、ストレージプールが既存のストレージクラスのストレージクラスを満たしているかどうかが確認され、除外済みとマークされているストレージが削除されます。

ボリュームの作成

Trident がさらに、ストレージクラスとストレージプールの間の関連付けを使用して、ボリュームのプロビジョニング先を決定します。ボリュームを作成すると、最初にそのボリュームのストレージクラス用の一連のストレージプールが Trident から取得されます。 また、ボリュームにプロトコルを指定した場合、 Astra Trident は要求されたプロトコルを提供できないストレージプールを削除します（たとえば、 NetApp HCI / SolidFire バックエンドはファイルベースのボリュームを提供できませんが、 ONTAP NAS バックエンドはブロックベースのボリュームを提供できません）。Trident がこのセットの順序をランダム化し、ボリュームを均等に分散してから、各ストレージプールでボリュームを順番にプロビジョニングしようとします。成功した場合は正常に返され、プロセスで発生したエラーが記録されます。Astra Trident は、要求されたストレージクラスとプロトコルで使用可能なすべてのストレージプールで * プロビジョニングに失敗した場合にのみ、障害 * を返します。

ボリューム **Snapshot**

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

ボリューム**Snapshot**の作成方法について説明します

- `ontap-san`、、、 `gcp-cvs` `azure-netapp-files`ドライバの場合、 `ontap-nas`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安定性'拡張性'リカバリ性'パフォーマンスを提供しますSnapshot コピーは、作成時とストレージスペースの両方で非常に効率的です。
- ドライバの場合 `ontap-nas-flexgroup`、各永続ボリューム（PV）はFlexGroupにマッピングされます。その結果、ボリューム Snapshot は NetApp FlexGroup Snapshot として作成されます。NetAppのスナップショット・テクノロジーは'競合するスナップショット・テクノロジーよりも安定性'拡張性'リカバリ性'パフォーマンスを提供しますSnapshot コピーは、作成時とストレージスペースの両方で非常に効率的です。
- ドライバの場合 `ontap-san-economy`、PVSは共有FlexVol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 FlexCloneテクノロジを使用すると、大規模なデータセットのコピーをほぼ瞬時に作成できます。コピーと親でデータブロックが共有されるため、メタデータに必要な分しかストレージは消費されません。
- ドライバの場合 `solidfire-san`、各PVは、NetApp Elementソフトウェア/ NetApp HCIクラスタ上に作成されたLUNにマッピングされます。ボリューム Snapshot は、基盤となる LUN の Element Snapshot で表されます。これらの Snapshot はポイントインタイムコピーであり、消費するシステムリソースとスペースはごくわずかです。
- ドライバと `ontap-san`ドライバを使用する場合、 `ontap-nas`ONTAPスナップショットはFlexVolのポイ

ントインタイムコピーであり、FlexVol自体のスペースを消費します。その結果、ボリューム内の書き込み可能なスペースが、 Snapshot の作成やスケジュール設定にかかる時間を短縮できます。この問題に対処する簡単な方法の 1 つは、 Kubernetes を使用してサイズを変更することでボリュームを拡張することです。もう 1 つの方法は、不要になった Snapshot を削除することです。Kubernetes で作成されたボリューム Snapshot を削除すると、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。

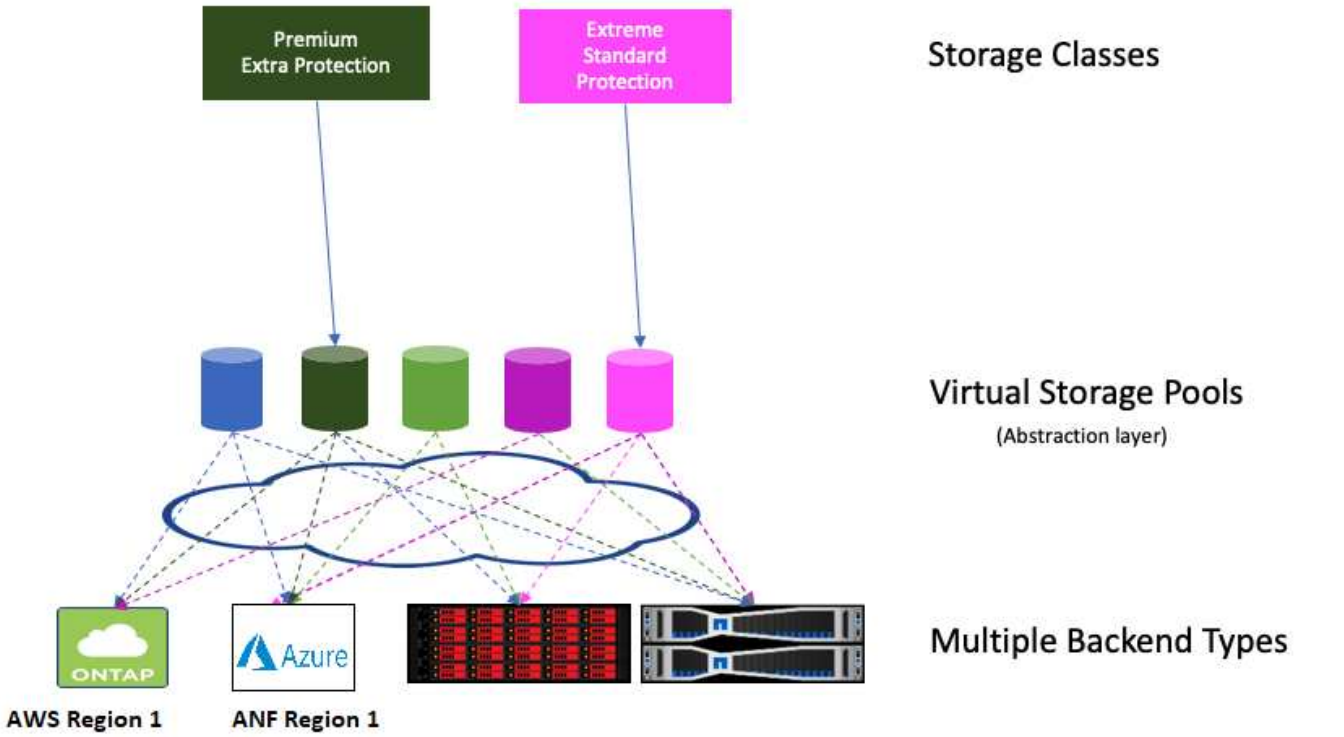
ネットアップの Trident では、ボリューム Snapshot を使用して PVS を新規作成できます。これらの Snapshot から PVS を作成するには、サポート対象の ONTAP および CVS バックエンドに対して FlexClone テクノロジを使用します。SnapshotからPVを作成する場合、元のボリュームはSnapshotの親ボリュームのFlexCloneになります。 `solidfire-san`ドライバは、Elementソフトウェアのボリュームクローンを使用してSnapshotからPVSを作成します。ここで、 Element Snapshot からクローンを作成しま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

仮想プール

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

仮想プールについて説明します

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



仮想プールリストの外部で指定されたすべての要素はバックエンドにグローバルであり、すべての仮想プール

に適用されます。一方、各仮想プールは、 1 つまたは複数の要素を個別に指定できます（バックエンドグローバルな要素を上書きします）。

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

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

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

運用者	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

Astra Tridentの使用方法の詳細をご確認 "ボリュームアクセスグループ"ください。

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

ボリュームアクセスグループについて学習する

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

Astra Tridentは、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ますが、アクセスグループ自体の作成や管理は行いません。アクセスグループには、ストレージバックエンドをAstra Tridentに追加する前に存在している必要があります。また、そのバックエンドでプロビジョニングされたボリュームをマウントできる可能性があるKubernetesクラスタ内のすべてのノードのiSCSI IQNが含まれている必要があります。ほとん

どのインストール環境では、クラスタ内のすべてのワーカーノードがこれに含まれます。

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

デフォルトのアクセスグループを使用しているアクセスグループから他のアクセスグループを使用しているアクセスグループに設定を変更する場合 `trident`は、リストにアクセスグループのIDを含め `trident`ます。

Astra Tridentのクイックスタート

Astra Tridentをインストールすると、わずかな手順でストレージリソースの管理を開始できます。作業を開始する前に、を参照してください"Astra Trident の要件"。

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

1 Astra Tridentのインストール

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

"Astra Trident をインストール"

2 ワーカーノードの準備

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

"ワーカーノードを準備します"

3 バックエンドの作成

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。

"バックエンドの設定"ストレージシステム

4 Kubernetesストレージクラスの作成

Kubernetes StorageClassオブジェクトでは、Astra Tridentがプロビジョニングツールとして指定され、カスタマイズ可能な属性を使用してボリュームをプロビジョニングするためのストレージクラスを作成できます。Astra Tridentは、Astra Tridentプロビジョニングツールを指定する、Kubernetesオブジェクト用の一致するストレージクラスを作成します。

"ストレージクラスを作成する。"

5 ボリュームをプロビジョニングする

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

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

"ボリュームをプロビジョニングする"

次の手順

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

要件

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

Astra Tridentに関する重要な情報

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

- Astra TridentでKubernetes 1.31がサポートされるようになりました。Kubernetesをアップグレードする前にAstra Tridentをアップグレードしてください。
- Astra Tridentでは、SAN環境でのマルチパス構成の使用が厳密に実施されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  マルチパス以外の構成を使用するか、multipath.confファイルにまたは `find_multipaths: smart`の値を使用する `find_multipaths: yes`と、マウントが失敗します。Astra Tridentでは、2007年21.07リリースからの使用が推奨されて `find_multipaths: no`います。

サポートされるフロントエンド（オーケストレーションツール）

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

- Anthosオンプレミス（VMware）とAnthos（ベアメタル1.16）
- Kubernetes 1.24~1.31
- OpenShift 4.10-4.16

Trident オペレータは、次のリリースでサポートされています。

- Anthosオンプレミス（VMware）とAnthos（ベアメタル1.16）

- Kubernetes 1.24~1.31
- OpenShift 4.10-4.16

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

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

Astra Tridentがインストールされ ているKubernetesクラスタを1.24から1.25以降にアップグレードする前に、を参照してください"Helmインストールのアップグレード"。

サポートされるバックエンド（ストレージ）

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

機能の要件

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

機能	**Kubernetes** のバージョン	フィーチャーゲートが必要ですか？
Astra Trident	1.24 ~ 1.31	いいえ
ボリューム Snapshot	1.24 ~ 1.31	いいえ
ボリューム Snapshot からの PVC	1.24 ~ 1.31	いいえ
iSCSI PV のサイズ変更	1.24 ~ 1.31	いいえ
ONTAP 双方向 CHAP	1.24 ~ 1.31	いいえ
動的エクスポートポリシー	1.24 ~ 1.31	いいえ
Trident のオペレータ	1.24 ~ 1.31	いいえ

機能	**Kubernetes** のバージョン	フィーチャーゲートが必要ですか？
CSI トポロジ	1.24 ~ 1.31	いいえ

テスト済みのホストオペレーティングシステム

Astra Tridentは特定のオペレーティングシステムを正式にサポートしていませんが、動作確認済みのものは次のとおりです。

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

  NVMe/TCPにはRHEL 9以降が必要です。

- Ubuntu 22.04以降（AMD64およびARM64）
- Windows Server 2022

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。

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

ホストの設定

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

"ワーカーノードを準備します"

ストレージシステムの構成：

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

"バックエンドを設定"

Astra Trident ポート

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

"Astra Trident ポート"

コンテナイメージと対応する **Kubernetes** バージョン

エアギャップのある環境では、 Astra Trident のインストールに必要なコンテナイメージを次の表に示しま

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

Kubernetesのバージョン	コンテナイメージ
v1.24.0、v1.25.0、v1.26.0、v1.27.0、v1.28.0、v1.29.0、v1.30.0、v1.31.0	• Docker .io / NetApp / Trident：24.06.0 • docker.io / netapp/trident-autosupport：24.06 • registry.k8s.io/sig-storage/csi-provisioner：v4.0.1 • registry.k8s.io/sig-storage/csi-attacher：v4.6.0 • registry.k8s.io/sig-storage/csi-resizer：v1.11.0 • registry.k8s.io/sig-storage/csi-snapshotter：v7.0.2 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.10.0 • docker.io/netapp/trident-operator：24.06.0（オプション）

Astra Trident をインストール

Astra Tridentのインストール方法をご確認ください

ネットアップでは、Astra Tridentをさまざまな環境や組織に導入できるように、複数のインストールオプションを提供しています。Astra Tridentは、Tridentオペレータ（手動またはHelmを使用）またはでインストールできます `tridentctl`。このトピックでは、適切なインストールプロセスを選択するための重要な情報を提供します。

Astra Tridentに関する重要な情報24.06

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

- Astra TridentでKubernetes 1.31がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Astra Tridentでは、SAN環境でのマルチパス構成の使用が厳密に実施されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  マルチパス以外の構成を使用するか、multipath.confファイルにまたは `find_multipaths: smart`の値を使用する `find_multipaths: yes`と、マウントが失敗します。Tridentでは、21.07リリース以降での使用を推奨してい `find_multipaths: no`ます。

開始する前に

インストールパスに関係なく、次のものが必要です。

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

に慣れていない場合は"基本概念"、今すぐそれを行うのに最適な時期です。

インストール方法を選択します

適切なインストール方法を選択します。また、決定を下す前に、の考慮事項も確認しておく必要があります"メソッド間を移動しています"。

Trident演算子を使用する

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

Tridentオペレータには次のようなメリットがあります。

** Astra Tridentオブジェクト作成**

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

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

リソースアカウンタビリティ

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

**** 自己回復機能****

OperatorはAstra Tridentのインストールを監視し、導入が削除されたときや誤って変更された場合などの問題に対処するための手段をアクティブに講じます。 `trident-operator-<generated-id>`CRをAstra Tridentのインストール環境に関連付けるポッドが作成され `TridentOrchestrator`ます。これにより、クラスタ内にAstra Tridentのインスタンスが1つだけ存在し、そのセットアップを制御することで、インストールがべき等の状態であることを確認できます。インストールに変更が加えられると（展開またはノードのデミスタなど）、オペレータはそれらを識別し、個別に修正します。

**** は、インストール済みの既存の**** を簡単に更新できます

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

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、を `TridentOrchestrator` `true`次のように設定し `spec.debug`ます。

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

が更新されると `TridentOrchestrator`、オペレータは更新を処理し、既存のインストールにパッチを適用します。これにより、新しいポッドが作成され、それに応じてインストールが変更される可能性があります。

****クリーン再インストール****

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

** Kubernetes**の自動アップグレード処理****

Kubernetes バージョンのクラスタをサポート対象バージョンにアップグレードすると、オペレータが既存の 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。

使用方法 `tridentctl`

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

Tridentリソースのマニフェストを生成できます。導入、開始、サービスアカウント、 Astra Trident がインストールの一部として作成するクラスタロールが含まれます。

> 22.04 リリース以降、 Astra Trident がインストールされるたびに AES キーが再生成されなくなりました。今回のリリースでは、 Astra Trident がインストールする新しいシークレットオブジェクトが、インストール全体で維持されます。つまり、 `tridentctl`22.04では以前のバージョンのTridentをアンインストールできますが、以前のバージョンでは22.04のインストールをアンインストールできません。適切なインストール方法_を選択します。

インストールモードを選択します

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

標準インストール

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

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

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

オフラインインストール

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

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

リモートインストール

次に、リモートインストールプロセスの概要を示します。

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

メソッドとモードに基づいてプロセスを選択します

決定が終わったら、適切なプロセスを選択します。

方法	インストールモード
Tridentのオペレータ（手動）	"標準インストール" "オフラインインストール"

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

インストール方法を切り替える

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

- Astra Tridentのインストールとアンインストールには、常に同じ方法を使用します。を使用してを導入した場合は `tridentctl`、適切なバージョンのバイナリを使用してAstra Tridentをアンインストールする必要があります `tridentctl`。同様に、オペレータを使用して導入する場合は、CRを編集し、Astra Tridentをアンインストールするようにを設定する `spec.uninstall=true`必要があります `TridentOrchestrator`。
- オペレータベースの導入環境を削除してAstra Tridentの導入に使用する場合 `tridentctl`は、まずAstra Tridentを編集し、アンインストールするようにを設定する `spec.uninstall=true`必要があります `TridentOrchestrator`。次に、とオペレータの配置を削除し `TridentOrchestrator`ます。その後、を使用してをインストールできます `tridentctl`。
- 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。ベースのデプロイメントを使用している場合は `tridentctl`、問題なくHelmベースのデプロイメントを使用できます。

その他の既知の設定オプション

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

- クラスタが特権ワークロードをサポートしている必要があります。
- `--kubelet-dir`フラグは`kubelet`ディレクトリの場所に設定する必要があります。デフォルトはです `/var/vcap/data/kubelet`。

  を使用してkubeletの場所を指定する `--kubelet-dir`ことは、Trident Operator、Helm、および配置で機能することがわかってい `tridentctl`ます。

Tridentオペレータを使用してインストール

Tridentオペレータを手動で導入（標準モード）

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

は、を使用し"オフライン導入のプロセス"ます。

Astra Tridentに関する重要な情報24.06

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

- Astra TridentでKubernetes 1.31がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Astra Tridentでは、SAN環境でのマルチパス構成の使用が厳密に実施されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  マルチパス以外の構成を使用するか、multipath.confファイルにまたは `find_multipaths: smart`の値を使用する `find_multipaths: yes`と、マウントが失敗します。Tridentでは、21.07リリース以降での使用を推奨してい `find_multipaths: no`ます。

Tridentオペレータを手動で導入し、Tridentをインストール

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

開始する前に

インストールを開始する前に、Linuxホストにログインし、動作中のを管理していること、および必要なPrivilegesが揃っていることを確認し"サポートされる Kubernetes クラスタ"ます。

> OpenShiftでは、以降のすべての例での代わり `kubectl`にを使用し、または `oc login -u kube-admin`を `oc`実行して最初に* system:admin *としてログインし `oc login -u system:admin`ます。

1. Kubernetesのバージョンを確認します。

```
kubectl version
```

2. クラスタ管理者の権限を確認します。

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

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

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

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

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

```
wget https://github.com/NetApp/trident/releases/download/v24.06.0/trident-
installer-24.06.0.tar.gz
tar -xf trident-installer-24.06.0.tar.gz
cd trident-installer
```

手順**2**：**CRD**を作成する `TridentOrchestrator`

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

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

ステップ**3**：**Trident**オペレータを導入

Astra Tridentインストーラには、オペレータのインストールや関連オブジェクトの作成に使用できるバンドルファイルが用意されています。このバンドルファイルを使用すると、オペレータを簡単に導入し、デフォルトの設定でAstra Tridentをインストールできます。

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

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

開始する前に

- デフォルトでは、Tridentインストーラによってネームスペースにオペレータが配置され `trident`ます。ネームスペースが存在しない場合は `trident`、次のコマンドを使用して作成します。

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

- 名前空間以外の名前空間にオペレータを配置するには `trident`、 `clusterrolebinding.yaml`を使用してバンドルファイルを更新し `serviceaccount.yaml`、 `operator.yaml`生成します `kustomization.yaml`。

  a. 次のコマンドを使用して、<bundle.yaml> `bundle_pre_1_25.yaml` `bundle_post_1_25.yaml`のバージョンに基づいてを作成し `kustomization.yaml`ます。

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

  b. 次のコマンドWHERE__ ISまたは `bundle_post_1_25.yaml`使用している<bundle.yaml>のバージョンに基づいて、バンドルをコンパイルし `bundle_pre_1_25.yaml`ます。

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

手順

1. リソースを作成し、オペレータを配置します。

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

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

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

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

手順**4**：を作成 `**TridentOrchestrator**`して**Trident**をインストールする

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

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

kubectl describe torc trident

Name:        trident
Namespace:
Labels:      <none>
Annotations: <none>
API Version: trident.netapp.io/v1
Kind:        TridentOrchestrator
...
Spec:
  Debug:     true
  Namespace: trident
Status:
  Current Installation Params:
    IPv6:                      false
    Autosupport Hostname:
    Autosupport Image:         netapp/trident-autosupport:24.06
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                     true
    Image Pull Secrets:
    Image Registry:
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Silence Autosupport:  false
    Trident Image:        netapp/trident:24.06.0
  Message:                  Trident installed  Namespace:
trident
  Status:                   Installed
  Version:                  v24.06.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

インストールの確認

インストールを確認するには、いくつかの方法があります。

ステータスの使用 TridentOrchestrator

のステータス TridentOrchestrator`は、インストールが正常に完了したかどうかを示し、インストールされているTridentのバージョンを表示します。インストール中に、のステータス `TridentOrchestrator`がからに `Installed`変わります `Installing。ステータスを確認し、オペレータが単独で回復できない場合は Failed、を"ログをチェックしてください"参照してください。

ステータス	説明
インストール	オペレータはこのCRを使用してAstra Tridentをインストールしています TridentOrchestrator。
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	オペレータがAstra Tridentをアンインストールしています。 spec.uninstall=true
アンインストール済み	Astra Trident がアンインストールされました。
失敗	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	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	は `TridentOrchestrator`使用されません。別のファイルがすでに存在します。

ポッドの作成ステータスを使用する

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

```
kubectl get pods -n trident

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

使用方法 `tridentctl`

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

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 24.06.0        | 24.06.0        |
+----------------+----------------+
```

Tridentオペレータを手動で導入（オフラインモード）

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

Astra Tridentに関する重要な情報24.06

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

- Astra TridentでKubernetes 1.31がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Astra Tridentでは、SAN環境でのマルチパス構成の使用が厳密に実施されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  マルチパス以外の構成を使用するか、multipath.confファイルにまたは `find_multipaths: smart`の値を使用する `find_multipaths: yes`と、マウントが失敗します。Tridentでは、21.07リリース以降での使用を推奨してい `find_multipaths: no`ます。

Tridentオペレータを手動で導入し、Tridentをインストール

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

開始する前に

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

> OpenShiftでは、以降のすべての例での代わり `kubectl`にを使用し、または `oc login -u kube-admin`を `oc`実行して最初に* system:admin *としてログインし `oc login -u system:admin`ます。

1. Kubernetesのバージョンを確認します。

```
kubectl version
```

2. クラスタ管理者の権限を確認します。

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

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

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

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

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

```
wget https://github.com/NetApp/trident/releases/download/v24.06.0/trident-
installer-24.06.0.tar.gz
tar -xf trident-installer-24.06.0.tar.gz
cd trident-installer
```

手順**2**：**CRD**を作成する `TridentOrchestrator`

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

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

手順**3**：オペレータのレジストリの場所を更新します

で `/deploy/operator.yaml`、イメージレジストリの場所を反映するように更新し `image: docker.io/netapp/trident-operator:24.06.0` ます。は "TridentとCSIの画像" 1つのレジストリまたは別のレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。例：

- `image: <your-registry>/trident-operator:24.06.0` イメージがすべて1つのレジストリに格納されている場合。

- `image: <your-registry>/netapp/trident-operator:24.06.0` TridentイメージがCSIイメージとは別のレジストリにある場合。

ステップ**4**：**Trident**オペレータを導入

Astra Tridentインストーラには、オペレータのインストールや関連オブジェクトの作成に使用できるバンドルファイルが用意されています。このバンドルファイルを使用すると、オペレータを簡単に導入し、デフォルトの設定でAstra Tridentをインストールできます。

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

開始する前に

- デフォルトでは、Tridentインストーラによってネームスペースにオペレータが配置され `trident`ます。ネームスペースが存在しない場合は `trident`、次のコマンドを使用して作成します。

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

- 名前空間以外の名前空間にオペレータを配置するには `trident`、 `clusterrolebinding.yaml`を使用してバンドルファイルを更新し `serviceaccount.yaml`、 `operator.yaml`生成します `kustomization.yaml`。

  a. 次のコマンドを使用して、<bundle.yaml> `bundle_pre_1_25.yaml` `bundle_post_1_25.yaml`のバージョンに基づいてを作成し `kustomization.yaml`ます。

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

  b. 次のコマンドWHERE__ ISまたは `bundle_post_1_25.yaml`使用している<bundle.yaml>のバージョンに基づいて、バンドルをコンパイルし `bundle_pre_1_25.yaml`ます。

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

手順

1. リソースを作成し、オペレータを配置します。

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

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

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

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

手順**5**： `TridentOrchestrator`

は "TridentとCSIの画像" 1つのレジストリまたは別のレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。レジストリ構成に基づいて追加のロケーション仕様を追加するには、更新し `deploy/crds/tridentorchestrator_cr.yaml`ます。

1つのレジストリ内のイメージ

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:24.06"
tridentImage: "<your-registry>/trident:24.06.0"
```

異なるレジストリ内の画像

別のレジストリの場所を使用するには、をに `imageRegistry`追加する必要があります `sig-storage`。

```
imageRegistry: "<your-registry>/sig-storage"
autosupportImage: "<your-registry>/netapp/trident-autosupport:24.06"
tridentImage: "<your-registry>/netapp/trident:24.06.0"
```

手順**6**：を作成 `**TridentOrchestrator**`して**Trident**をインストールする

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

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

kubectl describe torc trident

Name:        trident
Namespace:
Labels:      <none>
Annotations: <none>
API Version: trident.netapp.io/v1
Kind:        TridentOrchestrator
...
Spec:
  Autosupport Image:  <your-registry>/netapp/trident-autosupport:24.06
  Debug:              true
  Image Registry:     <your-registry>/sig-storage
  Namespace:          trident
  Trident Image:      <your-registry>/netapp/trident:24.06.0
Status:
  Current Installation Params:
    IPv6:                       false
    Autosupport Hostname:
    Autosupport Image:          <your-registry>/netapp/trident-
autosupport:24.06
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                      true
    Http Request Timeout:       90s
    Image Pull Secrets:
    Image Registry:       <your-registry>/sig-storage
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Probe Port:           17546
    Silence Autosupport:  false
    Trident Image:        <your-registry>/netapp/trident:24.06.0
  Message:                Trident installed
  Namespace:              trident
  Status:                 Installed
  Version:                v24.06.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

インストールの確認

インストールを確認するには、いくつかの方法があります。

ステータスの使用 `TridentOrchestrator`

のステータス `TridentOrchestrator`は、インストールが正常に完了したかどうかを示し、インストールされている`Trident`のバージョンを表示します。インストール中に、のステータス `TridentOrchestrator`がからに `Installed`変わります `Installing`。ステータスを確認し、オペレータが単独で回復できない場合は `Failed`、を"ログをチェックしてください"参照してください。

ステータス	説明
インストール	オペレータはこのCRを使用してAstra Tridentをインストールしています `TridentOrchestrator`。
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	オペレータがAstra Tridentをアンインストールしています。 `spec.uninstall=true`
アンインストール済み	Astra Trident がアンインストールされました。
失敗	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	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	は `TridentOrchestrator`使用されません。別のファイルがすでに存在します。

ポッドの作成ステータスを使用する

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

```
kubectl get pods -n trident

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

使用方法 `tridentctl`

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

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 24.06.0        | 24.06.0        |
+----------------+----------------+
```

Helm（標準モード）を使用してTridentを導入

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

Astra Tridentに関する重要な情報24.06

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

- Astra TridentでKubernetes 1.31がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Astra Tridentでは、SAN環境でのマルチパス構成の使用が厳密に実施されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  マルチパス以外の構成を使用するか、multipath.confファイルにまたは `find_multipaths: smart`の値を使用する `find_multipaths: yes`と、マウントが失敗します。Tridentでは、21.07リリース以降での使用を推奨してい `find_multipaths: no`ます。

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

Tridentを使用すると、"Helmチャート"Tridentオペレータを導入し、Tridentをワンステップでインストールできます。

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

開始する前に

あなたが必要とする"Helm バージョン 3"ことに加えて"導入の前提条件"。

手順

1. Astra Trident Helmリポジトリを追加します。

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

2. を使用し `helm install` 、次の例のように環境の名前を指定します `100.2404.0` 。はインストールするAstra Tridentのバージョンです。

```
helm install <name> netapp-trident/trident-operator --version 100.2406.0
--create-namespace --namespace <trident-namespace>
```

> Trident用のネームスペースをすでに作成している場合、パラメータを使用する `--create-namespace`と追加のネームスペースは作成されません。

を使用して、インストールの詳細（名前、ネームスペース、グラフ、ステータス、アプリケーションのバージョンなど）を確認できます `helm list` 。 およびリビジョン番号。

インストール中に設定データを渡す

インストール中に設定データを渡すには、次の 2 つの方法があります。

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

たとえば、のデフォルト値を変更するには `debug`、次のコマンドを実行し `--set` ます。はインストールするAstra Tridentのバージョンです。 `100.2406.0`

```
helm install <name> netapp-trident/trident-operator --version 100.2406.0
--create-namespace --namespace trident --set tridentDebug=true
```

設定オプション

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

オプション	説明	デフォルト
`nodeSelector`	ポッド割り当てのノードラベル	
`podAnnotations`	ポッドの注釈	
`deploymentAnnotations`	配置のアノテーション	
`tolerations`	ポッド割り当ての許容値	
`affinity`	ポッド割り当てのアフィニティ	
`tridentControllerPluginNodeSelector`	ポッド用の追加のノードセレクタ。詳細については、を参照してください [コントローラポッドとノードポッドについて]。	
`tridentControllerPluginTolerations`	ポッドに対するKubernetesの許容範囲を上書きします。詳細については、を参照してください [コントローラポッドとノードポッドについて]。	
`tridentNodePluginNodeSelector`	ポッド用の追加のノードセレクタ。詳細については、を参照してください [コントローラポッドとノードポッドについて]。	
`tridentNodePluginTolerations`	ポッドに対するKubernetesの許容範囲を上書きします。詳細については、を参照してください [コントローラポッドとノードポッドについて]。	
`imageRegistry`	、、 `trident`およびその他のイメージのレジストリを指定します `trident-operator`。デフォルトをそのまま使用する場合は、空のままにします。	""
`imagePullPolicy`	のイメージプルポリシーを設定します `trident-operator`。	`IfNotPresent`

オプション	説明	デフォルト
imagePullSecrets	、、 trident`およびその他のイメージのプルシークレットを設定します `trident-operator。	
kubeletDir	kubeletの内部状態のホスト位置を上書きできます。	"/var/lib/kubelet"
operatorLogLevel	Trident演算子のログレベルを、、 debug info、、 warn、 `error`または `fatal`に設定 `trace`できます。	"info"
operatorDebug	Tridentオペレータのログレベルをdebugに設定できます。	true
operatorImage	のイメージを完全に上書きできます trident-operator。	""
operatorImageTag	イメージのタグを上書きできます trident-operator。	""
tridentIPv6	IPv6クラスタでAstra Tridentを動作させることができます。	false
tridentK8sTimeout	ほとんどのKubernetes API処理でデフォルトの30秒タイムアウトを上書きします（0以外の場合は秒単位）。	0
tridentHttpRequestTimeout	HTTP要求のデフォルトの90秒タイムアウトを上書きします。タイムアウト時間は無制限です。 `0s`負の値は使用できません。	"90s"
tridentSilenceAutosupport	Astra Tridentの定期的なAutoSupport レポートを無効にできます。	false
tridentAutosupportImageTag	Astra Trident AutoSupport コンテナのイメージのタグを上書きできます。	<version>
tridentAutosupportProxy	Astra TridentのAutoSupport コンテナがHTTPプロキシ経由で自宅に通信できるようになります。	""
tridentLogFormat	Astra Tridentのロギング形式または `json`を設定し(`text`ます）。	"text"
tridentDisableAuditLog	Astra Trident監査ロガーを無効にします。	true
tridentLogLevel	Astra Tridentのログレベルを、、 debug info、、 warn、、 error`またはに `fatal`設定できます `trace。	"info"
tridentDebug	Astra Tridentのログレベルをに設定できます debug。	false
tridentLogWorkflows	特定のAstra Tridentワークフローを有効にして、トレースロギングやログ抑制を実行できます。	""
tridentLogLayers	特定のAstra Tridentレイヤでトレースロギングやログ抑制を有効にできます。	""
tridentImage	Astra Tridentのイメージを完全に上書きできます。	""
tridentImageTag	Astra Tridentのイメージのタグを上書きできます。	""

オプション	説明	デフォルト
`tridentProbePort`	Kubernetesの活性/準備プローブに使用されるデフォルトポートを上書きできます。	""
`windows`	WindowsワーカーノードにAstra Tridentをインストールできます。	`false`
`enableForceDetach`	強制切り離し機能を有効にできます。	`false`
`excludePodSecurityPolicy`	オペレータポッドのセキュリティポリシーを作成から除外します。	`false`
`cloudProvider`	AKSクラスタで管理IDまたはクラウドIDを使用する場合はに設定します `"Azure"`。EKSクラスタでクラウドIDを使用する場合は、「aws」に設定します。	""
`cloudIdentity`	AKSクラスタでクラウドIDを使用する場合は、ワークロードID（「azure.workload.identity/client-id：xxxxxxxxxx-xxxx-xxxxxxx」）に設定します。EKSクラスタでクラウドIDを使用する場合は、AWS IAMロール（「eks.amazonaws.com/role-arn: arn：aws：iam：：123456：role/astratrident-role」）に設定されます。	""
`iscsiSelfHealingInterval`	iSCSIの自己修復が実行される間隔。	`5m0s`
`iscsiSelfHealingWaitTime`	iSCSIの自己修復が、ログアウトとその後のログインを実行して古いセッションの解決を開始するまでの時間。	`7m0s`

コントローラポッドとノードポッドについて

Astra Tridentは、単一のコントローラポッドと、クラスタ内の各ワーカーノード上のノードポッドとして実行されます。Astra Tridentボリュームをマウントするすべてのホストでノードポッドが実行されている必要があります。

Kubernetes"ノードセレクタ"を使用して、"寛容さと汚れ"ポッドを特定のノードまたは優先ノードで実行するように制限します。「ControllerPlugin」とを使用して、 `NodePlugin`制約とオーバーライドを指定できます。

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

Helm（オフラインモード）を使用したTridentのオペレータの導入

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

Astra Tridentに関する重要な情報24.06

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

- Astra TridentでKubernetes 1.31がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Astra Tridentでは、SAN環境でのマルチパス構成の使用が厳密に実施されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  マルチパス以外の構成を使用するか、multipath.confファイルにまたは `find_multipaths: smart`の値を使用する `find_multipaths: yes`と、マウントが失敗します。Tridentでは、21.07リリース以降での使用を推奨してい `find_multipaths: no`ます。

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

Tridentを使用すると、"Helmチャート"Tridentオペレータを導入し、Tridentをワンステップでインストールできます。

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

開始する前に

あなたが必要とする"Helm バージョン 3"ことに加えて"導入の前提条件"。

手順

1. Astra Trident Helmリポジトリを追加します。

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

2. を使用し `helm install` 、展開とイメージレジストリの場所の名前を指定します。は "TridentとCSIの画像" 1つのレジストリまたは別のレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。この例では `100.2406.0` 、インストールするAstra Tridentのバージョンを示しています。

1つのレジストリ内のイメージ

```
helm install <name> netapp-trident/trident-operator --version
100.2406.0 --set imageRegistry=<your-registry> --create-namespace
--namespace <trident-namespace>
```

異なるレジストリ内の画像

別のレジストリの場所を使用するには、をに `imageRegistry`追加する必要があります `sig-storage`。

```
helm install <name> netapp-trident/trident-operator --version
100.2406.0 --set imageRegistry=<your-registry>/sig-storage --set
operatorImage=<your-registry>/netapp/trident-operator:24.06.0 --set
tridentAutosupportImage=<your-registry>/netapp/trident-
autosupport:24.06 --set tridentImage=<your-
registry>/netapp/trident:24.06.0 --create-namespace --namespace
<trident-namespace>
```

Trident用のネームスペースをすでに作成している場合、パラメータを使用する `--create-namespace`と追加のネームスペースは作成されません。

を使用して、インストールの詳細（名前、ネームスペース、グラフ、ステータス、アプリケーションのバージョンなど）を確認できます `helm list`。 およびリビジョン番号。

インストール中に設定データを渡す

インストール中に設定データを渡すには、次の 2 つの方法があります。

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

たとえば、のデフォルト値を変更するには `debug`、次のコマンドを実行し `--set` ます。はインストールするAstra Tridentのバージョンです。 `100.2406.0`

```
helm install <name> netapp-trident/trident-operator --version 100.2406.0
--create-namespace --namespace trident --set tridentDebug=true
```

設定オプション

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

オプション	説明	デフォルト
`nodeSelector`	ポッド割り当てのノードラベル	
`podAnnotations`	ポッドの注釈	
`deploymentAnnotations`	配置のアノテーション	
`tolerations`	ポッド割り当ての許容値	
`affinity`	ポッド割り当てのアフィニティ	
`tridentControllerPluginNodeSelector`	ポッド用の追加のノードセレクタ。詳細については、を参照してください "コントローラポッドとノードポッドについて"。	
`tridentControllerPluginTolerations`	ポッドに対するKubernetesの許容範囲を上書きします。詳細については、を参照してください "コントローラポッドとノードポッドについて"。	
`tridentNodePluginNodeSelector`	ポッド用の追加のノードセレクタ。詳細については、を参照してください "コントローラポッドとノードポッドについて"。	
`tridentNodePluginTolerations`	ポッドに対するKubernetesの許容範囲を上書きします。詳細については、を参照してください "コントローラポッドとノードポッドについて"。	
`imageRegistry`	、、 `trident`およびその他のイメージのレジストリを指定します `trident-operator`。デフォルトをそのまま使用する場合は、空のままにします。	""
`imagePullPolicy`	のイメージプルポリシーを設定します `trident-operator`。	`IfNotPresent`
`imagePullSecrets`	、、 `trident`およびその他のイメージのプルシークレットを設定します `trident-operator`。	
`kubeletDir`	kubeletの内部状態のホスト位置を上書きできます。	`"/var/lib/kubelet"`
`operatorLogLevel`	Trident演算子のログレベルを、、 `debug` `info`、、 `warn`、 `error`または `fatal`に設定 `trace`できます。	`"info"`

オプション	説明	デフォルト
operatorDebug	Tridentオペレータのログレベルをdebugに設定できます。	true
operatorImage	のイメージを完全に上書きできます trident-operator。	""
operatorImageTag	イメージのタグを上書きできます trident-operator。	""
tridentIPv6	IPv6クラスタでAstra Tridentを動作させることができます。	false
tridentK8sTimeout	ほとんどのKubernetes API処理でデフォルトの30秒タイムアウトを上書きします（0以外の場合は秒単位）。	0
tridentHttpRequestTimeout	HTTP要求のデフォルトの90秒タイムアウトを上書きします。タイムアウト時間は無制限です。 `0s`負の値は使用できません。	"90s"
tridentSilenceAutosupport	Astra Tridentの定期的なAutoSupport レポートを無効にできます。	false
tridentAutosupportImageTag	Astra Trident AutoSupport コンテナのイメージのタグを上書きできます。	<version>
tridentAutosupportProxy	Astra TridentのAutoSupport コンテナがHTTPプロキシ経由で自宅に通信できるようになります。	""
tridentLogFormat	Astra Tridentのロギング形式または `json`を設定し(`text`ます）。	"text"
tridentDisableAuditLog	Astra Trident監査ロガーを無効にします。	true
tridentLogLevel	Astra Tridentのログレベルを、、 debug info、、 warn、、 error`またはに `fatal`設定できます `trace。	"info"
tridentDebug	Astra Tridentのログレベルをに設定できます debug。	false
tridentLogWorkflows	特定のAstra Tridentワークフローを有効にして、トレースロギングやログ抑制を実行できます。	""
tridentLogLayers	特定のAstra Tridentレイヤでトレースロギングやログ抑制を有効にできます。	""
tridentImage	Astra Tridentのイメージを完全に上書きできます。	""

オプション	説明	デフォルト
`tridentImageTag`	Astra Tridentのイメージのタグを上書きできます。	""
`tridentProbePort`	Kubernetesの活性/準備プローブに使用されるデフォルトポートを上書きできます。	""
`windows`	WindowsワーカーノードにAstra Tridentをインストールできます。	`false`
`enableForceDetach`	強制切り離し機能を有効にできます。	`false`
`excludePodSecurityPolicy`	オペレータポッドのセキュリティポリシーを作成から除外します。	`false`

Tridentオペレータのインストールをカスタマイズ

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

コントローラポッドとノードポッドについて

Astra Tridentは、単一のコントローラポッドと、クラスタ内の各ワーカーノード上のノードポッドとして実行されます。Astra Tridentボリュームをマウントするすべてのホストでノードポッドが実行されている必要があります。

Kubernetes"ノードセレクタ"を使用して、"寛容さと汚れ"ポッドを特定のノードまたは優先ノードで実行するように制限します。「ControllerPlugin」とを使用して、 `NodePlugin`制約とオーバーライドを指定できます。

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

設定オプション

> `spec.namespace`は `TridentOrchestrator`、Astra Tridentがインストールされているネームスペースを示します。このパラメータ * は、 Astra Trident のインストール後に更新できません * 。これを実行しようとすると、 `TridentOrchestrator`ステータスがに変わり `Failed`ます。Astra Tridentは、ネームスペース間での移行を意図していません。

このテーブルは、属性の詳細を示し `TridentOrchestrator`ます。

パラメータ	説明	デフォルト
`namespace`	Astra Trident をインストールするネームスペース	`"default"`
`debug`	Astra Trident のデバッグを有効にします	`false`

パラメータ	説明	デフォルト
`enableForceDetach`	`ontap-san`そして `ontap-san-economy`のみ。KubernetesのNon-Graceful Node Shutdown（NGN）と連携して、ノードに障害が発生した場合に、マウントされたボリュームを含むワークロードを新しいノードに安全に移行する機能をクラスタ管理者に提供します。	`false`
`windows`	をに設定する `true`と、Windowsワーカーノードへのインストールが有効になります。	`false`
`cloudProvider`	AKSクラスタで管理IDまたはクラウドIDを使用する場合はに設定します `"Azure"`。EKSクラスタでクラウドIDを使用する場合は、「aws」に設定します。	`""`
`cloudIdentity`	AKSクラスタでクラウドIDを使用する場合は、ワークロードID（「azure.workload.identity/client-id：xxxxxxxxxx-xxxx-xxxxxxx」）に設定します。EKSクラスタでクラウドIDを使用する場合は、AWS IAMロール（「eks.amazonaws.com/role-arn: arn：aws：iam：：123456：role/astratrident-role」）に設定されます。	`""`
`IPv6`	IPv6 経由の Astra Trident をインストール	正しくない
`k8sTimeout`	Kubernetes 処理のタイムアウト	`30sec`
`silenceAutosupport`	AutoSupport バンドルをネットアップに自動的に送信しない	`false`
`autosupportImage`	AutoSupport テレメトリのコンテナイメージ	`"netapp/trident-autosupport:24.06"`
`autosupportProxy`	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	`false`
`logFormat`	Astra Trident のログ形式が使用 [text 、 JSON]	`"text"`
`tridentImage`	インストールする Astra Trident イメージ	`"netapp/trident:24.06"`
`imageRegistry`	内部レジストリへのパス（形式） `<registry FQDN>[:port][/subpath]`	`"k8s.gcr.io/sig-storage"`（Kubernetes 1.19以降）または `"quay.io/k8scsi"`
`kubeletDir`	ホスト上の kubelet ディレクトリへのパス	`"/var/lib/kubelet"`
`wipeout`	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
`imagePullPolicy`	Tridentオペレータのイメージプルポリシーを設定します。有効な値は次のとおりです。常に画像をプルします。 `Always` `IfNotPresent`ノードにイメージが存在しない場合にのみ取得します。 `Never`画像を引っ張らないようにしました	`IfNotPresent`
`controllerPluginNodeSelector`	ポッド用の追加のノードセレクタ。の形式はと同じ`pod.spec.nodeSelector`です。	デフォルトはありません。オプションです
`controllerPluginTolerations`	ポッドに対するKubernetesの許容範囲を上書きします。はと同じ形式`pod.spec.Tolerations`です。	デフォルトはありません。オプションです
`nodePluginNodeSelector`	ポッド用の追加のノードセレクタ。の形式はと同じ`pod.spec.nodeSelector`です。	デフォルトはありません。オプションです
`nodePluginTolerations`	ポッドに対するKubernetesの許容範囲を上書きします。はと同じ形式`pod.spec.Tolerations`です。	デフォルトはありません。オプションです

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

フォースデタッチの詳細

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

Astra TridentはKubernetes NGNに依存しているため、許容できないすべてのワークロードのスケジュールが再設定されるまで、正常でないノードからテイントを削除しないで`out-of-service`ください。汚染を無謀に適用または削除すると、バックエンドのデータ保護が危険にさらされる可能性があります。

Kubernetesクラスタ管理者がtaintをノードに適用し、 `enableForceDetach`をに設定する `true`と`node.kubernetes.io/out-of-service=nodeshutdown:NoExecute`、Astra Tridentによってノードのステータスが確認され、次の処理が実行されます。

1. そのノードにマウントされたボリュームのバックエンドI/Oアクセスを停止します。
2. Astra Tridentノードオブジェクトを（新しいドキュメントに対しては安全ではない）とマークします `dirty`。

Tridentコントローラは、Tridentノードポッドによって（とマークされた後で）ノードが再修飾されるまで、新しいパブリッシュボリューム要求を拒否し`dirty`ます。マウントされたPVCでスケジュールされているワークロード（クラスタノードが正常で準備が完了したあとも含む）は、`Astra Trident`がノードを検証できる（新しいドキュメントに対応）まで受け付けられません `clean`。

ノードの健常性が回復してtaintが削除されると、Astra Tridentは次の処理を実行します。

1. ノード上の古い公開パスを特定してクリーンアップします。
2. ノードが状態で（サービス停止状態が削除され、ノードが状態である `Ready`）、古い公開パスがすべてクリーンである場合、 `cleanable`Astra Tridentはノードをとして再登録し、新しい公開ボリュームをノードに許可します `clean`。

構成例

を定義してインストールをカスタマイズするときに、 `TridentOrchestrator`の属性を使用できます[設定オプション]。

基本的なカスタム設定

これは、基本的なカスタムインストールの例です。

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

ノードセレクタ

この例では、Astra Tridentとノードセレクタをインストールします。

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

Windowsワーカーノード

この例では、WindowsワーカーノードにAstra Tridentをインストールします。

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

AKSクラスタ上の管理対象**ID**

この例では、AKSクラスタで管理対象IDを有効にするためにAstra Tridentをインストールします。

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

AKSクラスタ上のクラウド**ID**

この例では、AKSクラスタにクラウドIDで使用するAstra Tridentをインストールします。

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

EKSクラスタ上のクラウド**ID**

この例では、AKSクラスタにクラウドIDで使用するAstra Tridentをインストールします。

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  cloudProvider: "AWS"
  cloudIdentity: "'eks.amazonaws.com/role-arn:
arn:aws:iam::123456:role/astratrident-role'"
```

tridentctlを使用してインストールします

tridentctlを使用してインストールします

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

Astra Tridentに関する重要な情報**24.06**

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

- TridentでKubernetes 1.27がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Astra Tridentでは、SAN環境でのマルチパス構成の使用が厳密に実施されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  マルチパス以外の構成を使用するか、multipath.confファイルにまたは `find_multipaths: smart`の値を使用する `find_multipaths: yes`と、マウントが失敗します。Tridentでは、21.07リリース以降での使用を推奨してい `find_multipaths: no`ます。

以下を使用して**Astra Trident**をインストール `tridentctl`

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

開始する前に

インストールを開始する前に、Linuxホストにログインし、動作中のを管理していること、および必要なPrivilegesが揃っていることを確認し"サポートされる Kubernetes クラスタ"ます。

OpenShiftでは、以降のすべての例での代わり `kubectl`にを使用し、または `oc login -u kube-admin`を `oc`実行して最初に* system:admin *としてログインし `oc login -u system:admin`ます。

1. Kubernetesのバージョンを確認します。

```
kubectl version
```

2. クラスタ管理者の権限を確認します。

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

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

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

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

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

```
wget https://github.com/NetApp/trident/releases/download/v24.06.0/trident-
installer-24.06.0.tar.gz
tar -xf trident-installer-24.06.0.tar.gz
cd trident-installer
```

手順**2**：**Astra Trident**をインストールする

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

標準モード

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

1つのレジストリ内のイメージ

```
./tridentctl install -n trident --image-registry <your-registry>
--autosupport-image <your-registry>/trident-autosupport:24.06 --trident
-image <your-registry>/trident:24.06.0
```

異なるレジストリ内の画像

別のレジストリの場所を使用するには、をに `imageRegistry`追加する必要があります `sig-storage`。

```
./tridentctl install -n trident --image-registry <your-registry>/sig-
storage --autosupport-image <your-registry>/netapp/trident-
autosupport:24.06 --trident-image <your-
registry>/netapp/trident:24.06.0
```

インストールステータスは次のようになります。

```
....
INFO Starting Trident installation.                namespace=trident
INFO Created service account.
INFO Created cluster role.
INFO Created cluster role binding.
INFO Added finalizers to custom resource definitions.
INFO Created Trident service.
INFO Created Trident secret.
INFO Created Trident deployment.
INFO Created Trident daemonset.
INFO Waiting for Trident pod to start.
INFO Trident pod started.                          namespace=trident
pod=trident-controller-679648bd45-cv2mx
INFO Waiting for Trident REST interface.
INFO Trident REST interface is up.                 version=24.06.0
INFO Trident installation succeeded.
....
```

インストールの確認

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

ポッドの作成ステータスを使用する

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

```
kubectl get pods -n trident

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

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

使用方法 `tridentctl`

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

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 24.06.0        | 24.06.0        |
+----------------+----------------+
```

構成例

次の例は、を使用してAstra Tridentをインストールするための設定例 `tridentctl`です。

Windowsノオト

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

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

強制的に切り離し

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

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

tridentctlのインストールをカスタマイズします

Astra Tridentインストーラを使用して、インストールをカスタマイズできます。

インストーラの詳細を確認してください

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

Kubernetesのディストリビューションを使用していて、でデータが通常以外のパスに保持される `/var/lib/kubelet`場合は `kubelet`、を使用して代替パスを指定できます `--kubelet-dir`。

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

- `trident-clusterrolebinding.yaml`
- `trident-deployment.yaml`
- `trident-crds.yaml`
- `trident-clusterrole.yaml`
- `trident-daemonset.yaml`
- `trident-service.yaml`
- `trident-namespace.yaml`
- `trident-serviceaccount.yaml`
- `trident-resourcequota.yaml`

これらのファイルを生成したら、必要に応じてファイルを変更し、を使用してカスタム配置をインストールできます `--use-custom-yaml`。

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

Astra Trident を使用

ワーカーノードを準備します

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

適切なツールを選択する

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

NFSツール

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

iSCSIツール

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

NVMeツール

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

NVMe/TCPにはONTAP 9.12以降を推奨します。

ノードサービスの検出

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

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

イベントを確認します

Astra Tridentが、検出されたサービスを特定するためのイベントをノードに対して作成次のイベントを確認するには、を実行します。

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

検出されたサービスを確認

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

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

NFSボリューム

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

RHEL 8以降

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

Ubuntu

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

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

iSCSI ボリューム

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

iSCSIの自己回復機能

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

1. *希望するiSCSIセッションの状態と現在のiSCSIセッションの状態を識別します
2. *希望する状態と現在の状態を比較して、必要な修理を特定します。Astra Tridentが、修理の優先順位と、修理に先手を打つタイミングを判断
3. *現在のiSCSIセッションの状態を希望するiSCSIセッションの状態に戻すために必要な修復*を実行します。

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

Astra Tridentの自動修復機能は、次のような問題を防止します。

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

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

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

iSCSIツールをインストール

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

開始する前に

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

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

- iSCSI PVSでRHEL / RedHat CoreOSを実行するワーカーノードを使用する場合は、StorageClassでmountOptionを指定し `discard`てインラインのスペース再生を実行します。を参照してください "Red Hat のドキュメント"。

RHEL 8以降

1. 次のシステムパッケージをインストールします。

```
sudo yum install -y lsscsi iscsi-initiator-utils sg3_utils device-
mapper-multipath
```

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

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

3. スキャンを手動に設定：

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

4. マルチパスを有効化：

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

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

5. および `multipathd`が実行されていることを確認し `iscsid`ます。

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

6. 有効にして開始 `iscsi`：

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

Ubuntu

1. 次のシステムパッケージをインストールします。

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

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

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

3. スキャンを手動に設定：

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

4. マルチパスを有効化：

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

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

5. とが `multipath-tools`有効で実行されていることを確認し `open-iscsi`ます。

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

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

iSCSI自己回復の設定または無効化

次のAstra TridentのiSCSI自己修復設定を使用して、古いセッションを修正できます。

- * iSCSIの自己修復間隔*：iSCSIの自己修復を実行する頻度を指定します（デフォルト：5分）。小さい数値を設定することで実行頻度を高めるか、大きい数値を設定することで実行頻度を下げることができます。

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

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

Helm

iSCSIの自己修復設定を設定または変更するには、Helmのインストール時またはHelmの更新時にパラメータと `iscsiSelfHealingWaitTime`パラメータを渡します `iscsiSelfHealingInterval`。

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

```
helm install trident trident-operator-100.2406.0.tgz --set
iscsiSelfHealingInterval=3m0s --set iscsiSelfHealingWaitTime=6m0s -n
trident
```

Tridentctl

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

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

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

NVMe/TCPホリユウム

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

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

RHEL 9

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

Ubuntu

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

インストールの確認

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

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

Astra Tridentでは、NVMeがダウンしたときにパスがあきらめないように値を変更し `ctrl_device_tmo`ます。この設定は変更しないでください。

バックエンドの構成と管理

バックエンドを設定

バックエンドは、 Astra Trident とストレージシステムの関係を定義します。Trident がストレージシステムとの通信方法を Trident から指示し、 Astra Trident がボリュームをプロビジョニングする方法も解説します。

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

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

Azure NetApp Files

Azure NetApp Files バックエンドを設定します

Azure NetApp FilesはAstra Tridentのバックエンドとして設定できます。Azure NetApp Filesバックエンドを使用してNFSボリュームとSMBボリュームを接続できます。Astra Tridentでは、Azure Kubernetes Services（AKS）クラスタの管理対象IDを使用したクレデンシャル管理もサポートされます。

Azure NetApp Filesドライバの詳細

Astra Tridentは、次のAzure NetApp Filesストレージドライバを使用してクラスタと通信します。サポートされているアクセスモードは、*ReadWriteOnce*(RWO)、*ReadOnlyMany*(ROX)、*ReadWriteMany*(RWX)、*ReadWriteOncePod*(RWOP)です。

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

考慮事項

- Azure NetApp Files サービスでは、100GB未満のボリュームはサポートされません。容量の小さいボリュームが要求されると、Astra Tridentによって自動的に100GiBのボリュームが作成されます。
- Astra Tridentは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみをサポート

AKSの管理対象ID

Astra TridentはAzure Kubernetes Servicesクラスタでサポートされます"管理対象ID"。管理されたアイデンティティによって提供される合理的なクレデンシャル管理を利用するには、次のものが必要です。

- AKSを使用して導入されるKubernetesクラスタ
- AKS Kubernetesクラスタに設定された管理対象ID
- 指定する `"Azure"` を含む `Astra Trident` がインストールされます `cloudProvider`。

Trident オペレータ

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

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

Helm

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

```
helm install trident trident-operator-100.2406.0.tgz --create
-namespace --namespace <trident-namespace> --set cloudProvider=$CP
```

<code> tridentctl </code>

次の例では、Astra Tridentをインストールし、フラグをに `Azure`設定してい `cloudProvider`ます。

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

AKSのクラウドID

クラウドIDを使用すると、Kubernetesポッドは、明示的なAzureクレデンシャルを指定するのではなく、ワークロードIDとして認証することでAzureリソースにアクセスできます。

Azureでクラウドを活用するには、以下が必要です。

- AKSを使用して導入されるKubernetesクラスタ
- AKS KubernetesクラスタにワークロードIDとoidc-issuer
- ワークロードIDを指定し `"Azure"`て `cloudIdentity`指定するを含むAstra Tridentをインストール `cloudProvider`

Trident オペレータ

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

例：

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

Helm

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

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

次の例では、Astra Tridentをインストールし、環境変数を使用してをAzureに `$CP`設定し `cloudProvider`、を使用して環境変数を `$CI`設定してい `cloudIdentity`ます。

```
helm install trident trident-operator-100.2406.0.tgz --set
cloudProvider=$CP --set cloudIdentity=$CI
```

<code> tridentctl </code>

次の環境変数を使用して、* cloud provider フラグと cloud identity *フラグの値を設定します。

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

次の例では、Astra Tridentをインストールし、フラグを `$CP`、 `cloud-identity`に `$CI`設定してい `cloud-provider`ます。

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

Azure NetApp Files バックエンドを設定する準備をします

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

NFSボリュームと**SMB**ボリュームの前提条件

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

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

- `subscriptionID`、 `tenantID`、 `clientID`、、 `location`およびは、 `clientSecret`AKSクラスタで管理対象IDを使用する場合はオプションです。
- `tenantID`、 `clientID`、およびは、 `clientSecret`AKSクラスタでクラウドIDを使用する場合はオプションです。

- 容量プール。を参照してください "Microsoft：Azure NetApp Files 用の容量プールを作成します"。
- Azure NetApp Files に委任されたサブネット。を参照してください "Microsoft：サブネットをAzure NetApp Files に委任します"。
- `subscriptionID`Azure NetApp Filesを有効にしたAzureサブスクリプションから削除します。
- `tenantID` `clientID` `clientSecret`Azure NetApp Filesサービスへの十分な権限を持つ、Azure Active Directory内のから"アプリケーション登録"。アプリケーション登録では、次のいずれかを使用します。
  - 所有者ロールまたは寄与者ロール"Azureで事前定義"。
  - "カスタム投稿者ロール"(`assignableScopes`次の権限が付与されます。権限はAstra Tridentで必要な権限のみに制限されます。カスタムロールを作成したら、"Azureポータルを使用してロールを割り当てます"を参照してください。

カスタムコントリビュータロール

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- 少なくとも1つを含む "委任されたサブネット"Azure `location`。Trident 22.01では、この `location`パラメータはバックエンド構成ファイルの最上位レベルにある必須フィールドです。仮想プールで指定された場所の値は無視されます。
- を使用するに `Cloud Identity`は、から "ユーザーが割り当てた管理ID"を取得し `client ID`、でそのIDを指定します `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`。

SMBボリュームに関するその他の要件

SMBボリュームを作成するには、以下が必要です。

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

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

- Windowsサービスとして設定されたCSIプロキシ。を設定するには `csi-proxy`、Windowsで実行されているKubernetesノードについて、またはを"GitHub: Windows向けCSIプロキシ"参照してください"GitHub: CSIプロキシ"。

Azure NetApp Files バックエンド構成のオプションと例

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

バックエンド構成オプション

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

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

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

パラメータ	説明	デフォルト
`version`		常に 1
`storageDriverName`	ストレージドライバの名前	「 azure-NetApp-files 」
`backendName`	カスタム名またはストレージバックエンド	ドライバ名 + "_" + ランダムな文字
`subscriptionID`	AzureサブスクリプションからのサブスクリプションID管理されたIDがAKSクラスタで有効になっている場合はオプションです。	
`tenantID`	AKSクラスタで管理IDまたはクラウドIDが使用されている場合は、アプリ登録からのテナントIDはオプションです。	
`clientID`	管理対象IDまたはクラウドIDがAKSクラスタで使用されている場合、アプリ登録からのクライアントIDはオプションです。	
`clientSecret`	アプリ登録からのクライアントシークレット管理されたIDまたはクラウドIDがAKSクラスタで使用されている場合はオプションです。	
`serviceLevel`	、 `Premium`または `Ultra`のいずれか `Standard`	"" （ランダム）
`location`	新しいボリュームが作成されるAzureの場所の名前AKSクラスタで管理IDが有効になっている場合はオプションです。	

パラメータ	説明	デフォルト
resourceGroups	検出されたリソースをフィルタリングするためのリソースグループのリスト	"[]" （フィルタなし）
netappAccounts	検出されたリソースをフィルタリングするためのネットアップアカウントのリスト	"[]" （フィルタなし）
capacityPools	検出されたリソースをフィルタリングする容量プールのリスト	"[]" （フィルタなし、ランダム）
virtualNetwork	委任されたサブネットを持つ仮想ネットワークの名前	""
subnet	委任先のサブネットの名前 Microsoft.Netapp/volumes	""
networkFeatures	ボリュームのVNet機能のセットは Basic、または `Standard`です。ネットワーク機能は一部の地域では使用できず、サブスクリプションで有効にする必要がある場合があります。この機能を有効にしないタイミングを指定する `networkFeatures`と、ボリュームのプロビジョニングが失敗します。	""
nfsMountOptions	NFS マウントオプションのきめ細かな制御。SMBボリュームでは無視されます。NFSバージョン4.1を使用してボリュームをマウントするには、カンマで区切ったマウントオプションのリストにを追加してNFS v4.1を `nfsvers=4`選択します。ストレージクラス定義で設定されたマウントオプションは、バックエンド構成で設定されたマウントオプションよりも優先されます。	"nfsvers=3 "
limitVolumeSize	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します	"" （デフォルトでは適用されません）
debugTraceFlags	トラブルシューティング時に使用するデバッグフラグ。例： `\{"api": false, "method": true, "discovery": true}`トラブルシューティングを行って詳細なログダンプが必要な場合を除き、このオプションは使用しないでください。	null

パラメータ	説明	デフォルト
`nasType`	NFSボリュームまたはSMBボリュームの作成を設定オプションは `nfs`、 `smb`またはnullです。nullに設定すると、デフォルトでNFSボリュームが使用されます。	`nfs`
`supportedTopologies`	このバックエンドでサポートされているリージョンとゾーンのリストを表します。詳細については、を参照してください "CSI トポロジを使用します"。	

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

必要な権限とリソース

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

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

`resourceGroups` `netappAccounts`、、および `capacityPools`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、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。

タイプ	形式
リソースグループ	< リソースグループ >
ネットアップアカウント	< リソースグループ >/< ネットアップアカウント >
容量プール	< リソースグループ >/< ネットアップアカウント >/< 容量プール >
仮想ネットワーク	< リソースグループ >/< 仮想ネットワーク >
サブネット	<resource group>/< 仮想ネットワーク >/< サブネット >

ボリュームのプロビジョニング

構成ファイルの特別なセクションで次のオプションを指定することで、デフォルトのボリュームプロビジョニ

ングを制御できます。詳細については、を参照してください [構成例] 。

パラメータ	説明	デフォルト
`exportRule`	新しいボリュームに対するエクスポートルール `exportRule`IPv4アドレスまたはIPv4サブネットをCIDR表記で任意に組み合わせたリストをカンマで区切って指定する必要があります。SMBボリュームでは無視されます。	"0.0.0.0/0 "
`snapshotDir`	.snapshot ディレクトリの表示を制御します	いいえ
`size`	新しいボリュームのデフォルトサイズ	"100G"
`unixPermissions`	新しいボリュームのUNIX権限（8進数の4桁）。SMBボリュームでは無視されます。	"" （プレビュー機能、サブスクリプションでホワイトリスト登録が必要）

構成例

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

最小限の構成

これは、バックエンドの絶対的な最小構成です。この構成では、Astra Tridentが設定された場所のAzure NetApp Filesに委譲されたすべてのNetAppアカウント、容量プール、サブネットを検出し、それらのプールとサブネットの1つに新しいボリュームをランダムに配置します。は省略されているため、`nasType` `nfs` デフォルトが適用され、バックエンドでNFSボリュームがプロビジョニングされます。

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

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

AKSの管理対象**ID**

このバックエンド構成では、、 `tenantID`、 `clientID`、が `clientSecret`省略されてい `subscriptionID` ます。これらは、管理対象IDを使用する場合はオプションです。

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  capacityPools: ["ultra-pool"]
  resourceGroups: ["aks-ami-eastus-rg"]
  netappAccounts: ["smb-na"]
  virtualNetwork: eastus-prod-vnet
  subnet: eastus-anf-subnet
```

AKSのクラウド**ID**

このバックエンド構成では、クラウドIDを使用する場合はオプションである、、 `clientID`、が `clientSecret`省略されて `tenantID`います。

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

容量プールフィルタを使用した特定のサービスレベル構成

このバックエンド構成では、容量プール内のAzureの場所 `Ultra`にボリュームが配置され `eastus`ます。Astra Tridentは、その場所のAzure NetApp Filesに委譲されているすべてのサブネットを自動的に検出し、そのいずれかに新しいボリュームをランダムに配置します。

```
---
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
```

高度な設定

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

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

仮想プール構成

このバックエンド構成では、 1 つのファイルに複数の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。に基づいてプールを区別するために、仮想プールラベルが使用されました `performance`。

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

サポートされるトポロジ構成

Astra Tridentを使用すると、リージョンやアベイラビリティゾーンに基づいてワークロード用のボリュームを簡単にプロビジョニングできます。 `supportedTopologies`このバックエンド構成のブロックは、バックエンドごとにリージョンとゾーンのリストを提供するために使用されます。ここで指定するリージョンとゾーンの値は、各Kubernetesクラスタノードのラベルのリージョンとゾーンの値と一致している必要があります。これらのリージョンとゾーンは、ストレージクラスで指定できる許容値のリストです。バックエンドで提供されるリージョンとゾーンのサブセットを含むストレージクラスの場合、Astra Tridentは該当するリージョンとゾーンにボリュームを作成します。詳細については、を参照してください "CSI トポロジを使用します"。

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

ストレージクラスの定義

以下の `StorageClass`定義は、上記のストレージプールを表しています。

フィールドヲシヨウシタテイノレイ `parameter.selector`

を使用する `parameter.selector`と、ボリュームのホストに使用する仮想プールごとにを指定できます `StorageClass`。ボリュームには、選択したプールで定義された要素があります。

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

SMBボリュームの定義例

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

デフォルトネームスペースの基本設定

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

ネームスペースごとに異なるシークレットを使用する

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

ボリュームごとに異なるシークレットを使用する

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

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

バックエンドを作成します

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs
```

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

Google Cloud NetAppボリューム

Google Cloud NetApp Volumeバックエンドの設定

Google Cloud NetApp VolumesをAstra Tridentのバックエンドとして設定できるようになりました。Google Cloud NetApp Volumeバックエンドを使用してNFSボリュームを接続できます。

```
 Google Cloud NetApp Volumes is a tech preview feature in Astra Trident
24.06.
```

Google Cloud NetApp Volumesドライバの詳細

Astra Tridentは、クラスタと通信するためのドライバを提供します `google-cloud-netapp-volumes` 。サポートされているアクセスモードは、*ReadWriteOnce*(RWO)、*ReadOnlyMany*(ROX)、*ReadWriteMany*(RWX)、*ReadWriteOncePod*(RWOP)です。

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

Google Cloud NetApp Volumeバックエンドを設定する準備

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

NFSボリュームノゼンテイジョウケン

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

Google Cloud NetApp Volumeバックエンドを設定する前に、次の条件を満たしていることを確認してください。

- Google Cloud NetApp Volumes Serviceで設定されたGoogle Cloudアカウント。を参照してください "Google Cloud NetAppボリューム"。
- Google Cloudアカウントのプロジェクト番号。を参照してください "プロジェクトの特定"。
- NetApp Volume Admin）ロールが割り当てられたGoogle Cloudサービスアカウント (`netappcloudvolumes.admin`。を参照してください "IDおよびアクセス管理のロールと権限"。
- GCNVアカウントのAPIキーファイル。を参照して "APIキーを使用した認証"
- ストレージプール。を参照してください "ストレージプールの概要"。

Google Cloud NetApp Volumeへのアクセスの設定方法の詳細については、を参照してください "Google Cloud NetApp Volumeへのアクセスをセットアップする"。

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

Google Cloud NetApp VolumeのNFSバックエンド構成オプションについて説明し、構成例を確認します。

バックエンド構成オプション

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

パラメータ	説明	デフォルト
`version`		常に 1
`storageDriverName`	ストレージドライバの名前	の値は `storageDriverName` 「google-cloud-netapp-volumes」と指定する必要があります。
`backendName`	（オプション）ストレージバックエンドのカスタム名	ドライバ名 + "_" + API キーの一部
`storagePools`	ボリューム作成用のストレージプールを指定するオプションのパラメータ。	
`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。	

パラメータ	説明	デフォルト
`location`	Astra TridentがGCNVボリュームを作成するGoogle Cloudの場所。リージョン間Kubernetesクラスタを作成する場合、で作成したボリュームは `location` 、複数のGoogle Cloudリージョンのノードでスケジュールされているワークロードで使用できます。リージョン間トラフィックは追加コストを発生させます。	
`apiKey`	ロールが割り当てられたGoogle Cloudサービスアカウントの`APIキー` `netappcloudvolumes.admin` 。このレポートには、 Google Cloud サービスアカウントの秘密鍵ファイルの JSON 形式のコンテンツが含まれています（バックエンド構成ファイルにそのままコピーされます）。には `apiKey` 、、、、、の各キーのキーと値のペアを含める必要があります。 `type` `project_id` `client_email` `client_id` `auth_uri` `token_uri` `auth_provider_x509_cert_url`、、および `client_x509_cert_url`。	
`nfsMountOptions`	NFS マウントオプションのきめ細かな制御。	"nfsvers=3 "
`limitVolumeSize`	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します。	"" （デフォルトでは適用されません）
`serviceLevel`	ストレージプールとそのボリュームのサービスレベル。値は `flex`、 `standard`、 `premium`、または `extreme`です。	
`network`	GCNVボリュームに使用されるGoogle Cloudネットワーク。	
`debugTraceFlags`	トラブルシューティング時に使用するデバッグフラグ。例： `{"api":false, "method":true}`トラブルシューティングを行って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

`supportedTopologies`	このバックエンドでサポートされているリージョンとゾーンのリストを表します。詳細については、を参照してください "CSI トポロジを使用します"。例：
	`supportedTopologies:`
	`- topology.kubernetes.io/region: europe-west6`
	`topology.kubernetes.io/zone: europe-west6-b`	

ボリュームプロビジョニングオプション

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

パラメータ	説明	デフォルト
`exportRule`	新しいボリュームのエクスポートルール。IPv4アドレスの任意の組み合わせをカンマで区切って指定する必要があります。	"0.0.0.0/0 "
`snapshotDir`	ディレクトリへのアクセス `.snapshot`	いいえ
`snapshotReserve`	Snapshot 用にリザーブされているボリュームの割合	""（デフォルトの0を使用）
`unixPermissions`	新しいボリュームのUNIX権限（8進数の4桁）。	""

構成例

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

最小限の構成

これは、バックエンドの絶対的な最小構成です。この構成では、Astra Tridentが、設定された場所にあるGoogle Cloud NetApp Volumeに委譲されたすべてのストレージプールを検出し、それらのプールの1つに新しいボリュームをランダムに配置します。は省略されているため、 `nasType` `nfs` デフォルトが適用され、バックエンドでNFSボリュームがプロビジョニングされます。

この構成は、Google Cloud NetApp Volumeの使用を開始して試用する場合に最適ですが、実際には、プロビジョニングするボリュームに対して追加の範囲設定が必要になることがよくあります。

```
---

apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: 'f2cb6ed6d7cc10c453f7d3406fc700c5df0ab9ec'
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7OlwWgLwGa==
```

```
    -----END PRIVATE KEY-----

---

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

StoragePoolsフイルタヲシヨウシタセツテイ

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: 'f2cb6ed6d7cc10c453f7d3406fc700c5df0ab9ec'
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7OlwWgLwGa==
    -----END PRIVATE KEY-----

---

apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
```

```
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: '123455380079'
  location: europe-west6
  serviceLevel: premium
  storagePools:
  - premium-pool1-europe-west6
  - premium-pool2-europe-west6
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: '103346282737811234567'
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

仮想プール構成

このバックエンド構成では、1つのファイルに複数の仮想プールが定義されます。仮想プールは、セクションで定義し `storage` ます。さまざまなサービスレベルをサポートする複数のストレージプールがあり、それらを表すストレージクラスをKubernetesで作成する場合に役立ちます。仮想プールラベルは、プールを区別するために使用されます。たとえば、次の例では `performance` 、仮想プールを区別するためにラベルと `serviceLevel` タイプが使用されています。

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

詳細については、を参照してください "仮想プール"。

```
---

apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: 'f2cb6ed6d7cc10c453f7d3406fc700c5df0ab9ec'
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
```

```
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7OlwWgLwGa==
    -----END PRIVATE KEY-----

---

apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: '123455380079'
  location: europe-west6
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: '103346282737811234567'
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
  defaults:
    snapshotReserve: '10'
    exportRule: 10.0.0.0/24
  storage:
    - labels:
        performance: extreme
      serviceLevel: extreme
      defaults:
        snapshotReserve: '5'
        exportRule: 0.0.0.0/0
    - labels:
        performance: premium
      serviceLevel: premium
    - labels:
```

```
        performance: standard
      serviceLevel: standard
```

次の手順

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
kubectl get tridentbackendconfig

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

バックエンドの作成に失敗した場合は、バックエンドの設定に何か問題があります。バックエンドについては、コマンドを使用して説明するか、次のコマンドを実行してログを表示して原因を特定できます `kubectl get tridentbackendconfig <backend-name>` 。

```
tridentctl logs
```

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

その他の例

ストレージクラスの定義の例

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

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

フィールドを使用した定義例 **`parameter.selector`** ：

を使用する `parameter.selector` と、ボリュームのホストに使用される各に対してを指定できます `StorageClass` "仮想プール"。ボリュームには、選択したプールで定義された要素があります。

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

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

PVC定義の例**PVC**テイギノレイ

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

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

```
kubectl get pvc gcnv-nfs-pvc

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

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

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

Google Cloudドライバの詳細

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

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

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

Astra Tridentでは、"サービスタイプ"次の2つのいずれかにCloud Volumes Serviceボリュームを作成できます。

- * CVS - Performance *：デフォルトのAstra Tridentサービスタイプ。パフォーマンスが最適化されたこのサービスタイプは、パフォーマンスを重視する本番環境のワークロードに最適です。CVS -パフォーマンスサービスタイプは、サイズが100GiB以上のボリュームをサポートするハードウェアオプションです。"3

つのサービスレベル"次のいずれかを選択できます。

- `standard`
- `premium`
- `extreme`

- * CVS *：CVSサービスタイプは、中程度のパフォーマンスレベルに制限された高レベルの可用性を提供します。CVSサービスタイプは、ストレージプールを使用して1GiB未満のボリュームをサポートするソフトウェアオプションです。ストレージプールには最大50個のボリュームを含めることができ、すべてのボリュームでプールの容量とパフォーマンスを共有できます。"2つのサービスレベル"次のいずれかを選択できます。
  - `standardsw`
  - `zoneredundantstandardsw`

必要なもの

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

- NetApp Cloud Volumes Service で設定されたGoogle Cloudアカウント
- Google Cloud アカウントのプロジェクト番号
- ロールが割り当てられたGoogle Cloudサービスアカウント `netappcloudvolumes.admin`
- Cloud Volumes Service アカウントのAPIキーファイル

バックエンド構成オプション

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

パラメータ	説明	デフォルト
`version`		常に 1
`storageDriverName`	ストレージドライバの名前	"GCP-cvs"
`backendName`	カスタム名またはストレージバックエンド	ドライバ名 + "_" + API キーの一部
`storageClass`	CVSサービスタイプを指定するためのオプションのパラメータ。CVSサービスタイプを選択するために使用し `software`ます。それ以外の場合、Astra Tridentはサービスタイプがcvs-Performanceとみなされ(`hardware`ます）。	
`storagePools`	CVSサービスタイプのみ。ボリューム作成用のストレージプールを指定するオプションのパラメータ。	
`projectNumber`	Google Cloud アカウントのプロジェクト番号。この値は、Google Cloudポータルのホームページにあります。	

パラメータ	説明	デフォルト
`hostProjectNumber`	共有VPCネットワークを使用する場合は必須です。このシナリオでは、 `projectNumber`はサービスプロジェクト、 `hostProjectNumber`はホストプロジェクトです。	
`apiRegion`	Astra TridentがCloud Volumes Service ボリュームを作成するGoogle Cloudリージョン。リージョン間Kubernetesクラスタを作成する場合、で作成したボリュームは `apiRegion`、複数のGoogle Cloudリージョンのノードでスケジュールされているワークロードで使用できます。リージョン間トラフィックは追加コストを発生させます。	
`apiKey`	ロールが割り当てられたGoogle Cloudサービスアカウントの`APIキー` `netappcloudvolumes.admin` 。このレポートには、 Google Cloud サービスアカウントの秘密鍵ファイルの JSON 形式のコンテンツが含まれています（バックエンド構成ファイルにそのままコピーされます）。	
`proxyURL`	CVSアカウントへの接続にプロキシサーバが必要な場合は、プロキシURLを指定します。プロキシサーバには、 HTTP プロキシまたは HTTPS プロキシを使用できます。HTTPS プロキシの場合、プロキシサーバで自己署名証明書を使用するために証明書の検証はスキップされます。認証が有効になっているプロキシサーバはサポートされていません。	
`nfsMountOptions`	NFS マウントオプションのきめ細かな制御。	"nfsvers=3 "
`limitVolumeSize`	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します。	"" （デフォルトでは適用されません）
`serviceLevel`	新しいボリュームのCVS -パフォーマンスレベルまたはCVSサービスレベル。CVS-Performanceの値は `standard`、、 `premium`または `extreme`です。CVS値は `standardsw`または `zoneredundantstandardsw`です。	CVS -パフォーマンスのデフォルトは「Standard」です。CVSのデフォルトは"standardsw"です。
`network`	Cloud Volumes Service ボリュームに使用するGoogle Cloudネットワーク。	デフォルト
`debugTraceFlags`	トラブルシューティング時に使用するデバッグフラグ。例： `\{"api":false, "method":true}`トラブルシューティングを行って詳細なログダンプが必要な場合を除き、このオプションは使用しないでください。	null

`allowedTopologies`	リージョン間アクセスを有効にするには、のStorageClass定義 `allowedTopologies`にすべてのリージョンが含まれている必要があります。例：
	`- key: topology.kubernetes.io/region`
	`values:`
	`- us-east1`
	`- europe-west1`	

ボリュームプロビジョニングオプション

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

パラメータ	説明	デフォルト
`exportRule`	新しいボリュームのエクスポートルール。CIDR 表記の IPv4 アドレスまたは IPv4 サブネットの任意の組み合わせをカンマで区切って指定する必要があります。	"0.0.0.0/0 "
`snapshotDir`	ディレクトリへのアクセス `.snapshot`	いいえ
`snapshotReserve`	Snapshot 用にリザーブされているボリュームの割合	"" （ CVS のデフォルト値をそのまま使用）
`size`	新しいボリュームのサイズ。CVS -パフォーマンス最小値は100GiBです。CVS最小値は1GiBです。	CVS -パフォーマンスサービスのタイプはデフォルトで「100GiB」です。CVSサービスのタイプではデフォルトが設定されませんが、1GiB以上が必要です。

CVS -パフォーマンスサービスの種類の例

次の例は、CVS -パフォーマンスサービスタイプの設定例を示しています。

例 **1** ：最小限の構成

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

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

例**2**：サービスレベルの設定

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

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

例3：仮想プールの構成

この例では、を使用して、 `storage`仮想プールとを参照するを設定し `StorageClasses`ます。ストレージクラスの定義方法については、を参照して[ストレージクラスの定義]ください。

ここでは、すべての仮想プールに特定のデフォルトが設定されます。これにより、が5%に設定され、が `exportRule`0.0.0.0/0に設定され `snapshotReserve`ます。仮想プールは、セクションで定義し `storage`ます。個 々 の仮想プールはそれぞれ独自に定義され `serviceLevel`、一部のプールはデフォルト値を上書きします。仮想プールラベルを使用して、および `protection`に基づいてプールを区別しました `performance`。

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

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

ストレージクラスの定義

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

ストレージクラスの例

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

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

- 最初のStorageClass(`cvs-extreme-extra-protection`）が最初の仮想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リザーブを提供するストレージプールを呼び出します。Tridentが、どの仮想プールを選択するかを決定し、スナップショット予約の要件が満たされていることを確認します。

CVSサービスタイプの例

次の例は、CVSサービスタイプの設定例を示しています。

例**1**：最小構成

これは、CVSサービスタイプとデフォルトのサービスレベルを指定するために `standardsw`を使用する最小のバックエンド構成 `storageClass`です。

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

例**2**：ストレージプールの構成

このバックエンド構成の例では、を使用して `storagePools`ストレージプールを構成しています。

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

次の手順

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

バックエンドの作成に失敗した場合は、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

Astra Tridentインストール環境で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。

Elementドライバの詳細

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

`solidfire-san`ストレージドライバは、_file_and_block_volumeモードをサポートしています。volumeModeの場合`Filesystem`、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
solidfire-san	iSCSI	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムがありません。raw ブロックデバイスです。
solidfire-san	iSCSI	ファイルシステム	RWO、RWOP	xfs、 ext3、 ext4

開始する前に

Elementバックエンドを作成する前に、次の情報が必要になります。

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

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

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

例**1**：**3**つのボリュームタイプを持つドライバのバックエンド構成 `solidfire-san`

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。次に、ストレージクラスパラメータを使用して各ストレージクラスを使用するように定義します `IOPS`。

```
---
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**：仮想プールを使用するドライバのバックエンドとストレージクラスの構成 `solidfire-san`

この例は、仮想プールとともに、それらを参照するStorageClassesとともに構成されているバックエンド定義ファイルを示しています。

Astra Tridentは、ストレージプール上にあるラベルを、プロビジョニング時にバックエンドストレージLUNにコピーします。ストレージ管理者は、仮想プールごとにラベルを定義したり、ボリュームをラベルでグループ化したりできます。

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

```
---
version: 1
storageDriverName: solidfire-san
Endpoint: https://<user>:<password>@<mvip>/json-rpc/8.0
SVIP: "<svip>:3260"
```

```
TenantName: "<tenant>"
UseCHAP: true
Types:
- Type: Bronze
  Qos:
    minIOPS: 1000
    maxIOPS: 2000
    burstIOPS: 4000
- Type: Silver
  Qos:
    minIOPS: 4000
    maxIOPS: 6000
    burstIOPS: 8000
- Type: Gold
  Qos:
    minIOPS: 6000
    maxIOPS: 8000
    burstIOPS: 10000
type: Silver
labels:
  store: solidfire
  k8scluster: dev-1-cluster
region: us-east-1
storage:
- labels:
    performance: gold
    cost: '4'
  zone: us-east-1a
  type: Gold
- labels:
    performance: silver
    cost: '3'
  zone: us-east-1b
  type: Silver
- labels:
    performance: bronze
    cost: '2'
  zone: us-east-1c
  type: Bronze
- labels:
    performance: silver
    cost: '1'
  zone: us-east-1d
```

次のStorageClass定義は、上記の仮想プールを参照しています。フィールドを使用して `parameters.selector`、各StorageClassはボリュームのホストに使用できる仮想プールを呼び出します。

ボリュームには、選択した仮想プール内で定義された要素があります。

最初のStorageClass(`solidfire-gold-four`）が最初の仮想プールにマッピングされます。これは、ゴールドのパフォーマンスとゴールドのパフォーマンスを提供する唯一のプールです `Volume Type QoS`。最後のStorageClass(`solidfire-silver`）は、Silverパフォーマンスを提供するストレージプールを呼び出します。Tridentが、どの仮想プールを選択するかを判断し、ストレージ要件を確実に満たすようにします。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-gold-four
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=gold; cost=4"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver-three
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=silver; cost=3"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-bronze-two
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=bronze; cost=2"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver-one
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=silver; cost=1"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=silver"
  fsType: "ext4"
```

詳細情報

- "ボリュームアクセスグループ"

ONTAP SANドライバ

ONTAP SANドライバの概要

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

ONTAP SANドライバの詳細

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

保護、リカバリ、モビリティにAstra Controlを使用している場合は、を参照してください。Astra Controlドライバの互換性

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされるファイルシステム
`ontap-san`	iSCSI	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし。rawブロックデバイスです
`ontap-san`	iSCSI	ファイルシステム	RWO、RWOP ROXおよびRWXは、ファイルシステムボリュームモードでは使用できません。	`xfs`、 `ext3`、 `ext4`
`ontap-san`	NVMe / TCP を参照してください NVMe/TCPに関するその他の考慮事項。	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし。rawブロックデバイスです

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされるファイルシステム
ontap-san	NVMe / TCP を参照してください NVMe/TCPに関するその他の考慮事項。	ファイルシステム	RWO、RWOP ROXおよびRWXは、ファイルシステムボリュームモードでは使用できません。	xfs、 ext3、 ext4
ontap-san-economy	iSCSI	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし。rawブロックデバイスです
ontap-san-economy	iSCSI	ファイルシステム	RWO、RWOP ROXおよびRWXは、ファイルシステムボリュームモードでは使用できません。	xfs、 ext3、 ext4

Astra Controlドライバの互換性

Astra Controlは、 ontap-nas-flexgroup、、および ontap-san`ドライバで作成されたボリュームに対して、シームレスな保護、ディザスタリカバリ、モビリティ（Kubernetesクラスタ間でのボリューム移動）を提供します `ontap-nas。詳細については、を参照してください "Astra Controlレプリケーションの前提条件" 。

- 永続的ボリュームの使用数がよりも多くなると予想される場合にのみ使用します `ontap-san-economy`"サポートされるONTAPの制限"。
- 永続的ボリュームの使用数がよりも多いと予想され、 `ontap-san-economy`ドライバを使用できない場合にのみ"サポートされるONTAPの制限"使用して `ontap-nas-economy`ください。
- データ保護、ディザスタリカバリ、モビリティの必要性が予想される場合は使用しない `ontap-nas-economy`でください。

ユーザ権限

Astra Tridentは、ONTAP管理者またはSVM管理者（通常はクラスタユーザまたは vsadmin`SVMユーザ、または同じロールの別の名前のユーザを使用）として実行することを想定しています `admin。Amazon FSx for NetApp ONTAP環境の場合、Astra Tridentは、クラスタユーザまたは `vsadmin`SVMユーザ、または同じロールの別の名前のユーザを使用して、ONTAP管理者またはSVM管理者として実行されることが想定され `fsxadmin`ます。この `fsxadmin`ユーザは、クラスタ管理者ユーザに代わる限定的なユーザです。

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

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

NVMe/TCPに関するその他の考慮事項

Astra Tridentでは、次のドライバを使用してNon-Volatile Memory Express（NVMe）プロトコルがサポートされ `ontap-san`ます。

- IPv6
- NVMeボリュームのSnapshotとクローン
- NVMeボリュームのサイズ変更
- Astra Tridentでライフサイクルを管理できるように、Astra Tridentの外部で作成されたNVMeボリュームをインポートする
- NVMeネイティブマルチパス
- Kubernetesノードのグレースフルシャットダウンまたはグレースフルシャットダウン（24.06）

Astra Tridentでは次の機能がサポートされません。

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

ONTAP SANドライバを使用してバックエンドを設定する準備をします

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

要件

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、ドライバを使用するクラス `ontap-san`と、ドライバ `san-default`を使用するクラスを `ontap-san-economy`設定できます `san-dev`。

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。詳細については、を参照してください "ワーカーノードを準備します" 。

ONTAPバックエンドの認証

Astra Trident には、 ONTAP バックエンドを認証する 2 つのモードがあります。

- credential based ：必要な権限を持つ ONTAP ユーザのユーザ名とパスワード。ONTAPのバージョンと最大限の互換性を確保するために、や `vsadmin`などの事前定義されたセキュリティログインロールを使用することを推奨し `admin`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。

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

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

クレデンシャルベースの認証を有効にします

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

バックエンド定義の例は次のようになります。

YAML

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

JSON

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

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

証明書ベースの認証を有効にする

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のセキュリティログインロールが認証方式をサポートしていることを確認します `cert`。

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

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

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

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

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

認証方法を更新するか、クレデンシャルをローテーションして

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

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

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

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

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

双方向 **CHAP** を使用して接続を認証します

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

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

`useCHAP`パラメータはブール値のオプションで、一度だけ設定できます。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。

さらに `useCHAP=true`、 `chapInitiatorSecret`、 `chapTargetInitiatorSecret`、 `chapTargetUsername`、および `chapUsername`フィールドを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、を実行してバックエンドを作成したあとに変更できます `tridentctl update`。

仕組み

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

- SVM で CHAP をセットアップします。
  - SVMのデフォルトのイニシエータセキュリティタイプがnone（デフォルトで設定）*で、*ボリュームに既存のLUNがない場合、Astra Tridentはデフォルトのセキュリティタイプをに設定し `CHAP`、CHAP イニシエータとターゲットのユーザ名とシークレットの設定に進みます。
  - SVM に LUN が含まれている場合、 Trident は SVM で CHAP を有効にしません。これにより、SVM にすでに存在するLUNへのアクセスが制限されなくなります。
- CHAP イニシエータとターゲットのユーザ名とシークレットを設定します。これらのオプションは、バックエンド構成で指定する必要があります（上記を参照）。

バックエンドが作成されると、Astra Tridentは対応するCRDを作成し `tridentbackend`、CHAPシークレットとユーザ名をKubernetesシークレットとして格納します。このバックエンドの Astra Trident によって作成されたすべての PVS がマウントされ、 CHAP 経由で接続されます。

クレデンシャルをローテーションし、バックエンドを更新

CHAPクレデンシャルを更新するには、ファイルのCHAPパラメータを更新し `backend.json`ます。そのためには、CHAPシークレットを更新し、コマンドを使用して変更を反映する必要があり `tridentctl update`ます。

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

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

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

既存の接続は影響を受けません。 SVM の Astra Trident でクレデンシャルが更新されても、引き続きアクティブです。新しい接続では更新されたクレデンシャルが使用され、既存の接続は引き続きアクティブです。古い PVS を切断して再接続すると、更新されたクレデンシャルが使用されます。

ONTAP SANの設定オプションと例

Astra Tridentのインストール環境でONTAP SANドライバを作成して使用する方法をご紹介します。このセクションでは、バックエンドの構成例と、バックエンドをStorageClassesにマッピングするための詳細を示します。

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
`version`		常に 1
`storageDriverName`	ストレージドライバの名前	`ontap-nas` `ontap-nas-economy`、、 `ontap-nas-flexgroup`、、 `ontap-san` `ontap-san-economy`
`backendName`	カスタム名またはストレージバックエンド	ドライバ名+"_"+ dataLIF
`managementLIF`	クラスタ管理LIFまたはSVM管理LIFのIPアドレス。Fully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定できます。IPv6フラグを使用してAstra Tridentをインストールした場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、のように角かっこで定義する必要があります `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。シームレスなMetroClusterスイッチオーバーについては、を参照して[mcc-best]ください。	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」
`dataLIF`	プロトコル LIF の IP アドレス。* iSCSIの場合は指定しないでください。**Astra Trident**は、を使用して**"ONTAP の選択的LUNマップ"**、マルチパスセッションの確立に必要な**iSCI LIF**を検出します。が明示的に定義されている場合は、警告が生成され `**dataLIF**` ます。 MetroClusterの場合は省略してください。*を参照してください[mcc-best]。	SVMの派生物です
`svm`	使用するStorage Virtual Machine * MetroClusterでは省略*を参照してください[mcc-best]。	SVMが指定されている場合に派生 `managementLIF`
`useCHAP`	CHAPを使用してONTAP SANドライバのiSCSIを認証します（ブーリアン）。Astra Tridentで、バックエンドで指定されたSVMのデフォルト認証として双方向CHAPを設定して使用する場合は、をに設定 `true` します。詳細については、を参照してください "ONTAP SANドライバを使用してバックエンドを設定する準備をします" 。	`false`
`chapInitiatorSecret`	CHAP イニシエータシークレット。必要な場合 `useCHAP=true`	""
`labels`	ボリュームに適用する任意の JSON 形式のラベルのセット	""
`chapTargetInitiatorSecret`	CHAP ターゲットイニシエータシークレット。必要な場合 `useCHAP=true`	""
`chapUsername`	インバウンドユーザ名。必要な場合 `useCHAP=true`	""
`chapTargetUsername`	ターゲットユーザ名。必要な場合 `useCHAP=true`	""
`clientCertificate`	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""

パラメータ	説明	デフォルト
clientPrivateKey	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""
trustedCACertificate	信頼された CA 証明書の Base64 エンコード値。オプション。証明書ベースの認証に使用されます。	""
username	ONTAP クラスタとの通信に必要なユーザ名。クレデンシャルベースの認証に使用されます。	""
password	ONTAP クラスタとの通信にパスワードが必要です。クレデンシャルベースの認証に使用されます。	""
svm	使用する Storage Virtual Machine	SVMが指定されている場合に派生 managementLIF
storagePrefix	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。あとから変更することはできません。このパラメータを更新するには、新しいバックエンドを作成する必要があります。	trident
limitAggregateUsage	使用率がこの割合を超えている場合は、プロビジョニングが失敗します。Amazon FSx for NetApp ONTAP バックエンドを使用している場合は、を指定しないで limitAggregateUsage`ください。指定されたと `vsadmin`には `fsxadmin、アグリゲートの使用量を取得し、Astra Tridentを使用してアグリゲートを制限するために必要な権限が含まれていません。	"" （デフォルトでは適用されません）
limitVolumeSize	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。また、qtreeおよびLUNに対して管理するボリュームの最大サイズを制限します。	"" （デフォルトでは適用されません）
lunsPerFlexvol	FlexVol あたりの最大 LUN 数。有効な範囲は 50 、 200 です	100
debugTraceFlags	トラブルシューティング時に使用するデバッグフラグ。例： {"api"：false、"method"：true} トラブルシューティングを行って詳細なログダンプが必要な場合を除き、は使用しないでください。	null

パラメータ	説明	デフォルト
useREST	ONTAP REST API を使用するためのブーリアンパラメータ。 useREST に設定する true`と、Astra Trident はONTAP REST APIを使用してバックエンドと通信します。に設定する `false`と、Astra Trident はONTAP ZAPI呼び出しを使用してバックエンドと通信します。この機能にはONTAP 9.11.1以降が必要です。また、使用するONTAPログインロールには、アプリケーションへのアクセス権が必要です `ontap 。これは、事前に定義された役割と役割によって実現され vsadmin cluster-admin ます。Astra Trident 24.06リリースおよびONTAP 9.15.1以降では、が userREST デフォルトでに設定されてい true ます。ONTAP ZAPI呼び出しを使用するようににに変更してください。 useREST false useREST はNVMe/TCPに完全修飾されています。	true ONTAP 9.15.1以降の場合は、それ以外の場合は false。
sanType	iSCSIまたは `nvme`NVMe/TCPの選択に使用し `iscsi` ます。	`iscsi`空白の場合

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

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

パラメータ	説明	デフォルト
spaceAllocation	space-allocation for LUN のコマンドを指定します	"正しい"
spaceReserve	スペースリザベーションモード：「none」（シン）または「volume」（シック）	"なし"
snapshotPolicy	使用する Snapshot ポリシー	"なし"
qosPolicy	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します。Trident が Astra で QoS ポリシーグループを使用するには、 ONTAP 9.8 以降が必要です。非共有のQoSポリシーグループを使用して、各コンスティチュエントに個別にポリシーグループを適用することを推奨します。共有 QoS ポリシーグループにより、すべてのワークロードの合計スループットに対して上限が適用されます。	""
adaptiveQosPolicy	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	""

パラメータ	説明	デフォルト
`snapshotReserve`	Snapshot 用にリザーブされているボリュームの割合	が「none」の場合は「0」`snapshotPolicy`、それ以外の場合は「」
`splitOnClone`	作成時にクローンを親からスプリットします	いいえ
`encryption`	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです。 `false`このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームがに有効になります。詳細については、を参照してください"Astra TridentとNVEおよびNAEの相互運用性"。	いいえ
`luksEncryption`	LUKS暗号化を有効にします。を参照してください"Linux Unified Key Setup（LUKS；統合キーセットアップ）を使用"。LUKS暗号化はNVMe/TCPではサポートされません。	""
`securityStyle`	新しいボリュームのセキュリティ形式	`unix`
`tieringPolicy`	「none」を使用する階層化ポリシー	ONTAP 9.5より前のSVM-DR設定の場合は「snapshot-only」
`nameTemplate`	カスタムボリューム名を作成するためのテンプレート。	""
`limitVolumePoolSize`	ONTAP SANエコノミーバックエンドでLUNを使用する場合の、要求可能な最大FlexVolサイズ。	""（デフォルトでは適用されません）

ボリュームプロビジョニングの例

デフォルトが定義されている例を次に示します。

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

ドライバを使用して作成されたすべてのボリュームについて、 `ontap-san`Astra TridentはLUNメタデータに対応するために10%の容量をFlexVolに追加します。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。

定義されたバックエンドについては `snapshotReserve`、Astra Tridentで次のようにボリュームのサイズが計算されます。

```
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。= 5%、PVC要求= 5GiBの場合、 `snapshotReserve`ボリュームの合計サイズは5.79GiB、使用可能なサイズは5.5GiBです。 `volume show`次の例のような結果が表示されます。

```
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。

Amazon FSx on NetApp ONTAPとAstra Tridentを使用している場合は、IPアドレスではなく、LIFのDNS名を指定することを推奨します。

ONTAP SANの例

これはドライバを使用した基本的な設定です `ontap-san`。

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

ONTAP SANの経済性の例

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

1. 例

スイッチオーバー後およびスイッチバック中にバックエンド定義を手動で更新する必要がないようにバックエンドを設定できます"SVMレプリケーションとリカバリ"。

スイッチオーバーとスイッチバックをシームレスに実行するには、を使用してSVMを指定し `managementLIF`、パラメータと `svm`パラメータを省略します `dataLIF`。例：

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

証明書ベースの認証の例

この基本的な設定例では `clientCertificate` `clientPrivateKey`、および `trustedCACertificate`（信頼されたCAを使用している場合はオプション）が入力され `backend.json`、それぞれクライアント証明書、秘密鍵、および信頼されたCA証明書のbase64でエンコードされた値が使用されます。

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

双方向CHAPの例

これらの例では、がに設定され `true`たバックエンドが作成され `useCHAP`ます。

ONTAP SAN CHAPの例

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

ONTAP SANエコノミーCHAPの例

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

NVMe/TCPの例

ONTAPバックエンドでNVMeを使用するSVMを設定しておく必要があります。これはNVMe/TCPの基本的なバックエンド構成です。

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

nameTemplateを使用したバックエンド構成の例

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

仮想プールを使用するバックエンドの例

これらのサンプルバックエンド定義ファイルでは、none、 `spaceAllocation`false、false `encryption`など、すべてのストレージプールに特定のデフォルトが設定されています `spaceReserve`。仮想プールは、ストレージセクションで定義します。

Astra Tridentでは、[Comments]フィールドにプロビジョニングラベルが設定されます。FlexVol にコメントが設定されます。Astra Tridentは、プロビジョニング時に仮想プール上にあるすべてのラベルをストレージボリュームにコピーします。ストレージ管理者は、仮想プールごとにラベルを定義したり、ボリュームをラベルでグループ化したりできます。

これらの例では、一部のストレージプールで独自の、、 `spaceAllocation`および `encryption`の値が設定され `spaceReserve`、一部のプールでデフォルト値が上書きされます。

ONTAP SANの例

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

ONTAP SANの経済性の例

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

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

NVMe/TCPの例

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

バックエンドを **StorageClasses** にマッピングします

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

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

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

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

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

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

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

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

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

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

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

- `my-test-app-sc`StorageClassは、を使用してドライバの `sanType: nvme`仮想プールに `ontap-san`マッピングされます `testAPP。これは唯一のプール`testApp`です。

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

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

ONTAP NASドライバ

ONTAP NASドライバの概要

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

ONTAP NASドライバの詳細

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

保護、リカバリ、モビリティにAstra Controlを使用している場合は、を参照してください。Astra Controlドライバの互換性

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

Astra Controlドライバの互換性

Astra Controlは、 `ontap-nas-flexgroup`、、および `ontap-san`ドライバで作成されたボリュームに対して、シームレスな保護、ディザスタリカバリ、モビリティ（`Kubernetes`クラスタ間でのボリューム移動）を提供します `ontap-nas`。詳細については、を参照してください "Astra Controlレプリケーションの前提条件" 。

- 永続的ボリュームの使用数がよりも多くなると予想される場合にのみ使用します `ontap-san-economy`"サポートされるONTAPの制限"。
- 永続的ボリュームの使用数がよりも多いと予想され、 `ontap-san-economy`ドライバを使用できない場合にのみ"サポートされるONTAPの制限"使用して `ontap-nas-economy`ください。
- データ保護、ディザスタリカバリ、モビリティの必要性が予想される場合は使用しない `ontap-nas-economy`でください。

ユーザ権限

Astra Tridentは、ONTAP管理者またはSVM管理者（通常はクラスタユーザまたは `vsadmin` `SVM`ユーザ、または同じロールの別の名前のユーザを使用）として実行することを想定しています `admin`。

Amazon FSx for NetApp ONTAP環境の場合、Astra Tridentは、クラスタユーザまたは `vsadmin`SVMユーザ、または同じロールの別の名前のユーザを使用して、ONTAP管理者またはSVM管理者として実行されることが想定され `fsxadmin`ます。この `fsxadmin`ユーザは、クラスタ管理者ユーザに代わる限定的なユーザです。

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

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

ONTAP NASドライバを使用してバックエンドを設定する準備をします

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

要件

- ONTAP バックエンドすべてに対して、 Astra Trident が SVM に少なくとも 1 つのアグリゲートを割り当てておく必要があります。
- 複数のドライバを実行し、どちらか一方を参照するストレージクラスを作成できます。たとえば、ドライバを使用するGoldクラスと、ドライバを使用するBronzeクラスを `ontap-nas-economy`設定できます `ontap-nas`。
- 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。"ここをクリック"詳細については、を参照してください。
- Astra Tridentは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみをサポート詳細については、を参照してください SMBボリュームをプロビジョニングする準備をします 。

ONTAPバックエンドの認証

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：このモードでは、ONTAPバックエンドに十分な権限が必要です。ONTAPのバージョンと最大限の互換性を確保するために、や `vsadmin`などの事前定義されたセキュリティログインロールに関連付けられたアカウントを使用することを推奨します `admin`。
- Certificate-based：Astra TridentがONTAPクラスタと通信するためには、バックエンドに証明書がインストールされている必要があります。この場合、バックエンド定義には、 Base64 でエンコードされたクライアント証明書、キー、および信頼された CA 証明書（推奨）が含まれている必要があります。

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

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

クレデンシャルベースの認証を有効にします

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

バックエンド定義の例は次のようになります。

YAML

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

JSON

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

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

証明書ベースの認証を有効にします

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

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

一般的なワークフローは次の手順で構成されます。

手順

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

Name （ CN ；共通名）を設定します。

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

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

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

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

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

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

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

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

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

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

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

認証方法を更新するか、クレデンシャルをローテーションして

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

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

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

パスワード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

Astra Tridentでは、ONTAPバックエンドのエクスポートポリシーを動的に管理できます。これにより、ストレージ管理者は、明示的なルールを手動で定義するのではなく、ワーカーノードの IP で許容されるアドレススペースを指定できます。エクスポートポリシーの管理が大幅に簡易化され、エクスポートポリシーを変更しても、ストレージクラスタに対する手動の操作は不要になります。さらに、この方法を使用すると、ストレージクラスタへのアクセスを指定した範囲内のIPを持つワーカーノードだけに制限できるため、きめ細かい管理が可能になります。

ダイナミックエクスポートポリシーを使用する場合は、Network Address Translation（NAT;ネットワークアドレス変換）を使用しないでください。NATを使用すると、ストレージコントローラは実際のIPホストアドレスではなくフロントエンドのNATアドレスを認識するため、エクスポートルールに一致しない場合はアクセスが拒否されます。

例

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

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

この機能を使用する場合は、SVMのルートジャンクションに、ノードのCIDRブロックを許可するエクスポートルール（デフォルトのエクスポートポリシーなど）を含む事前に作成したエクスポートポリシーがあることを確認する必要があります。NetAppが推奨するベストプラクティスに従って、1つのSVMをAstra Trident専用にする。

ここでは、上記の例を使用してこの機能がどのように動作するかについて説明します。

- `autoExportPolicy`がに設定されてい `true`ます。これは、Astra TridentがSVM用のエクスポートポリシーを作成し、アドレスブロックを使用してルールの追加と削除を処理する `autoExportCIDRs`ことを示します `svm1`。たとえば、UUIDが403b5326-8482-40dB-96d0-d83fb3f4daecのバックエンドをに設定する `true`と `autoExportPolicy`、という名前のエクスポートポリシーがSVMに作成されます `trident-403b5326-8482-40db-96d0-d83fb3f4daec`。

- `autoExportCIDRs`アドレスブロックのリストが含まれます。このフィールドは省略可能で、デフォルト値は ["0.0.0.0/0" 、 "::/0" です。定義されていない場合は、 Astra Trident が、ワーカーノードで検出されたすべてのグローバルにスコープ指定されたユニキャストアドレスを追加します。

この例では `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、実行先のノードを登録すると、ノードのIPアドレスを取得してに表示されるアドレスブロックと照合し`autoExportCIDRs`ます。IPをフィルタリングしたあと、Astra Tridentは、検出したクライアントIP用のエクスポートポリシールールを作成します。このルールには、特定したノードごとに1つのルールが含まれています。

バックエンドを作成した後で、バックエンドのおよびを `autoExportCIDRs`更新できます `autoExportPolicy`。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無効にして、手動で作成したエクスポートポリシーにフォールバックすることもできます `autoExportPolicy`。この場合、バックエンド設定でパラメータを設定する必要があります `exportPolicy`。

Astra Tridentでバックエンドが作成または更新されたら、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 `tridentctl`ます。

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

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

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。

既存のバックエンドがある場合は、を使用してバックエンドを更新する `tridentctl update backend`と、Astra Tridentがエクスポートポリシーを自動的に管理するようになります。これにより、バックエンドのUUIDに基づいてという名前の新しいエクスポートポリシーが作成され、バックエンドにあるボリュームは再マウント時に新しく作成されたエクスポートポリシーを使用します。

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

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

SMBボリュームをプロビジョニングする準備をします

少し準備をするだけで、ドライバを使用してSMBボリュームをプロビジョニングできます `ontap-nas`。

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

開始する前に

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

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

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

- Windowsサービスとして設定されたCSIプロキシ。を設定するには `csi-proxy`、Windowsで実行されているKubernetesノードについて、またはを"GitHub: Windows向けCSIプロキシ"参照してください"GitHub: CSIプロキシ"。

手順

1. オンプレミスのONTAPの場合は、必要に応じてSMB共有を作成するか、Astra TridentでSMB共有を作成できます。

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

   SMB管理共有は、共有フォルダスナップインを使用するか、ONTAP CLIを使用して作成できます"Microsoft管理コンソール"。ONTAP CLIを使用してSMB共有を作成するには、次の手順を実行します

   a. 必要に応じて、共有のディレクトリパス構造を作成します。

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

b. 指定したSVMに関連付けられているSMB共有を作成します。

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

c. 共有が作成されたことを確認します。

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

詳細については、を参照して"SMB共有を作成する"ください。

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

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

ONTAP NASの設定オプションと例

Astra Tridentのインストール環境でONTAP NASドライバを作成して使用する方法について説明します。このセクションでは、バックエンドの構成例と、バックエンドをStorageClassesにマッピングするための詳細を示します。

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
version		常に 1
storageDriverName	ストレージドライバの名前	「ontap-nas」、「ontap-nas-economy」、「ontap-nas-flexgroup」、「ontap-san」、「ontap-san-economy」
backendName	カスタム名またはストレージバックエンド	ドライバ名+"_"+ dataLIF
managementLIF	クラスタまたはSVM管理LIFのIPアドレス完全修飾ドメイン名（FQDN）を指定できます。IPv6フラグを使用してAstra Tridentをインストールした場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、のように角かっこで定義する必要があります `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。シームレスなMetroClusterスイッチオーバーについては、を参照して[mcc-best]ください。	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」
dataLIF	プロトコル LIF の IP アドレス。指定することをお勧めします `dataLIF`。指定しない場合は、Astra TridentがSVMからデータLIFを取得します。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。初期設定後に変更できます。を参照してください 。IPv6フラグを使用してAstra Tridentをインストールした場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、のように角かっこで定義する必要があります `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。* MetroClusterの場合は省略してください。*を参照してください[mcc-best]。	指定されたアドレス、または指定されていない場合はSVMから取得されるアドレス（非推奨）
svm	使用するStorage Virtual Machine * MetroClusterでは省略*を参照してください[mcc-best]。	SVMが指定されている場合に派生 `managementLIF`
autoExportPolicy	エクスポートポリシーの自動作成と更新を有効にします[ブーリアン]。オプションと `autoExportCIDRs`オプションを使用する `autoExportPolicy`と、Astra Tridentでエクスポートポリシーを自動的に管理できます。	正しくない
autoExportCIDRs	が有効な場合にKubernetesのノードIPをフィルタリングするCIDRのリスト `autoExportPolicy`。オプションと `autoExportCIDRs`オプションを使用する `autoExportPolicy`と、Astra Tridentでエクスポートポリシーを自動的に管理できます。	["0.0.0.0/0","::/0"]'
labels	ボリュームに適用する任意の JSON 形式のラベルのセット	""
clientCertificate	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""
clientPrivateKey	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""

パラメータ	説明	デフォルト
`trustedCACertificate`	信頼された CA 証明書の Base64 エンコード値。オプション。証明書ベースの認証に使用されます	""
`username`	クラスタ / SVM に接続するためのユーザ名。クレデンシャルベースの認証に使用されます	
`password`	クラスタ / SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます	
`storagePrefix`	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。設定後に更新することはできません	"トライデント"
`limitAggregateUsage`	使用率がこの割合を超えている場合は、プロビジョニングが失敗します。* Amazon FSX for ONTAP * には適用されません	"" （デフォルトでは適用されません）
`limitVolumeSize`	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。また、qtreeおよびLUNに対して管理するボリュームの最大サイズを制限し、オプションを使用すると、`qtreesPerFlexvol`FlexVolあたりのqtreeの最大数をカスタマイズできます。	"" （デフォルトでは適用されません）
`lunsPerFlexvol`	FlexVol あたりの最大 LUN 数。有効な範囲は 50 、 200 です	"100"
`debugTraceFlags`	トラブルシューティング時に使用するデバッグフラグ。例： {"api"：false、"method"：true} トラブルシューティングを行って詳細なログダンプが必要な場合を除き、は使用しない `debugTraceFlags`でください。	null
`nasType`	NFSボリュームまたはSMBボリュームの作成を設定オプションは `nfs`、 `smb`またはnullです。nullに設定すると、デフォルトでNFSボリュームが使用されます。	`nfs`
`nfsMountOptions`	NFSマウントオプションをカンマで区切ったリスト。Kubernetes永続ボリュームのマウントオプションは通常はストレージクラスで指定されますが、ストレージクラスでマウントオプションが指定されていない場合、Astra Tridentはストレージバックエンドの構成ファイルで指定されているマウントオプションを使用します。ストレージクラスや構成ファイルにマウントオプションが指定されていない場合、Astra Tridentは関連付けられた永続的ボリュームにマウントオプションを設定しません。	""
`qtreesPerFlexvol`	FlexVol あたりの最大 qtree 数。有効な範囲は [50 、 300] です。	"200"

パラメータ	説明	デフォルト
smbShare	Microsoft管理コンソールまたはONTAP CLIを使用して作成されたSMB共有の名前、Astra TridentでSMB共有を作成できる名前、ボリュームへの共有アクセスを禁止する場合はパラメータを空白のままにすることができます。オンプレミスのONTAPでは、このパラメータはオプションです。このパラメータはAmazon FSx for ONTAPバックエンドで必須であり、空にすることはできません。	smb-share
useREST	ONTAP REST API を使用するためのブーリアンパラメータ。 useREST に設定する true`と、Astra TridentはONTAP REST APIを使用してバックエンドと通信します。に設定する `false`と、Astra TridentはONTAP ZAPI呼び出しを使用してバックエンドと通信します。この機能にはONTAP 9.11.1以降が必要です。また、使用するONTAPログインロールには、アプリケーションへのアクセス権が必要です `ontap 。これは、事前に定義された役割と役割によって実現され vsadmin cluster-admin ます。Astra Trident 24.06リリースおよびONTAP 9.15.1以降では、が userREST デフォルトでに設定されてい true ます。ONTAP ZAPI呼び出しを使用するようににに変更してください。 useREST false	true ONTAP 9.15.1以降の場合は、それ以外の場合は false。
limitVolumePoolSize	ONTAP NASエコノミーバックエンドでqtreeを使用する場合の、要求可能なFlexVolの最大サイズ。	""（デフォルトでは適用されません）

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

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

パラメータ	説明	デフォルト
spaceAllocation	space-allocation for LUN のコマンドを指定します	"正しい"
spaceReserve	スペースリザベーションモード：「none」（シン）または「volume」（シック）	"なし"
snapshotPolicy	使用する Snapshot ポリシー	"なし"
qosPolicy	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	""
adaptiveQosPolicy	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します。経済性に影響する ONTAP - NAS ではサポートされません。	""

パラメータ	説明	デフォルト
`snapshotReserve`	Snapshot 用にリザーブされているボリュームの割合	が「none」の場合は「0」`snapshotPolicy`、それ以外の場合は「」
`splitOnClone`	作成時にクローンを親からスプリットします	いいえ
`encryption`	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです。 `false`このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームがに有効になります。詳細については、を参照してください"Astra TridentとNVEおよびNAEの相互運用性"。	いいえ
`tieringPolicy`	「none」を使用する階層化ポリシー	ONTAP 9.5より前のSVM-DR設定の場合は「snapshot-only」
`unixPermissions`	新しいボリュームのモード	NFSボリュームの場合は「777」、SMBボリュームの場合は空（該当なし）
`snapshotDir`	ディレクトリへのアクセスを管理します。`.snapshot`	いいえ
`exportPolicy`	使用するエクスポートポリシー	デフォルト
`securityStyle`	新しいボリュームのセキュリティ形式。NFSのサポート `mixed`と `unix`セキュリティ形式SMBのサポート `mixed`と `ntfs`セキュリティ形式。	NFSのデフォルトはです `unix`。SMBのデフォルトはです `ntfs`。
`nameTemplate`	カスタムボリューム名を作成するためのテンプレート。	""

Trident が Astra で QoS ポリシーグループを使用するには、 ONTAP 9.8 以降が必要です。共有されない QoS ポリシーグループを使用して、各コンスティチュエントに個別にポリシーグループを適用することを推奨します。共有 QoS ポリシーグループにより、すべてのワークロードの合計スループットに対して上限が適用されます。

ボリュームプロビジョニングの例

デフォルトが定義されている例を次に示します。

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

と `ontap-nas-flexgroups`については、 `ontap-nas`Trident新しい計算式を使用して、FlexVol がsnapshotReserveの割合とPVCで正しくサイジングされるようになりました。ユーザが PVC を要求すると、 Astra Trident は、新しい計算を使用して、より多くのスペースを持つ元の FlexVol を作成します。この計算により、ユーザは要求された PVC 内の書き込み可能なスペースを受信し、要求されたスペースよりも少ないスペースを確保できます。v21.07 より前のバージョンでは、ユーザが PVC を要求すると（ 5GiB など）、 snapshotReserve が 50% に設定されている場合、書き込み可能なスペースは 2.5GiB のみになります。これは、ユーザが要求したのはボリューム全体であり、その割合であるため `snapshotReserve`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、Astra Tridentではボリューム全体に対する割合が定義され `snapshotReserve`ます。これはには適用されませ `ontap-nas-economy`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`次の例のような結果が表示されます。

以前のインストールからの既存のバックエンドは、 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、以前のと2GiBのPVCで `snapshotReserve=50` は、1GiBの書き込み可能なスペースを提供するボリュームが作成されました。たとえば、ボリュームのサイズを 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の経済性の例

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

ONTAP NAS FlexGroupの例

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

MetroClusterの例

スイッチオーバー後およびスイッチバック中にバックエンド定義を手動で更新する必要がないようにバックエンドを設定できます"SVMレプリケーションとリカバリ"。

スイッチオーバーとスイッチバックをシームレスに実行するには、を使用してSVMを指定し `managementLIF`、パラメータと `svm`パラメータを省略します `dataLIF`。例：

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

SMBボリュームの例

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

証明書ベースの認証の例

これは最小限のバックエンド構成の例です。 `clientCertificate`、 `clientPrivateKey`、および `trustedCACertificate`（信頼されたCAを使用している場合はオプション）に値が入力され `backend.json`、それぞれクライアント証明書、秘密鍵、および信頼されたCA証明書のBase64でエンコードされた値が使用されます。

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

自動エクスポートポリシーの例

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

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

IPv6アドレスの例

次に、IPv6アドレスの使用例を示し `managementLIF`ます。

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

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

`smbShare`SMBボリュームを使用するFSx for ONTAPでは、パラメータは必須です。

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

nameTemplateを使用したバックエンド構成の例

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

仮想プールを使用するバックエンドの例

以下に示すサンプルのバックエンド定義ファイルでは、すべてのストレージプールに特定のデフォルトが設定されています（at none、at `spaceAllocation` false、at false `encryption`など） `spaceReserve`。仮想プールは、ストレージセクションで定義します。

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

これらの例では、一部のストレージプールで独自の、、 `spaceAllocation`および `encryption`の値が設定され `spaceReserve`、一部のプールでデフォルト値が上書きされます。

ONTAP NASの例

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

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

ONTAP NAS FlexGroupの例

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

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

ONTAP NASの経済性の例

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

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

バックエンドを **StorageClasses** にマッピングします

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

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

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

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

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

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

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

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

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

- creditpoints-5k`StorageClassは、バックエンドの3番目の仮想プールとバックエンドの2番目の仮想プール `ontap-nas-economy`にマッピングされます `ontap-nas。これらは、5000クレジットポイントを持つ唯一のプールオファリングです。

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

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

初期設定後に更新 dataLIF

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

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

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

NetApp ONTAP 対応の Amazon FSX

Amazon FSX for NetApp ONTAP で Astra Trident を使用

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

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

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。

要件

"Astra Trident の要件"FSx for ONTAPとAstra Tridentを統合するには、さらに次のものが必要です。

- 既存のAmazon EKSクラスタまたはがインストールされた自己管理型Kubernetesクラスタ `kubectl`。
- クラスタのワーカーノードから到達可能な既存のAmazon FSx for NetApp ONTAPファイルシステムおよびStorage Virtual Machine（SVM）。
- 用に準備されたワーカーノード"NFSまたはiSCSI"。

EKS AMIタイプに応じて、Amazon LinuxおよびUbuntu（AMIS）で必要なノードの準備手順に従って "Amazon Machine Images の略"ください。

考慮事項

- SMBボリューム：
  - SMBボリュームはドライバのみを使用してサポートされ `ontap-nas`ます。

- SMBボリュームはAstra Trident EKSアドオンではサポートされません。
- Astra Tridentは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみをサポート詳細については、を参照してください "SMBボリュームをプロビジョニングする準備をします"。

- Astra Trident 24.02より前のバージョンでは、自動バックアップが有効になっているAmazon FSxファイルシステム上に作成されたボリュームはTridentで削除できませんでした。Astra Trident 24.02以降でこの問題を回避するには、AWS FSx for ONTAPのバックエンド構成ファイルで、AWS `apiRegion`、AWS、およびAWS `apikey`を `secretKey`指定し `fsxFilesystemID`ます。

> Astra Tridentに対してIAMロールを指定する場合は、、 `apiKey`の `secretKey`各フィールドを`Astra Trident`に対して明示的に指定する必要はありません `apiRegion`。詳細については、を参照してください "FSX（ONTAP の構成オプションと例）"。

認証

Astra Tridentは、2種類の認証モードを提供します。

- クレデンシャルベース（推奨）：クレデンシャルをAWS Secrets Managerに安全に格納します。ファイルシステムのユーザ、またはSVM用に設定されているユーザを使用できます `fsxadmin` `vsadmin` 。

> Astra Tridentは、SVMユーザ、または別の名前で同じロールのユーザとして実行する必要があります `vsadmin`。Amazon FSx for NetApp ONTAPには、ONTAPクラスタユーザに代わる限定的なユーザが `admin`い `fsxadmin`ます。`Astra Trident`での使用を強く推奨します `vsadmin`。

- 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。

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

- "ONTAP NAS認証"
- "ONTAP SAN認証"

詳細情報

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

IAMロールと**AWS Secret**を作成する

KubernetesポッドがAWSリソースにアクセスするように設定するには、明示的なAWSクレデンシャルを指定する代わりに、AWS IAMロールとして認証します。

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

AWS Secret Managerシークレットの作成

この例では、Astra Trident CSIのクレデンシャルを格納するためのAWSシークレットマネージャシークレットを作成します。

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

IAMポリシーの作成

次の例は、AWS CLIを使用してIAMポリシーを作成します。

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

ポリシー**JSON**ファイル：

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

次の例では、EKSでサービスアカウント用のIAMロールを作成します。

```
eksctl create iamserviceaccount --name trident-controller --namespace trident
--cluster <my-cluster> --role-name <AmazonEKS_FSxN_CSI_DriverRole> --role-only
--attach-policy-arn arn:aws:iam::aws:policy/service-
role/AmazonFSxNCSIDriverPolicy --approve
```

Astra Trident をインストール

Astra Tridentは、KubernetesでのAmazon FSx for NetApp ONTAPストレージ管理を合理化し、開発者や管理者がアプリケーションの導入に集中できるようにします。

Astra Tridentは次のいずれかの方法でインストールできます。

- Helm
- EKSアドオン

```
If you want to make use of the snapshot functionality, install the CSI
snapshot controller add-on. Refer to
https://docs.aws.amazon.com/eks/latest/userguide/csi-snapshot-
controller.html.
```

Helmを使用してAstra Tridentをインストール

1. Astra Tridentインストーラパッケージをダウンロード

   Astra Tridentインストーラパッケージには、Tridentオペレータの導入とAstra Tridentのインストールに必要なものがすべて含まれています。最新バージョンのAstra TridentインストーラをGitHubの[Assets]セクションからダウンロードして展開します。
   ```
   wget https://github.com/NetApp/trident/releases/download/v24.06.0/trident-
   installer-24.06.0.tar.gz
   tar -xf trident-installer-24.06.0.tar.gz
   cd trident-installer
   ```

2. 次の環境変数を使用して、* cloud provider フラグと cloud identity *フラグの値を設定します。

   ```
   export CP="AWS"
   export CI="'eks.amazonaws.com/role-arn:
   arn:aws:iam::<accountID>:role/<AmazonEKS_FSxN_CSI_DriverRole>'"
   ```

   次の例では、Astra Tridentをインストールし、フラグを `$CP`、 `cloud-identity`に `$CI`設定してい `cloud-provider`ます。

   ```
   helm install trident trident-operator-100.2406.0.tgz --set
   cloudProvider=$CP --set cloudIdentity=$CI --namespace trident
   ```

コマンドを使用して、名前、ネームスペース、グラフ、ステータス、アプリケーションのバージョン、リビジョン番号など、インストールの詳細を確認できます `helm list`。

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                           APP VERSION

trident-operator    trident     1           2024-10-14 14:31:22.463122
+0300 IDT    deployed     trident-operator-100.2406.1    24.06.1
```

EKSアドオンを使用して**Astra Trident**をインストール

Astra Trident EKSアドオンには、最新のセキュリティパッチ、バグ修正が含まれており、AWSによってAmazon EKSとの連携が検証されています。EKSアドオンを使用すると、Amazon EKSクラスタの安全性と安定性を一貫して確保し、アドオンのインストール、構成、更新に必要な作業量を削減できます。

前提条件

AWS EKS用のAstra Tridentアドオンを設定する前に、次の条件を満たしていることを確認してください。

- アドオンサブスクリプションがあるAmazon EKSクラスタアカウント
- AWS MarketplaceへのAWS権限：
  `"aws-marketplace:ViewSubscriptions",`
  `"aws-marketplace:Subscribe",`
  `"aws-marketplace:Unsubscribe`
- AMIタイプ：Amazon Linux 2（AL2_x86_64）またはAmazon Linux 2 ARM（AL2_Linux_64 ARM）
- ノードタイプ：AMDまたはARM
- 既存のAmazon FSx for NetApp ONTAPファイルシステム

AWS向けの**Astra Trident**アドオンを有効にする

EKSクラスタ

次のコマンド例は、Astra Trident EKSアドオンをインストールします。

```
eksctl create addon --cluster clusterName --name netapp_trident-operator
--version v24.6.1-eksbuild
eksctl create addon --cluster clusterName --name netapp_trident-operator
--version v24.6.1-eksbuild.1 （専用バージョンを使用）
```

オプションのパラメータを設定する場合 `cloudIdentity`は、EKSアドオンを使用してTridentをインストールするときにを指定して `cloudProvider`ください。

管理コンソール

1. でAmazon EKSコンソールを開きます https://console.aws.amazon.com/eks/home#/clusters。
2. 左側のナビゲーションペインで、*[クラスタ]*をクリックします。
3. NetApp Trident CSIアドオンを設定するクラスタの名前をクリックします。
4. をクリックし、[その他のアドオンの入手]*をクリックします。
5. [* S * elect add-ons *]ページで、次の手順を実行します。
   a. [AWS Marketplace EKS-addons]セクションで、* Astra Trident by NetApp *チェックボックスを選択します。
   b. 「 * 次へ * 」をクリックします。
6. [Configure selected add-ons* settings]ページで、次の手順を実行します。
   a. 使用する*バージョン*を選択します。
   b. では、[Not set]*のままにします。
   c. *オプションの構成設定*を展開し、*アドオン構成スキーマ*に従って、*構成値*セクションのconfigurationValuesパラメーターを前の手順で作成したrole-arnに設定します（値は次の形式にする必要があります `eks.amazonaws.com/role-arn: arn:aws:iam::464262061435:role/AmazonEKS_FSXN_CSI_DriverRole`）。[Conflict resolution method]で[Override]を選択すると、既存のアドオンの1つ以上の設定をAmazon EKSアドオン設定で上書きできます。このオプションを有効にしない場合、既存の設定と競合すると、操作は失敗します。表示されたエラーメッセージを使用して、競合のトラブルシューティングを行うことができます。このオプションを選択する前に、Amazon EKSアドオンが自己管理に必要な設定を管理していないことを確認してください。

      オプションのパラメータを設定する場合 `cloudIdentity`は、EKSアドオンを使用してTridentをインストールするときにを指定して `cloudProvider`ください。

7. [次へ]*を選択します。
8. [確認して追加]ページで、*[作成]*を選択します。

   アドオンのインストールが完了すると、インストールされているアドオンが表示されます。

AWS CLI

1. ファイルを作成し `add-on.json` ます。

```
add-on.json
{

   "clusterName": "<eks-cluster>",
   "addonName": "netapp_trident-operator",
   "addonVersion": "v24.6.1-eksbuild.1",
   "serviceAccountRoleArn": "arn:aws:iam::123456:role/astratrident-
role",
   "configurationValues": "{"cloudIdentity":
"'eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/astratrident-
role'",
   "cloudProvider": "AWS"}"
}
```

オプションのパラメータを設定する場合 `cloudIdentity`は `cloudProvider`、EKSアドオンを使用したTridentのインストール時にとしてを指定して`AWS`ください。

2. Install the Astra<xmt-block0> Trident</xmt-block> EKS add-on

```
aws eks create-addon --cli-input-json file://add-on.json
```

Astra Trident EKSアドオンの更新

EKSクラスタ

- お使いのFSxN Trident CSIアドオンの現在のバージョンを確認してください。をクラスタ名に置き換え `my-cluster` ます。
  ```
  eksctl get addon --name netapp_trident-operator --cluster my-cluster
  ```

  出力例：

```
NAME                          VERSION              STATUS    ISSUES
IAMROLE    UPDATE AVAILABLE    CONFIGURATION VALUES
netapp_trident-operator    v24.6.1-eksbuild.1    ACTIVE    0
{"cloudIdentity":"'eks.amazonaws.com/role-arn:
arn:aws:iam::139763910815:role/AmazonEKS_FSXN_CSI_DriverRole'"}
```

- 前の手順の出力でupdate availableで返されたバージョンにアドオンを更新します。
  ```
  eksctl update addon --name netapp_trident-operator --version v24.6.1-
  eksbuild.1 --cluster my-cluster --force
  ```

  オプションを削除し、いずれかのAmazon EKSアドオン設定が既存の設定と競合している場合 `--force` 、Amazon EKSアドオンの更新は失敗します。競合の解決に役立つエラーメッセージが表示されます。このオプションを指定する前に、管理する必要がある設定がAmazon EKSアドオンで管理されていないことを確認してください。これらの設定はこのオプションで上書きされます。この設定のその他のオプションの詳細については、を参照してください "アドオン"。Amazon EKS Kubernetesフィールド管理の詳細については、を参照してください "Kubernetesフィールド管理"。

管理コンソール

1. Amazon EKSコンソールを開き https://console.aws.amazon.com/eks/home#/clustersます。
2. 左側のナビゲーションペインで、*[クラスタ]*をクリックします。
3. NetApp Trident CSIアドオンを更新するクラスタの名前をクリックします。
4. [アドオン]タブをクリックします。
5. をクリックし、[Edit]*をクリックします。
6. [Configure Astra Trident by NetApp *]ページで、次の手順を実行します。
   a. 使用する*バージョン*を選択します。
   b. （オプション）* Optional configuration settings *を展開し、必要に応じて変更できます。
   c. [変更の保存 *] をクリックします。

AWS CLI

次の例では、EKSアドオンを更新します。

```
aws eks update-addon --cluster-name my-cluster netapp_trident-operator vpc-cni
--addon-version v24.6.1-eksbuild.1 \
--service-account-role-arn arn:aws:iam::111122223333:role/role-name
--configuration-values '{}' --resolve-conflicts --preserve
```

Astra Trident EKSアドオンのアンインストールと削除

Amazon EKSアドオンを削除するには、次の2つのオプションがあります。

- クラスタにアドオンソフトウェアを保持–このオプションを選択すると、Amazon EKSによる設定の管理が削除されます。また、Amazon EKSが更新を通知し、更新を開始した後にAmazon EKSアドオンを自動的に更新する機能も削除されます。ただし、クラスタ上のアドオンソフトウェアは保持されます。このオプションを選択すると、アドオンはAmazon EKSアドオンではなく自己管理型インストールになります。このオプションを使用すると、アドオンのダウンタイムは発生しません。アドオンを保持するには、コマンドのオプションをそのまま使用し `--preserve` ます。
- クラスタからアドオンソフトウェアを完全に削除する–クラスターに依存するリソースがない場合にのみ、Amazon EKSアドオンをクラスターから削除することをお勧めします。コマンドからオプションを削除してアドオンを削除し `--preserve delete` ます。

アドオンにIAMアカウントが関連付けられている場合、IAMアカウントは削除されません。

EKSクラスタ

次のコマンドは、Astra Trident EKSアドオンをアンインストールします。

```
eksctl delete addon --cluster K8s-arm --name netapp_trident-operator
```

管理コンソール

1. でAmazon EKSコンソールを開きます https://console.aws.amazon.com/eks/home#/clusters。
2. 左側のナビゲーションペインで、*[クラスタ]*をクリックします。
3. NetApp Trident CSIアドオンを削除するクラスタの名前をクリックします。
4. タブをクリックし、[Astra Trident by NetApp]をクリックします。
5. [削除（ Remove ）] をクリックします。
6. [Remove netapp_trident-operator confirmation]*ダイアログで、次の手順を実行します。
   a. Amazon EKSでアドオンの設定を管理しないようにするには、*[クラスタに保持]*を選択します。クラスタにアドオンソフトウェアを残して、アドオンのすべての設定を自分で管理できるようにする場合は、この手順を実行します。
   b. 「netapp_trident -operator *」と入力します。
   c. [削除（ Remove ）] をクリックします。

AWS CLI

をクラスタの名前に置き換え `my-cluster` 、次のコマンドを実行します。

```
aws eks delete-addon --cluster-name my-cluster --addon-name netapp_trident-operator --preserve
```

ストレージバックエンドの設定

ONTAP SANとNASドライバの統合

バックエンドファイルは、次の例に示すように、AWS Secret Managerに保存されているSVMのクレデンシャ

ル（ユーザ名とパスワード）を使用して作成できます。

YAML

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: tbc-ontap-nas
  svm: svm-name
  aws:
    fsxFilesystemID: fs-xxxxxxxxxx
  credentials:
    name: "arn:aws:secretsmanager:us-west-2:xxxxxxxx:secret:secret-
name"
    type: awsarn
```

JSON

```
{
  "apiVersion": "trident.netapp.io/v1",
  "kind": "TridentBackendConfig",
  "metadata": {
    "name": "backend-tbc-ontap-nas"
  },
  "spec": {
    "version": 1,
    "storageDriverName": "ontap-nas",
    "backendName": "tbc-ontap-nas",
    "svm": "svm-name",
    "aws": {
      "fsxFilesystemID": "fs-xxxxxxxxxx"
    },
    "managementLIF": null,
    "credentials": {
      "name": "arn:aws:secretsmanager:us-west-2:xxxxxxxx:secret:secret-
name",
      "type": "awsarn"
    }
  }
}
```

バックエンドの作成については、次のページを参照してください。

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

FSx for ONTAPドライバの詳細

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

- `ontap-san`：プロビジョニングされる各PVは、それぞれのAmazon FSx for NetApp ONTAPボリューム内のLUNです。ブロックストレージに推奨されます。
- `ontap-nas`：プロビジョニングされる各PVは、完全なAmazon FSx for NetApp ONTAPボリュームです。NFSとSMBで推奨されます。
- `ontap-san-economy`：プロビジョニングされた各PVは、Amazon FSx for NetApp ONTAPボリュームごとに設定可能なLUN数を持つLUNです。
- `ontap-nas-economy`：プロビジョニングされる各PVはqtreeであり、Amazon FSx for NetApp ONTAPボリュームごとにqtree数を設定できます。
- `ontap-nas-flexgroup`：プロビジョニングされる各PVは、完全なAmazon FSx for NetApp ONTAP FlexGroupボリュームです。

ドライバの詳細については、およびを参照して"NASドライバ""SANドライバ"ください。

構成例

Secret Managerを使用したAWS FSx for ONTAPの設定

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: tbc-ontap-nas
  svm: svm-name
  aws:
    fsxFilesystemID: fs-xxxxxxxxxx
  managementLIF:
  credentials:
    name: "arn:aws:secretsmanager:us-west-2:xxxxxxxx:secret:secret-
name"
    type: awsarn
```

SMBホリユウムノストレエシクラスノセツテイ

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

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

バックエンドの高度な設定と例

バックエンド設定オプションについては、次の表を参照してください。

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

パラメータ	説明	例
managementLIF	クラスタまたはSVM管理LIFのIPアドレス完全修飾ドメイン名（FQDN）を指定できます。IPv6フラグを使用してAstra Tridentをインストールした場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、[28e8：d9fb：a825：b7bf：69a8：d02f：9e7b：3555]などの角かっこで定義する必要があります。フィールドでを指定する場合は fsxFilesystemID aws 、を指定する必要はありませ managementLIF ん。Astra TridentはAWSからSVM情報を取得するためです。 managementLIF`そのため、SVMの下のユーザ（vsadminなど）のクレデンシャルを指定し、そのユーザにロールが割り当てられている必要があり `vsadmin ます。	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」
dataLIF	プロトコル LIF の IP アドレス。* ONTAP NASドライバ*：データLIFを指定することを推奨します。指定しない場合は、Astra TridentがSVMからデータLIFを取得します。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。初期設定後に変更できます。を参照してください 。* ONTAP SANドライバ*: iSCSIには指定しないでくださいTridentがONTAP の選択的LUNマップを使用して、マルチパスセッションの確立に必要なiSCSI LIFを検出します。データLIFが明示的に定義されている場合は警告が生成されます。IPv6フラグを使用してAstra Tridentをインストールした場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、[28e8：d9fb：a825：b7bf：69a8：d02f：9e7b：3555]などの角かっこで定義する必要があります。	

パラメータ	説明	例
`autoExportPolicy`	エクスポートポリシーの自動作成と更新を有効にします[ブーリアン]。オプションと `autoExportCIDRs`オプションを使用する `autoExportPolicy`と、Astra Tridentでエクスポートポリシーを自動的に管理できます。	`false`
`autoExportCIDRs`	が有効な場合にKubernetesのノードIPをフィルタリングするCIDRのリスト `autoExportPolicy`。オプションと `autoExportCIDRs`オプションを使用する `autoExportPolicy`と、Astra Tridentでエクスポートポリシーを自動的に管理できます。	「[0.0.0.0/0]、「::/0」」」
`labels`	ボリュームに適用する任意のJSON 形式のラベルのセット	""
`clientCertificate`	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""
`clientPrivateKey`	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""
`trustedCACertificate`	信頼された CA 証明書の Base64 エンコード値。オプション。証明書ベースの認証に使用されます。	""
`username`	クラスタまたはSVMに接続するためのユーザ名。クレデンシャルベースの認証に使用されます。たとえば、vsadminのように指定します。	
`password`	クラスタまたはSVMに接続するためのパスワード。クレデンシャルベースの認証に使用されます。	
`svm`	使用する Storage Virtual Machine	SVM管理LIFが指定されている場合に生成されます。
`storagePrefix`	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。作成後に変更することはできません。このパラメータを更新するには、新しいバックエンドを作成する必要があります。	`trident`

パラメータ	説明	例
limitAggregateUsage	* Amazon FSx for NetApp ONTAP には指定しないでください。*指定されたと vsadmin`には `fsxadmin、アグリゲートの使用量を取得し、Astra Tridentを使用してアグリゲートを制限するために必要な権限が含まれていません。	使用しないでください。
limitVolumeSize	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。また、qtreeおよびLUNに対して管理するボリュームの最大サイズを制限し、オプションを使用すると、`qtreesPerFlexvol`FlexVolあたりのqtreeの最大数をカスタマイズできます。	“”（デフォルトでは適用されません）
lunsPerFlexvol	FlexVol あたりの最大LUN数。有効な範囲は50、200です。SANのみ。	“100”
debugTraceFlags	トラブルシューティング時に使用するデバッグフラグ。例： {"api"：false、"method"：true} 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以外は使用しない `debugTraceFlags`でください。	null
nfsMountOptions	NFSマウントオプションをカンマで区切ったリスト。Kubernetes永続ボリュームのマウントオプションは通常はストレージクラスで指定されますが、ストレージクラスでマウントオプションが指定されていない場合、Astra Tridentはストレージバックエンドの構成ファイルで指定されているマウントオプションを使用します。ストレージクラスや構成ファイルにマウントオプションが指定されていない場合、Astra Tridentは関連付けられた永続的ボリュームにマウントオプションを設定しません。	""
nasType	NFSボリュームまたはSMBボリュームの作成を設定オプションは nfs、、 smb`またはnullです。*SMBボリュームの場合はに設定する必要があります `smb。*nullに設定すると、デフォルトでNFSボリュームが使用されます。	nfs
qtreesPerFlexvol	FlexVol あたりの最大 qtree 数。有効な範囲は [50 、 300] です。	"200"

パラメータ	説明	例
smbShare	次のいずれかを指定できます。Microsoft管理コンソールまたはONTAP CLIを使用して作成されたSMB共有の名前、またはAstra TridentでSMB共有を作成できるようにする名前。このパラメータは、Amazon FSx for ONTAPバックエンドに必要です。	smb-share
useREST	ONTAP REST API を使用するためのブーリアンパラメータ。技術プレビュー useREST は技術プレビューとして提供されており、本番環境のワークロードには推奨されません。に設定する true`と、Astra TridentはONTAP REST APIを使用してバックエンドと通信します。この機能にはONTAP 9.11.1以降が必要です。また、使用するONTAPログインロールには、アプリケーションへのアクセス権が必要です `ontap。これは、事前に定義された役割と役割によって実現され vsadmin cluster-admin ます。	false
aws	AWS FSx for ONTAPの構成ファイルでは次のように指定できます。-：AWS FSxファイルシステムのIDを指定します。fsxFilesystemID- apiRegion：AWS APIリージョン名。-apikey：AWS APIキー。-secretKey：AWSシークレットキー。	"" "" ""
credentials	AWS Secret Managerに保存するFSx SVMのクレデンシャルを指定します。- name：シークレットのAmazonリソース名（ARN）。SVMのクレデンシャルが含まれています。- type：に設定します awsarn。詳細については、を参照してください "AWS Secrets Managerシークレットの作成" 。	

ボリュームのプロビジョニング用のバックエンド構成オプション

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

パラメータ	説明	デフォルト
`spaceAllocation`	space-allocation for LUN のコマンドを指定します	`true`
`spaceReserve`	スペースリザベーションモード：「 none 」（シン）または「 volume 」（シック）	`none`
`snapshotPolicy`	使用する Snapshot ポリシー	`none`
`qosPolicy`	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプールまたはバックエンドごとに、QOSPolicyまたはadaptiveQosPolicyのいずれかを選択します。Trident が Astra で QoS ポリシーグループを使用するには、 ONTAP 9.8 以降が必要です。非共有のQoSポリシーグループを使用して、各コンスティチュエントに個別にポリシーグループを適用することを推奨します。共有 QoS ポリシーグループにより、すべてのワークロードの合計スループットに対して上限が適用されます。	「」
`adaptiveQosPolicy`	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプールまたはバックエンドごとに、QOSPolicyまたはadaptiveQosPolicyのいずれかを選択します。経済性に影響する ONTAP - NAS ではサポートされません。	「」
`snapshotReserve`	スナップショット "0" 用に予約されたボリュームの割合	がの `none`場合 `snapshotPolicy else`
`splitOnClone`	作成時にクローンを親からスプリットします	`false`
`encryption`	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです。`false`このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。NAEがバックエンドで有効になっている場合は、Astra TridentでプロビジョニングされたすべてのボリュームがNAEに有効になります。詳細については、を参照してください"Astra TridentとNVEおよびNAEの相互運用性"。	`false`

パラメータ	説明	デフォルト
`luksEncryption`	LUKS暗号化を有効にします。を参照してください "Linux Unified Key Setup（LUKS；統合キーセットアップ）を使用"。SANのみ。	""
`tieringPolicy`	使用する階層化ポリシー `none`	`snapshot-only`ONTAP 9 .5より前のSVM-DR構成
`unixPermissions`	新しいボリュームのモード。* SMBボリュームは空にしておきます。*	「」
`securityStyle`	新しいボリュームのセキュリティ形式。NFSのサポート `mixed`と`unix`セキュリティ形式SMBのサポート `mixed`と `ntfs`セキュリティ形式。	NFSのデフォルトはです `unix`。SMBのデフォルトはです `ntfs`。

SMBボリュームをプロビジョニングする準備をします

ドライバを使用してSMBボリュームをプロビジョニングできます `ontap-nas`。完了する前に、次の手順を実行してONTAP SANとNASドライバの統合ください。

開始する前に

ドライバを使用してSMBボリュームをプロビジョニングする `ontap-nas`には、次の準備が必要です。

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

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

- Windowsサービスとして設定されたCSIプロキシ。を設定するには `csi-proxy`、Windowsで実行されているKubernetesノードについて、またはを"GitHub: Windows向けCSIプロキシ"参照してください"GitHub: CSIプロキシ"。

手順

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

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

   b. 指定したSVMに関連付けられているSMB共有を作成します。

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

c. 共有が作成されたことを確認します。

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

詳細については、を参照して"SMB共有を作成する"ください。

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

パラメータ	説明	例
`smbShare`	次のいずれかを指定できます。Microsoft管理コンソールまたはONTAP CLIを使用して作成されたSMB共有の名前、またはAstra TridentでSMB共有を作成できるようにする名前。このパラメータは、Amazon FSx for ONTAPバックエンドに必要です。	`smb-share`
`nasType`	*に設定する必要があります `smb`。*nullの場合、デフォルトは `nfs`になります。	`smb`
`securityStyle`	新しいボリュームのセキュリティ形式。* SMBボリュームの場合は `ntfs`または `mixed`に設定する必要があります。*	`ntfs`または `mixed`
`unixPermissions`	新しいボリュームのモード。* SMBボリュームは空にしておく必要があります。*	""

ストレージクラスと**PVC**を設定する

Kubernetes StorageClassオブジェクトを設定してストレージクラスを作成し、Astra Tridentでボリュームのプロビジョニング方法を指定設定したKubernetes StorageClassを使用してPVへのアクセスを要求するPersistentVolume（PV）とPersistentVolumeClaim（PVC）を作成します。その後、PVをポッドにマウントできます。

ストレージクラスを作成する。

Kubernetes StorageClassオブジェクトの設定

は、 "Kubernetes StorageClassオブジェクト" そのクラスで使用されるプロビジョニングツールとしてAstra Tridentを示し、Astra Tridentにボリュームのプロビジョニング方法を指示します。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
```

ストレージクラスとパラメータおよびパラメータとの連携によるAstra Tridentによるボリュームのプロビジョニング方法の詳細については `PersistentVolumeClaim`、を参照して"Kubernetes オブジェクトと Trident オブジェクト"ください。

ストレージクラスを作成する。

手順

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

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

2. 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
```

PVおよびPVCの作成

"永続ボリューム_"（PV）は、Kubernetesクラスタ上のクラスタ管理者によってプロビジョニングされる物理ストレージリソースです。 "PersistentVolumeClaim_"（PVC）は、クラスタ上のPersistentVolumeへのアクセス要求です。

PVCは、特定のサイズまたはアクセスモードのストレージを要求するように設定できます。クラスタ管理者は、関連付けられているStorageClassを使用して、PersistentVolumeのサイズとアクセスモード（パフォーマンスやサービスレベルなど）以上を制御できます。

PVとPVCを作成したら、ポッドにボリュームをマウントできます。

マニフェストの例

PersistentVolumeサンプルマニフェスト

このサンプルマニフェストは、StorageClassに関連付けられた10Giの基本PVを示しています `basic-csi`。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
  labels:
    type: local
spec:
  storageClassName: basic-csi
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/my/host/path"
```

PersistentVolumeClaimサンプルマニフェスト

次に、基本的なPVC設定オプションの例を示します。

RWOアクセスを備えたPVC

この例は、という名前のStorageClassに関連付けられたRWXアクセスを持つ基本的なPVCを示しています `basic-csi`。

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

NVMe / TCP対応PVC

この例は、という名前のStorageClassに関連付けられたNVMe/TCPの基本的なPVCとRWOアクセスを示しています `protection-gold`。

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

PVおよびPVCの作成

手順

1. PVを作成

```
kubectl create -f pv.yaml
```

2. PVステータスを確認します。

```
kubectl get pv
NAME         CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM
STORAGECLASS  REASON  AGE
pv-storage  4Gi       RWO           Retain          Available
7s
```

3. PVCを作成

```
kubectl create -f pvc.yaml
```

4. PVCステータスを確認します。

```
kubectl get pvc
NAME         STATUS VOLUME     CAPACITY ACCESS MODES STORAGECLASS AGE
pvc-storage Bound  pv-name 2Gi      RWO                          5m
```

ストレージクラスとパラメータおよびパラメータとの連携によるAstra Tridentによるボリュームのプロビジョニング方法の詳細については `PersistentVolumeClaim`、を参照して"Kubernetes オブジェクトと Trident オブジェクト"ください。

Astra Tridentの属性

これらのパラメータによって、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Astra Tridentで管理されるストレージプールが決まります。

属性	タイプ	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

属性	タイプ	値	提供	リクエスト	でサポートされます
プロビジョニングタイプ	文字列	シン、シック	プールはこのプロビジョニング方法をサポートします	プロビジョニング方法が指定されました	シック：All ONTAP ；thin ：All ONTAP & solidfire-san-SAN
backendType	文字列	ONTAPNAS、ONTAPNASエコノミー、ONTAP-NAS-flexgroup 、ONTAPSAN、solidfire-san-SAN、solidfire-san-SAN、GCP-cvs、azure-NetApp-files 、ONTAP-SAN-bエコノミー	プールはこのタイプのバックエンドに属しています	バックエンドが指定されて	すべてのドライバ
Snapshot	ブール値	true false	プールは、Snapshot を含むボリュームをサポートします	Snapshot が有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
クローン	ブール値	true false	プールはボリュームのクローニングをサポートします	クローンが有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
暗号化	ブール値	true false	プールでは暗号化されたボリュームをサポート	暗号化が有効なボリューム	ONTAP-NAS 、ONTAP-NAS-エコノミー 、ONTAP-NAS-FlexArray グループ、 ONTAP-SAN
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

サンプルアプリケーションのデプロイ

サンプルアプリケーションをデプロイします。

手順

1. ボリュームをポッドにマウントします。

```
kubectl create -f pv-pod.yaml
```

次に、PVCをポッドに接続するための基本的な設定例を示します。基本設定：

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: pv-storage
      persistentVolumeClaim:
       claimName: basic
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: pv-storage
```

進捗状況はを使用して監視でき `kubectl get pod --watch`ます。

2. ボリュームがにマウントされていることを確認します `/my/mount/path`。

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

```
Filesystem                                                      Size
Used Avail Use% Mounted on
192.168.188.78:/trident_pvc_ae45ed05_3ace_4e7c_9080_d2a83ae03d06  1.1G
320K  1.0G   1% /my/mount/path
```

1. ポッドを削除できるようになりました。Podアプリケーションは存在しなくなりますが、ボリュームは残ります。

```
kubectl delete pod task-pv-pod
```

EKSクラスタでの**Astra Trident EKS**アドオンの設定

Astra Tridentは、KubernetesでのAmazon FSx for NetApp ONTAPストレージ管理を合理化し、開発者や管理者がアプリケーションの導入に集中できるようにします。Astra Trident EKSアドオンには、最新のセキュリティパッチ、バグ修正が含まれており、AWSによってAmazon EKSとの連携が検証されています。EKSアドオンを使用すると、Amazon EKSクラスタの安全性と安定性を一貫して確保し、アドオンのインストール、構成、更新に必要な作業量を削減できます。

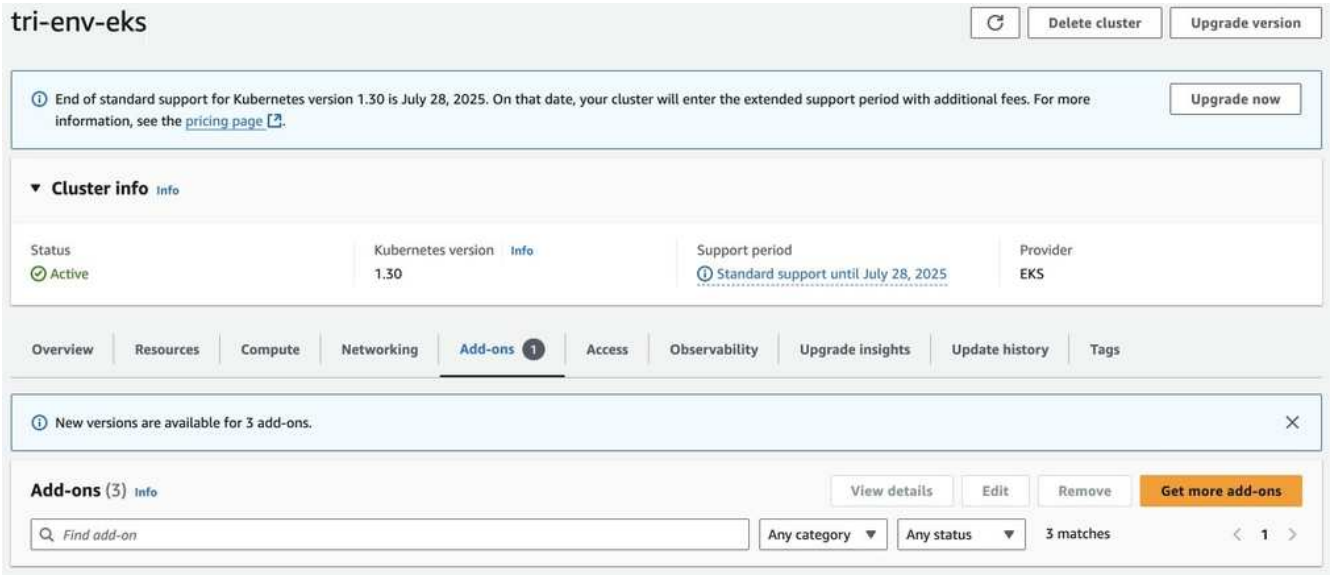
前提条件

AWS EKS用のAstra Tridentアドオンを設定する前に、次の条件を満たしていることを確認してください。

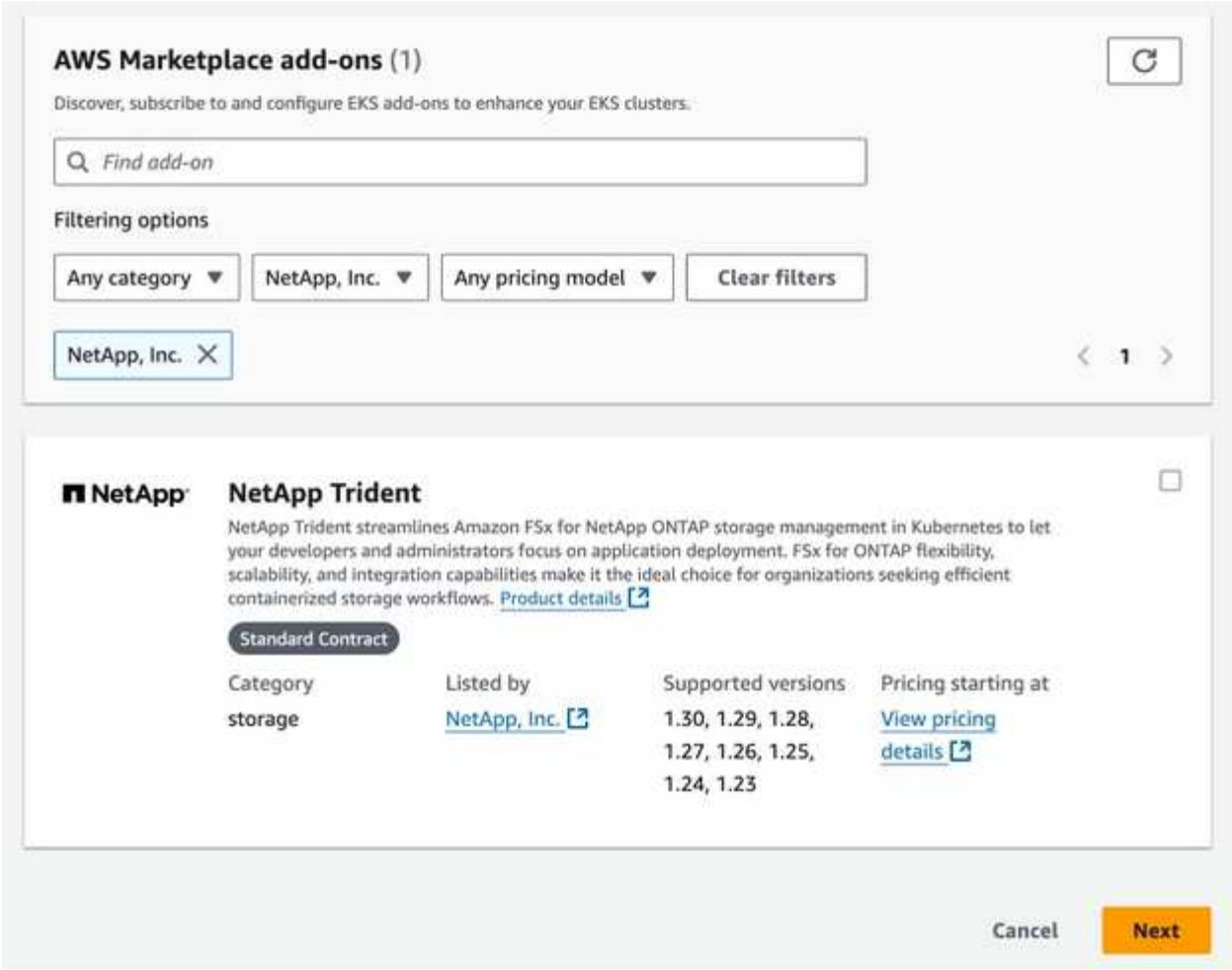
- アドオンサブスクリプションがあるAmazon EKSクラスタアカウント
- AWS MarketplaceへのAWS権限：
  `"aws-marketplace:ViewSubscriptions",`
  `"aws-marketplace:Subscribe",`
  `"aws-marketplace:Unsubscribe`
- AMIタイプ：Amazon Linux 2（AL2_x86_64）またはAmazon Linux 2 ARM（AL2_Linux_64 ARM）
- ノードタイプ：AMDまたはARM
- 既存のAmazon FSx for NetApp ONTAPファイルシステム

手順

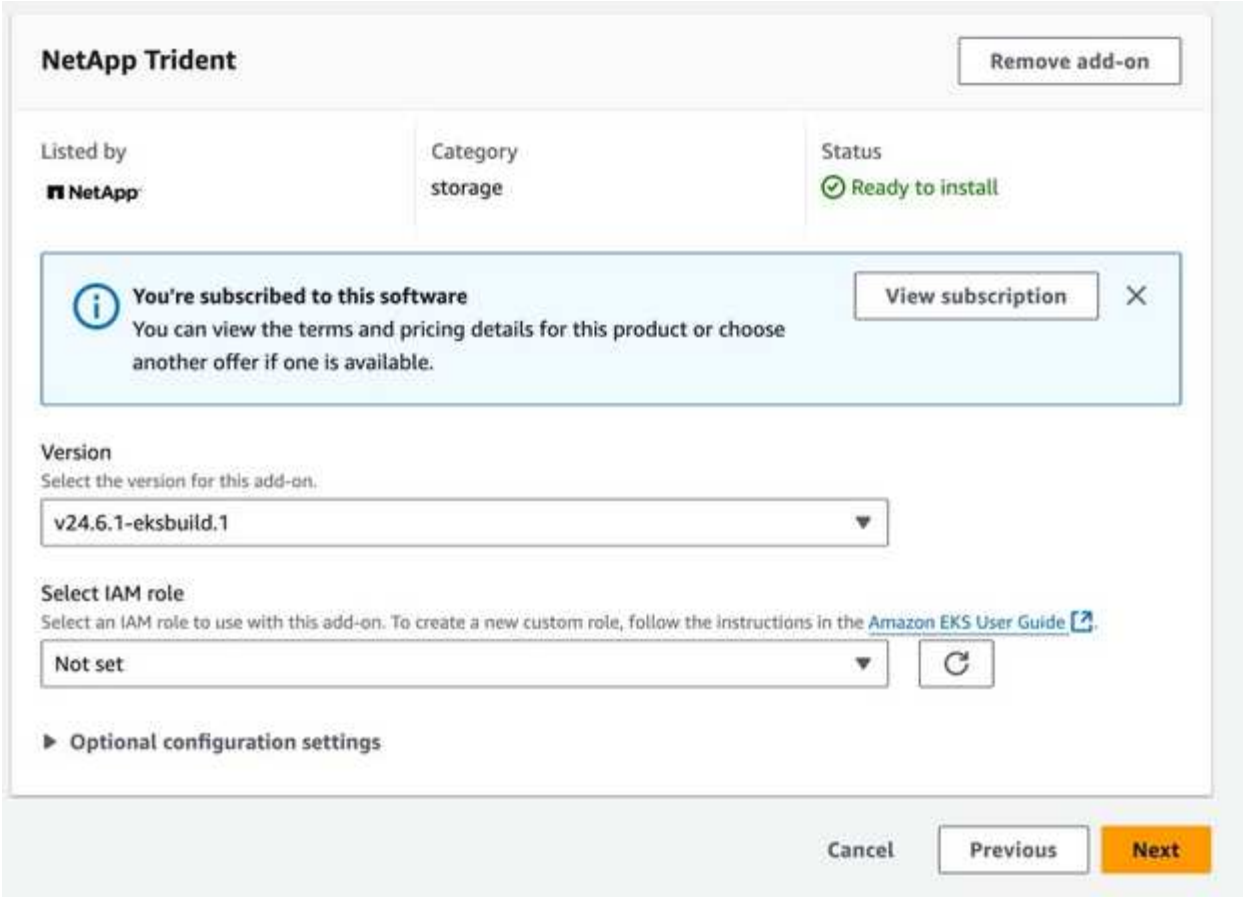
1. EKS Kubernetesクラスタで、*アドオン*タブに移動します。



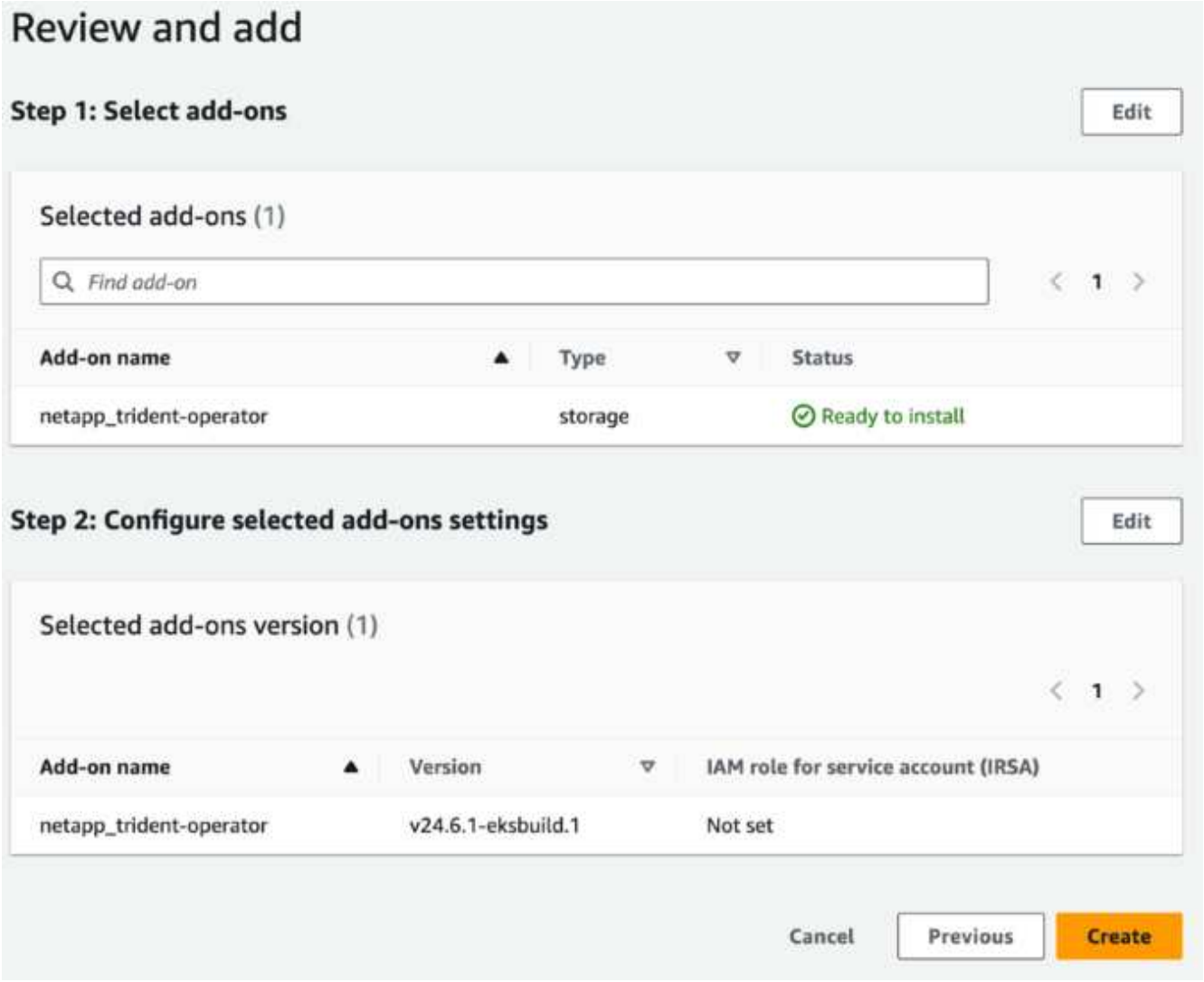
2. [AWS Marketplace add-ons]*にアクセスし、_storage_categoryを選択します。



3. NetApp Trident *を探し、Astra Tridentアドオンのチェックボックスを選択します。
4. 必要なアドオンのバージョンを選択します。



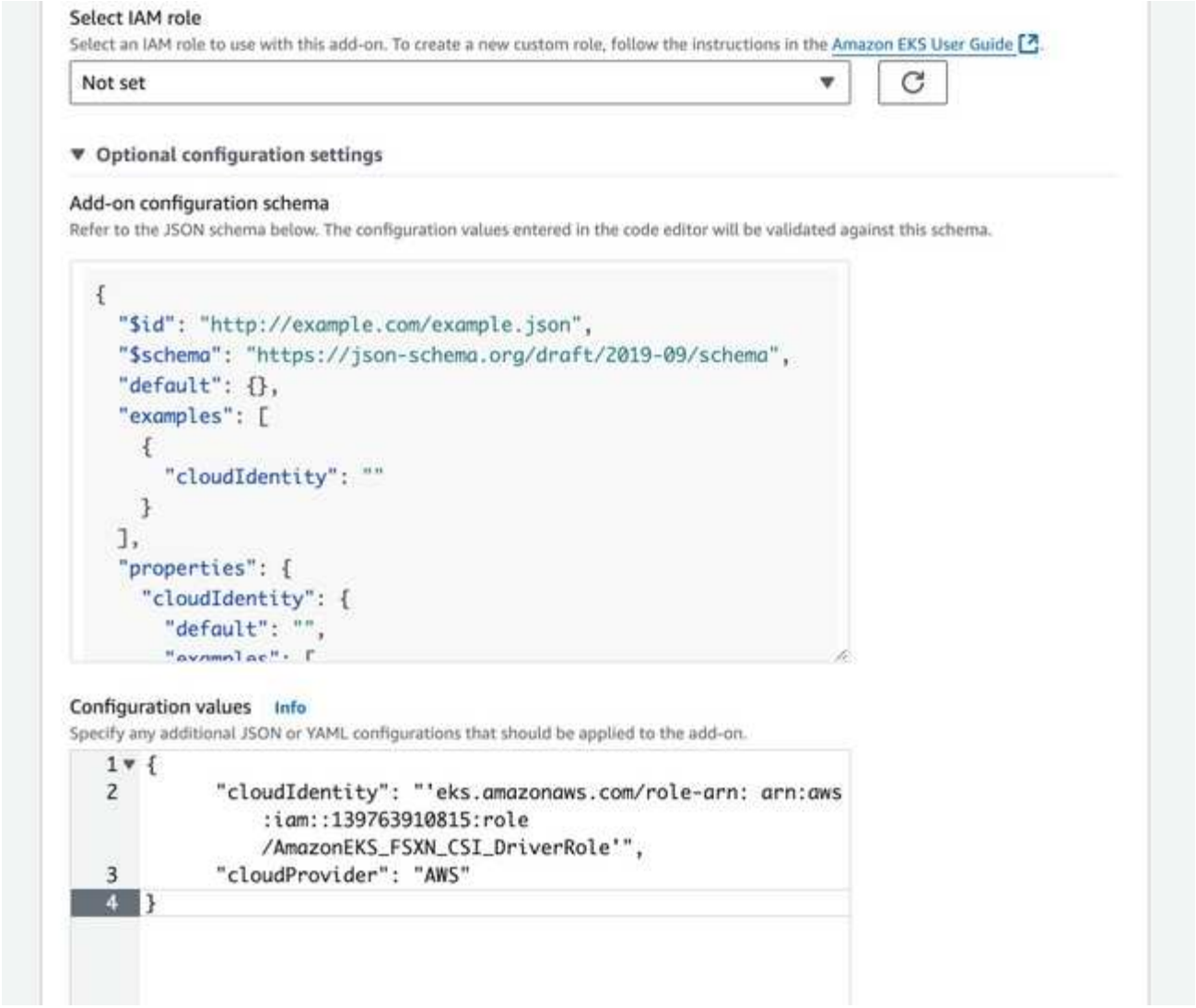
5. ノードから継承するIAMロールオプションを選択します。



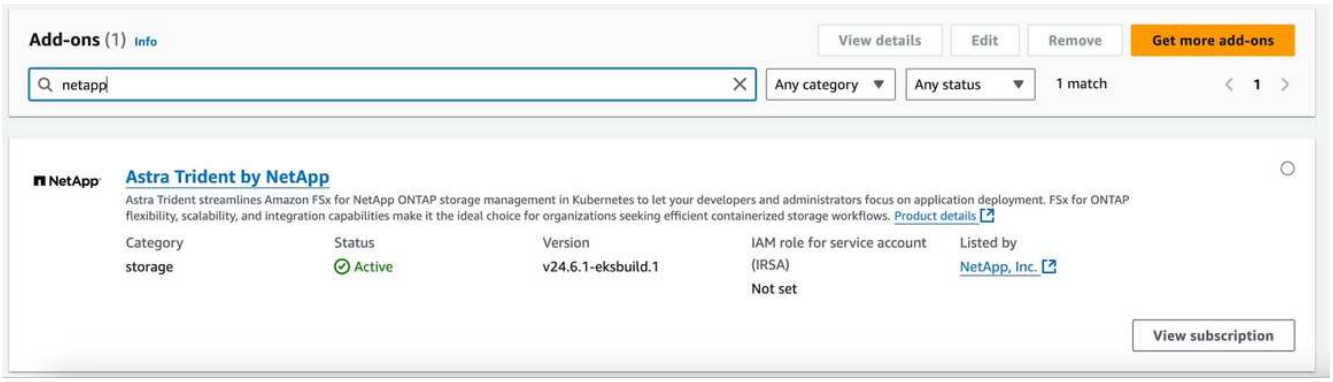
6. （オプション）必要に応じてオプションの設定を行い、* Next *を選択します。

Add-on構成スキーマ*に従って、* Configuration Values *セクションのconfigurationValuesパラメーターを前の手順で作成したrole-arnに設定します（値は次の形式にする必要があります `eks.amazonaws.com/role-arn: arn:aws:iam::464262061435:role/AmazonEKS_FSXN_CSI_DriverRole`）。[Conflict resolution method]で[Override]を選択すると、既存のアドオンの1つ以上の設定をAmazon EKSアドオン設定で上書きできます。このオプションを有効にしない場合、既存の設定と競合すると、操作は失敗します。表示されたエラーメッセージを使用して、競合のトラブルシューティングを行うことができます。このオプションを選択する前に、Amazon EKSアドオンが自己管理に必要な設定を管理していないことを確認してください。

オプションのパラメータを設定する場合 `cloudIdentity`は `cloudProvider`、EKSアドオンを使用したTridentのインストール時にとしてを指定して `AWS`ください。



7. 「* Create *」を選択します。
8. アドオンのステータスが_Active_であることを確認します。



CLIを使用した**Astra Trident EKS**アドオンのインストールとアンインストール

CLIを使用して**Astra Trident EKS**アドオンをインストールします。

次のコマンド例は、Astra Trident EKSアドオンをインストールします（専用バージョンを使用）。

```
eksctl create addon --cluster K8s-arm --name netapp_trident-operator --version
```

```
v24.6.1-eksbuild
eksctl create addon --cluster clusterName --name netapp_trident-operator
--version v24.6.1-eksbuild.1
```

オプションのパラメータを設定する場合 `cloudIdentity`は、EKSアドオンを使用してTridentをインストールするときにを指定して `cloudProvider`ください。

CLIを使用して**Astra Trident EKS**アドオンをアンインストールします。

次のコマンドは、Astra Trident EKSアドオンをアンインストールします。

```
eksctl delete addon --cluster K8s-arm --name netapp_trident-operator
```

kubectl を使用してバックエンドを作成します

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

`TridentBackendConfig`

`TridentBackendConfig`(tbc `tbconfig`、 `tbackendconfig`）は、を使用してAstra Tridentバックエンドを管理できるフロントエンドのネームスペースCRDです。 `kubectl`Kubernetes管理者やストレージ管理者は、Kubernetes CLIを使用して直接バックエンドを作成、管理できるようになりまし(`tridentctl`た。専用のコマンドラインユーティリティは必要ありません）。

オブジェクトを作成すると、 `TridentBackendConfig`次の処理が実行されます。

- 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 (`tbe`、 `tridentbackend`）CRとして表され `TridentBackend`ます。
- は `TridentBackendConfig`、Astra Tridentで作成されたに一意にバインドされます `TridentBackend`。

それぞれが `TridentBackendConfig`との1対1のマッピングを保持し `TridentBackend`ます。前者はバックエンドを設計および設定するためにユーザーに提供されるインターフェイスです。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には、オブジェクトを変更し `TridentBackendConfig`ます。

CRの形式については、次の例を参照して `TridentBackendConfig`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
```

必要なストレージプラットフォーム/サービスの設定例については、ディレクトリにある例を参照し "Trident インストーラ"てください。

は spec、バックエンド固有の設定パラメータを取得します。この例では、バックエンドでストレージドライバを使用し ontap-san、次の表に示す設定パラメータを使用しています。ご使用のストレージドライバの設定オプションのリストについては、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"。

この `spec`セクションには、CRで新たに導入されたフィールドと `deletionPolicy`フィールド `TridentBackendConfig`も含まれてい `credentials`ます。

- `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するクレデンシャルが含まれます。ユーザが作成した Kubernetes Secret に設定されます。クレデンシャルをプレーンテキストで渡すことはできないため、エラーになります。
- `deletionPolicy`：このフィールドは、が削除されたときの動作を定義します `TridentBackendConfig`。次の 2 つの値のいずれかを指定できます。
  - `delete`:これにより、CRと関連するバックエンドの両方が削除され `TridentBackendConfig`ます。これがデフォルト値です。
  - `retain`：CRが削除されても、 `TridentBackendConfig`バックエンド定義は引き続き存在し、で管理できます。 `tridentctl`削除ポリシーをに設定する `retain`と、ユーザは以前のリリース（`21.04`より前のリリース）にダウングレードして、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、の作成後に更新できます `TridentBackendConfig`。

> バックエンドの名前はを使用して設定され `spec.backendName`ます。指定しない場合、バックエンドの名前はオブジェクトの名前（metadata.name）に設定され `TridentBackendConfig`ます。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 `spec.backendName`ます。

> で作成されたバックエンドに `tridentctl`は、関連付けられたオブジェクトはありません `TridentBackendConfig`。このようなバックエンドをで管理するには、 `kubectl`CRを作成し `TridentBackendConfig`ます。同一の設定パラメータ（、、 `spec.storagePrefix spec.storageDriverName`など）を指定するように注意する必要があります `spec.backendName`。Astra Tridentは、新しく作成されたを既存のバックエンドに自動的にバインドし `TridentBackendConfig`ます。

手順の概要

を使用して新しいバックエンドを作成するには `kubectl`、次の手順を実行します。

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

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	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

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
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。および `ontap-san-economy`の場合 `ontap-san`
ONTAP	chapInitiatorSecret	CHAP イニシエータシークレット。useCHAP = true の場合は必須。および `ontap-san-economy`の場合 `ontap-san`
ONTAP	chapTargetUsername のコマンド	ターゲットユーザ名。useCHAP = true の場合は必須。および `ontap-san-economy`の場合 `ontap-san`
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。および `ontap-san-economy`の場合 `ontap-san`

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

ステップ**2**：**CR**を作成する `TridentBackendConfig`

これでCRを作成する準備ができ `TridentBackendConfig`ました。この例では、ドライバを使用するバックエンドが `ontap-san`、次のオブジェクトを使用して作成され `TridentBackendConfig`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**：**CR**のステータスを確認する `TridentBackendConfig`

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

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

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

フェーズには次のいずれかの値を指定できます。

- `Bound`: `TridentBackendConfig` CRはバックエンドに関連付けられており、そのバックエンドにはCRのuidがセットされ `TridentBackendConfig`てい `configRef`ます。
- `Unbound`:を使用して表されます `""`。 `TridentBackendConfig`オブジェクトはバックエンドにバインドされていません。デフォルトでは、新しく作成されたすべての `TridentBackendConfig`CRSがこのフェーズになります。フェーズが変更された後、再度 Unbound に戻すことはできません。
- `Deleting`：CR `deletionPolicy`は `TridentBackendConfig`削除するように設定されています。`CR`が削除されると `TridentBackendConfig`、CRは削除ステートに移行します。
  - バックエンドに永続的ボリューム要求（PVC）が存在しない場合は、を削除する `TridentBackendConfig`とAstra TridentでバックエンドとCRが削除され `TridentBackendConfig`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。 `TridentBackendConfig`その後、`CR`は削除フェーズに入ります。バックエンドとは `TridentBackendConfig`、すべてのPVCが削除された後にのみ削除されます。
- `Lost`：CRに関連付けられているバックエンドが `TridentBackendConfig`誤ってまたは故意に削除され、 `TridentBackendConfig`CRには削除されたバックエンドへの参照が残っています。 `TridentBackendConfig`CRは、値に関係なく削除できます `deletionPolicy`。

- `Unknown`：Astra Tridentは、CRに関連付けられたバックエンドの状態または存在を特定できません `TridentBackendConfig`。たとえば、APIサーバが応答していない場合やCRDが見つからない場合 `tridentbackends.trident.netapp.io`などです。これには介入が必要な場合があります

この段階では、バックエンドが正常に作成されます。など、追加で処理できる処理がいくつかあります"バックエンドの更新とバックエンドの削除"。

（オプション）手順 **4** ：詳細を確認します

バックエンドに関する詳細情報を確認するには、次のコマンドを実行します。

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

```
NAME                      BACKEND NAME         BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8   Bound   Success   ontap-san        delete
```

さらに、のyaml/jsonダンプを取得することもできます `TridentBackendConfig`。

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

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

`backendInfo`CRに応答して作成されたバックエンドの `TridentBackendConfig`とが `backendUUID`格納され `backendName`ます。 `lastOperationStatus`フィールドはCRの最後の処理のステータスを表し `TridentBackendConfig`ます。ユーザトリガー（でユーザが変更した場合など）、またはAstra Tridentによってトリガーされた（Astra Tridentの再起動時など）ことができます `spec`。成功または失敗のいずれかです。 `phase`CRとバックエンド間の関係のステータスを表します `TridentBackendConfig`。上の例では、の `phase`値がバインドされています。つまり、CRがバックエンドに関連付けられていることを意味します `TridentBackendConfig`。

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

> を使用して、関連付けられたオブジェクトを `tridentctl`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 `TridentBackendConfig`。とを `TridentBackendConfig`切り替える手順について説明します `tridentctl` "こちらを参照してください"。

バックエンドの管理

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。 kubectl

バックエンドを削除します

を削除することで TridentBackendConfig、Astra Tridentでバックエンドを（に基づいて）削除または保持するように指示し deletionPolicy`ます。バックエンドを削除するには、がdeleteに設定されていることを確認します `deletionPolicy。のみを削除するには TridentBackendConfig、がretainに設定されていることを確認します deletionPolicy。これにより、バックエンドが引き続き存在し、を使用して管理できるようになります tridentctl。

次のコマンドを実行します。

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

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

既存のバックエンドを表示します

次のコマンドを実行します。

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

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 `tridentctl get backend -n trident。このリストには、で作成されたバックエンドも含まれ `tridentctl`ます。

バックエンドを更新します

バックエンドを更新する理由はいくつかあります。

- ストレージシステムのクレデンシャルが変更されている。クレデンシャルを更新するには、オブジェクトで使用されるKubernetes Secretを `TridentBackendConfig`更新する必要があります。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。
  - 次のコマンドを使用して、Kubernetesから直接オブジェクトを更新できます TridentBackendConfig。

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

- または、次のコマンドを使用して既存のCRに変更を加えることもできます `TridentBackendConfig`。

```
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特定するには、またはを `kubectl describe tbc <tbc-name> -n trident` 実行し `kubectl get tbc <tbc-name> -o yaml -n trident` 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

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。 `tridentctl`

バックエンドを作成します

を作成したら"バックエンド構成ファイル"、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、コマンドをもう一度実行できます `create`。

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、コマンドをもう一度実行できます `update`。

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

これは、バックエンドオブジェクト用に出力するJSONで回答できる質問の例 `tridentctl`です。これは、インストールする必要があるユーティリティを使用し `jq`ます。

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

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

の導入により TridentBackendConfig、管理者はバックエンドを2つの独自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。

- を使用して作成したバックエンドはで管理 `TridentBackendConfig`でき `tridentctl`ますか。
- を使用して作成したバックエンドはを使用して管理 `tridentctl`でき `TridentBackendConfig`ますか。

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

このセクションでは、オブジェクトを作成してKubernetesインターフェイスから直接 `TridentBackendConfig`作成されたバックエンドを管理するために必要な手順について説明し `tridentctl`ます。

これは、次のシナリオに該当します。

- を使用して作成された tridentctl`既存のバックエンドにはがありません。 `TridentBackendConfig
- 他のオブジェクトが存在するときに、 TridentBackendConfig`で作成された新しいバックエンド `tridentctl。

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。

- を使用して作成されたバックエンドの管理に引き続き使用し `tridentctl`ます。
- を使用して作成したバックエンドを新しいオブジェクトに TridentBackendConfig`バインドし `tridentctl`ます。これは、バックエンドがではなくを使用して管理されることを意味します `kubectl tridentctl。

を使用して既存のバックエンドを管理するには kubectl、既存のバックエンドにバインドするを作成する必要があります TridentBackendConfig。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. オブジェクトを作成し TridentBackendConfig`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。同一の設定パラメータ（、、 `spec.storagePrefix spec.storageDriverName`など）を指定するように注意する必要があります `spec.backendName。 `spec.backendName`既存のバックエンドの名前に設定する必要があります。

手順 **0** ：バックエンドを特定します

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

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

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

cat ontap-nas-backend.json

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

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

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

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

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

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

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

手順**2**：**CR**を作成する `TridentBackendConfig`

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

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

この場合、は `TridentBackendConfig`次のようになります。

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

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

手順**3**：**CR**のステータスを確認する `TridentBackendConfig`

が作成されたら `TridentBackendConfig`、そのフェーズはにする必要があります `Bound`。また、既存のバックエンドと同じバックエンド名と UUID が反映されている必要があります。

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

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

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

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

`tridentctl`を使用して作成されたバックエンドの一覧表示に使用でき `TridentBackendConfig`ます。さらに、管理者は、を削除して、がに設定されている `retain`ことを確認する `spec.deletionPolicy`ことで、 `TridentBackendConfig`このようなバックエンドを完全に管理することもできます `tridentctl`。

手順 **0** ：バックエンドを特定します

たとえば、次のバックエンドがを使用して作成されたとし `TridentBackendConfig`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、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 `TridentBackendConfig`ます（[Observe the backend's UUUID]）。

手順**1**：**Confirm**がに設定されている `retain`ことを確認 `deletionPolicy`

の価値を見てみましょう `deletionPolicy`。これはに設定する必要があり `retain`ます。これにより、CRが削除されてもバックエンド定義が存在し、で管理できるように `TridentBackendConfig`なり `tridentctl`ます。

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

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

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

がに設定され `retain`ていない場合は、次の手順に進まないで `deletionPolicy`ください。

手順2：CRを削除する `TridentBackendConfig`

最後のステップはCRを削除することです `TridentBackendConfig`。がに設定されている `retain`ことを確認したら `deletionPolicy`、削除を続行できます。

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

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

オブジェクトが削除されると `TridentBackendConfig`、Astra Tridentはバックエンド自体を削除せずにオブジェクトを削除します。

ストレージクラスの作成と管理

ストレージクラスを作成する。

Kubernetes StorageClassオブジェクトを設定してストレージクラスを作成し、Astra Tridentでボリュームのプロビジョニング方法を指定

Kubernetes StorageClassオブジェクトの設定

は "Kubernetes StorageClassオブジェクト"、そのクラスで使用するプロビジョニングツールとしてAstra Tridentを指定し、ボリュームのプロビジョニング方法をAstra Tridentに指示します。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

ストレージクラスとパラメータおよびパラメータとの連携によるAstra Tridentによるボリュームのプロビジョニング方法の詳細については `PersistentVolumeClaim`、を参照して"Kubernetes オブジェクトと Trident オブジェクト"ください。

ストレージクラスを作成する。

StorageClassオブジェクトを作成したら、ストレージクラスを作成できます。[ストレージクラスノサンプル]に、使用または変更できる基本的なサンプルを示します。

手順

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

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

2. 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"
        ]
      }
    }
  ]
}
```

ストレージクラスノサンプル

Astra Tridentの特長 "特定のバックエンド向けのシンプルなストレージクラス定義"

または、インストーラに付属のファイルを編集して、ストレージドライバ名に置き換える *BACKEND_TYPE*`こともできます `sample-input/storage-class-csi.yaml.templ。

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

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

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

ストレージクラスを管理する

既存のストレージクラスを表示したり、デフォルトのストレージクラスを設定したり、ストレージクラスバックエンドを識別したり、ストレージクラスを削除したりできます。

既存のストレージクラスを表示します

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

```
kubectl get storageclass
```

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

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

- 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`storageclass.kubernetes.io/is-default-class`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。

クラスタ内のデフォルトのストレージクラスは一度に1つだけにしてください。Kubernetes では、技術的に複数のストレージを使用することはできますが、デフォルトのストレージクラスがまったくない場合と同様に動作します。

ストレージクラスのバックエンドを特定します

ここでは、Astra Tridentバックエンドオブジェクト用に出力されるJSONを使用して回答できる質問の例を示し`tridentctl`ます。これはユーティリティを使用し`jq`ます。このユーティリティは、最初にインストールする必要がある場合があります。

```
tridentctl get storageclass -o json | jq  '[.items[] | {storageClass:
.Config.name, backends: [.storage]|unique}]'
```

ストレージクラスを削除する

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

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

`<storage-class>`は、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では、作成するボリュームに空白が適用され `fsType`ます。`iSCSI`バックエンドの場合は、`StorageClass`で強制することを推奨します `parameters.fsType`。既存のストレージクラスを削除し、指定したで再作成してください `parameters.fsType`。

ボリュームのプロビジョニングと管理

ボリュームをプロビジョニングする

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

概要

"永続ボリューム_"（PV）は、Kubernetesクラスタ上のクラスタ管理者によってプロビジョニングされる物理ストレージリソースです。 "PersistentVolumeClaim_"（PVC）は、クラスタ上のPersistentVolumeへのアクセス要求です。

PVCは、特定のサイズまたはアクセスモードのストレージを要求するように設定できます。クラスタ管理者は、関連付けられているStorageClassを使用して、PersistentVolumeのサイズとアクセスモード（パフォーマンスやサービスレベルなど）以上を制御できます。

PVとPVCを作成したら、ポッドにボリュームをマウントできます。

マニフェストの例

PersistentVolumeサンプルマニフェスト

このサンプルマニフェストは、StorageClassに関連付けられた10Giの基本PVを示しています `basic-csi`。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
  labels:
    type: local
spec:
  storageClassName: basic-csi
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/my/host/path"
```

PersistentVolumeClaimサンプルマニフェスト

次に、基本的なPVC設定オプションの例を示します。

RWOアクセスを備えたPVC

この例は、という名前のStorageClassに関連付けられたRWOアクセスを持つ基本的なPVCを示しています `basic-csi`。

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

NVMe / TCP対応PVC

この例は、という名前のStorageClassに関連付けられたNVMe/TCPの基本的なPVCとRWOアクセスを示しています `protection-gold`。

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

PODマニフェストのサンプル

次の例は、PVCをポッドに接続するための基本的な設定を示しています。

キホンセツテイ

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: pv-storage
      persistentVolumeClaim:
       claimName: basic
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: pv-storage
```

NVMe/TCPの基本構成

```
---
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
    - image: nginx
      name: nginx
      resources: {}
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
      claimName: pvc-san-nvme
```

PVおよび**PVC**の作成

手順

1. PVを作成

```
kubectl create -f pv.yaml
```

2. PVステータスを確認します。

```
kubectl get pv
NAME        CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM
STORAGECLASS  REASON  AGE
pv-storage  4Gi       RWO           Retain          Available
7s
```

3. PVCを作成

```
kubectl create -f pvc.yaml
```

4. PVCステータスを確認します。

```
kubectl get pvc
NAME          STATUS VOLUME     CAPACITY ACCESS MODES STORAGECLASS AGE
pvc-storage Bound  pv-name 2Gi       RWO                          5m
```

5. ボリュームをポッドにマウントします。

```
kubectl create -f pv-pod.yaml
```

> 進捗状況はを使用して監視でき `kubectl get pod --watch`ます。

6. ボリュームがにマウントされていることを確認します `/my/mount/path`。

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

7. ポッドを削除できるようになりました。Podアプリケーションは存在しなくなりますが、ボリュームは残ります。

```
kubectl delete pod task-pv-pod
```

ストレージクラスとパラメータおよびパラメータとの連携によるAstra Tridentによるボリュームのプロビジョニング方法の詳細については `PersistentVolumeClaim`、を参照して"Kubernetes オブジェクトと Trident オブジェクト"ください。

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-economy` `solidfire-san`ドライバでサポートされ `ontap-san`ており、Kubernetes 1.16以降が必要です。

手順 **1** ：ボリュームの拡張をサポートするようにストレージクラスを設定する

StorageClass定義を編集して、フィールドをに `true`設定し `allowVolumeExpansion`ます。

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

既存のStorageClassの場合は、パラメータを含めるように編集します `allowVolumeExpansion`。

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

PVC定義を編集し、を更新して、 `spec.resources.requests.storage`新しく希望するサイズ（元のサイズよりも大きくなければなりません）を反映させます。

```
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

サイズを変更するポッドにPVを接続します。iSCSI PV のサイズ変更には、次の 2 つのシナリオがあります。

- PV がポッドに接続されている場合、 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。

この例では、を使用するポッドが作成されて `san-pvc`います。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

ステップ**4**：**PV**を拡張する

作成されたPVのサイズを1Giから2Giに変更するには、PVC定義を編集してを2Giに更新します `spec.resources.requests.storage`。

```
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では、、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、、 `gcp-cvs` `azure-netapp-files`バックエンドでプロビジョニングされるNFS PVのボリューム拡張がサポートされます `ontap-nas`。

手順 **1** ：ボリュームの拡張をサポートするようにストレージクラスを設定する

NFS PVのサイズを変更するには、管理者はまず、フィールドをに `true`設定してボリュームの拡張を許可するようにストレージクラスを設定する必要があります。 `allowVolumeExpansion`

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

このオプションを指定せずにストレージクラスを作成済みの場合は、を使用して既存のストレージクラスを編集するだけでボリュームを拡張できます `kubectl edit storageclass`。

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

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

Astra Trident が、この PVC に対して 20MiB の NFS PV を作成する必要があります。

```
kubectl get pvc
NAME           STATUS   VOLUME
CAPACITY     ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi
RWO            ontapnas        9s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi       RWO
Delete           Bound    default/ontapnas20mb   ontapnas
2m42s
```

ステップ**3**：**PV**を拡張する

新しく作成した20MiB PVのサイズを1GiBに変更するには、PVCを編集して1GiBに設定し `spec.resources.requests.storage`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

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 `tridentctl import`。

概要と考慮事項

Astra Tridentにボリュームをインポートすると、次のことが可能になります。

- アプリケーションをコンテナ化し、既存のデータセットを再利用する
- 一時的なアプリケーションにはデータセットのクローンを使用
- 障害が発生したKubernetesクラスタを再構築します
- ディザスタリカバリ時にアプリケーションデータを移行

考慮事項

ボリュームをインポートする前に、次の考慮事項を確認してください。

- Astra Tridentでインポートできるのは*RW*（読み取り/書き込み）タイプのONTAPボリュームのみです。*DP*（データ保護）タイプのボリュームはSnapMirrorデスティネーションボリュームです。ボリュームをAstra

Tridentにインポートする前に、ミラー関係を解除する必要があります。

- アクティブな接続がないボリュームをインポートすることを推奨します。アクティブに使用されているボリュームをインポートするには、ボリュームのクローンを作成してからインポートを実行します。

Kubernetesは以前の接続を認識せず、アクティブなボリュームをポッドに簡単に接続できるため、これはブロックボリュームで特に重要です。その結果、データが破損する可能性があります。

- PVCには指定する必要がありますが `StorageClass`、Astra Tridentはインポート時にこのパラメータを使用しません。ストレージクラスは、ボリュームの作成時に、ストレージ特性に基づいて使用可能なプールから選択するために使用されます。ボリュームはすでに存在するため、インポート時にプールを選択する必要はありません。そのため、PVCで指定されたストレージクラスと一致しないバックエンドまたはプールにボリュームが存在してもインポートは失敗しません。
- 既存のボリュームサイズはPVCで決定され、設定されます。ストレージドライバによってボリュームがインポートされると、 PV は ClaimRef を使用して PVC に作成されます。
  - 再利用ポリシーは、PVでは最初にに設定されてい `retain`ます。Kubernetes が PVC と PV を正常にバインドすると、再利用ポリシーがストレージクラスの再利用ポリシーに合わせて更新されます。
  - ストレージクラスの再利用ポリシーがの場合、 `delete`PVが削除されるとストレージボリュームが削除されます。
- デフォルトでは、Astra TridentがPVCを管理し、バックエンドでFlexVolとLUNの名前を変更します。フラグを渡して管理対象外のボリュームをインポートできます `--no-manage`。を使用している場合、 `--no-manage`Astra Tridentはオブジェクトのライフサイクル全体にわたってPVCまたはPVに対して追加の処理を実行しません。PVが削除されてもストレージボリュームは削除されず、ボリュームのクローンやボリュームのサイズ変更などのその他の処理も無視されます。

このオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、Kubernetes 以外でストレージボリュームのライフサイクルを管理する場合に便利です。

- PVC と PV にアノテーションが追加されます。このアノテーションは、ボリュームがインポートされたこと、および PVC と PV が管理されていることを示す二重の目的を果たします。このアノテーションは変更または削除しないでください。

ボリュームをインポートします

を使用してボリュームをインポートできます `tridentctl import`。

手順

1. PVCの作成に使用するPersistent Volume Claim（PVC；永続的ボリューム要求）ファイル（など）を作成します `pvc.yaml`。PVCファイルには、、 `namespace`、 `accessModes`および `storageClassName`が含まれている必要があります `name`。必要に応じて、PVC定義でを指定できます `unixPermissions`。

   最小仕様の例を次に示します。

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

PV名やボリュームサイズなどの追加のパラメータは指定しないでください。これにより原因、インポートコマンドが失敗する可能性があります。

2. コマンドを使用し `tridentctl import`て、ボリュームを含むAstra Tridentバックエンドの名前と、ストレージ上のボリュームを一意に識別する名前（ONTAP FlexVol、Element Volume、Cloud Volumes Serviceパスなど）を指定します。 `-f`PVCファイルへのパスを指定するには、引数が必要です。

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

例

サポートされているドライバについて、次のボリュームインポートの例を確認してください。

ONTAP NASおよびONTAP NAS FlexGroup

Astra Tridentでは、ドライバと `ontap-nas-flexgroup`ドライバを使用したボリュームインポートがサポートされます `ontap-nas`。

- `ontap-nas-economy`ドライバはqtreeをインポートおよび管理できません。
- `ontap-nas`ドライバと `ontap-nas-flexgroup`ドライバでは、ボリューム名の重複は許可されていません。

ドライバを使用して作成される各ボリューム `ontap-nas`は、ONTAPクラスタ上のFlexVolになります。ドライバを使用したFlexVolのインポート `ontap-nas`も同様です。ONTAPクラスタにすでに存在するFlexVolは、PVCとしてインポートできます `ontap-nas`。同様に、FlexGroupボリュームはPVCとしてインポートできます `ontap-nas-flexgroup`。

ONTAP NASの例

次の例は、管理対象ボリュームと管理対象外ボリュームのインポートを示しています。

管理対象ボリューム

次の例は、という名前のバックエンドにある `ontap_nas`という名前のボリュームをインポートし `managed_volume`ます。

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

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

管理対象外のボリューム

引数を使用した場合 `--no-manage`、Astra Tridentはボリュームの名前を変更しません。

次に、バックエンドで `ontap_nas`をインポートする例を示し `unmanaged_volume`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   |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+---------+---------+
```

ONTAP SAN

Astra Tridentでは、ドライバを使用したボリュームインポートがサポートされます `ontap-san`。ドライバを使用したボリュームのインポートはサポートされていませ `ontap-san-economy`ん。

Astra Tridentでは、単一のLUNを含むONTAP SAN FlexVolをインポートできます。これは、ドライバと一致してい `ontap-san`ます。ドライバは、PVCごとにFlexVolを作成し、FlexVol内にLUNを作成します。Astra TridentがFlexVolをインポートし、PVCの定義に関連付けます。

ONTAP SANの例

次の例は、管理対象ボリュームと管理対象外ボリュームのインポートを示しています。

管理対象ボリューム

管理対象ボリュームの場合、Astra TridentはFlexVolの名前をの形式に、FlexVol内のLUNの名前をに`lun0`変更`pvc-<uuid>`します。

次に、バックエンドにあるFlexVol `ontap_san_default`をインポートする例を示し`ontap-san-managed`ます。

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

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

管理対象外のボリューム

次に、バックエンドで`ontap_san`をインポートする例を示し`unmanaged_example_volume`ます。

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

次の例に示すように、KubernetesノードのIQNとIQNを共有するigroupにLUNをマッピングすると、というエラーが表示されます。 `LUN already mapped to initiator(s) in this group`ボリュームをインポートするには、イニシエータを削除するか、LUNのマッピングを解除する必要があります。

```
Vserver   Igroup       Protocol OS Type  Initiators
--------- ------------ -------- -------- ------------------------------------
svm0      k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3
                       iscsi    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
svm0      unmanaged-example-igroup
                       mixed    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
```

要素

Astra Tridentでは、NetApp Elementソフトウェアとドライバを使用したNetApp HCIボリュームのインポートがサポートされます `solidfire-san`。

> Element ドライバではボリューム名の重複がサポートされます。ただし、ボリューム名が重複している場合はAstra Tridentからエラーが返されます。回避策としてボリュームをクローニングし、一意のボリューム名を指定して、クローンボリュームをインポートします。

要素の例

次の例は、バックエンドにボリュームを `element_default`インポートし `element-managed`ます。

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

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

Google Cloud Platform

Astra Tridentでは、ドライバを使用したボリュームインポートがサポートされます `gcp-cvs`。

> NetApp Cloud Volumes Serviceから作成されたボリュームをGoogle Cloud Platformにインポートするには、ボリュームパスでボリュームを特定します。ボリュームパスは、ボリュームのエクスポートパスののに続く部分です `:/`。たとえば、エクスポートパスがの場合、`10.0.0.1:/adroit-jolly-swift`ボリュームパスはになり `adroit-jolly-swift`ます。

Google Cloud Platformの例

次の例は、ボリュームパスがの `adroit-jolly-swift`バックエンドにボリュームを `gcpcvs_YEppr`インポートし `gcp-cvs`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    |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files

Astra Tridentでは、ドライバを使用したボリュームインポートがサポートされます `azure-netapp-files`。

> Azure NetApp Filesボリュームをインポートするには、ボリュームパスでボリュームを特定します。ボリュームパスは、ボリュームのエクスポートパスののに続く部分です `:/`。たとえば、マウントパスがの場合、 `10.0.0.2:/importvol1`ボリュームパスはになり `importvol1`ます。

Azure NetApp Filesの例

次の例は、ボリュームパスを持つ `importvol1`バックエンドのボリューム `azurenetappfiles_40517`をインポートし `azure-netapp-files`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    |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
```

ボリュームの名前とラベルをカスタマイズする

Astra Tridentでは、作成したボリュームにわかりやすい名前とラベルを割り当てることができます。これにより、ボリュームを特定し、それぞれのKubernetesリソース（PVC）に簡単にマッピングできます。また、バックエンドレベルでテンプレートを定義してカスタムボリューム名とカスタムラベルを作成することもできます。作成、インポート、またはクローンを作成するボリュームは、テンプレートに準拠します。

開始する前に

カスタマイズ可能なボリューム名とラベルのサポート：

1. ボリュームの作成、インポート、クローニングの各処理。
2. ontap-nas-economyドライバの場合、qtreeボリュームの名前だけがテンプレート名に準拠します。
3. ontap-san-economyドライバの場合、名前テンプレートに準拠するのはLUN名のみです。

制限事項

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

カスタマイズ可能なボリューム名の主な動作

1. 名前テンプレートの無効な構文が原因でエラーが発生した場合、バックエンドの作成は失敗します。ただし、テンプレートアプリケーションが失敗した場合は、既存の命名規則に従ってボリュームに名前が付けられます。
2. バックエンド構成の名前テンプレートを使用してボリュームの名前が指定されている場合、ストレージプレフィックスは適用されません。任意のプレフィックス値をテンプレートに直接追加できます。

名前テンプレートとラベルを使用したバックエンド構成の例

カスタム名テンプレートは、ルートレベルまたはプールレベルで定義できます。

ルートレベルの例

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

プールレベルの例

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

名前テンプレートの例

*例1 *：

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

*例2 *：

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

考慮すべきポイント

1. ボリュームインポートの場合、既存のボリュームに特定の形式のラベルがある場合にのみラベルが更新されます。例： `{"provisioning":{"Cluster":"ClusterA", "PVC": "pvcname"}}`。
2. 管理対象ボリュームのインポートの場合、ボリューム名はバックエンド定義のルートレベルで定義された名前テンプレートの後に続きます。
3. Astra Tridentでは、ストレージプレフィックスを含むスライス演算子の使用はサポートされていません。
4. テンプレートで一意のボリューム名が生成されない場合、Astra Tridentではいくつかのランダムな文字が追加されて一意のボリューム名が作成されます。
5. NASエコノミーボリュームのカスタム名が64文字を超えると、Astra Tridentは既存の命名規則に従ってボリュームに名前を付けます。他のすべてのONTAPドライバでは、ボリューム名が名前の上限を超えると、ボリュームの作成プロセスが失敗します。

ネームスペース間で**NFS**ボリュームを共有します

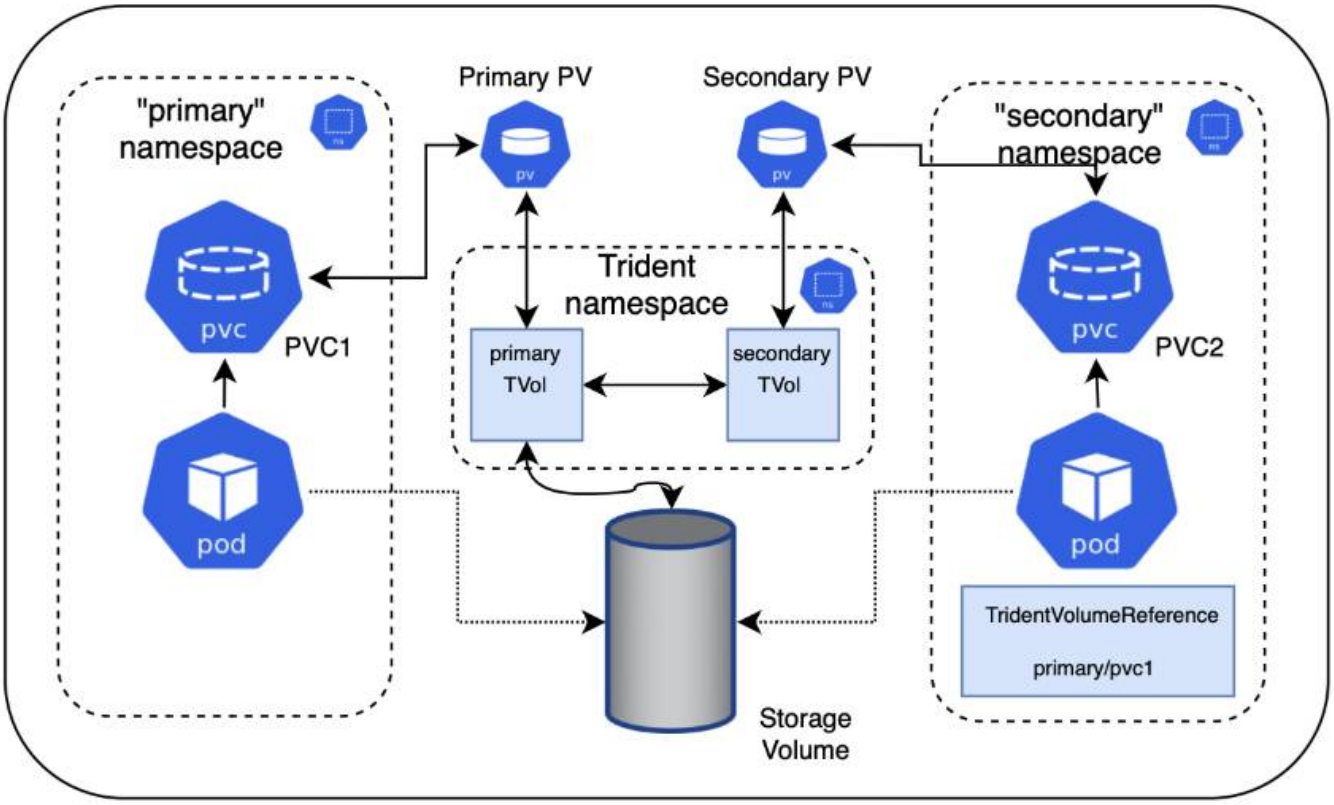
Tridentを使用すると、プライマリネームスペースにボリュームを作成し、1つ以上のセカンダリネームスペースで共有できます。

特徴

Astra TridentVolumeReference CRを使用すると、1つ以上のKubernetesネームスペース間でReadWriteMany（RWX）NFSボリュームをセキュアに共有できます。このKubernetesネイティブ解決策 には、次のようなメリットがあります。

- セキュリティを確保するために、複数のレベルのアクセス制御が可能です
- すべてのTrident NFSボリュームドライバで動作
- tridentctlやその他の非ネイティブのKubernetes機能に依存しません

この図は、2つのKubernetesネームスペース間でのNFSボリュームの共有を示しています。



クイックスタート

NFSボリューム共有はいくつかの手順で設定できます。

1 ボリュームを共有するように送信元**PVC**を設定する

ソースネームスペースの所有者は、ソースPVCのデータにアクセスする権限を付与します。

2 宛先名前空間に**CR**を作成する権限を付与する

クラスタ管理者が、デスティネーションネームスペースの所有者にTridentVolumeReference CRを作成する権限を付与します。

3 デスティネーションネームスペースに**TridentVolumeReference**を作成

宛先名前空間の所有者は、送信元PVCを参照するためにTridentVolumeReference CRを作成します。

4 宛先ネームスペースに下位**PVC**を作成します。

宛先名前空間の所有者は、送信元PVCからのデータソースを使用する下位PVCを作成します。

ソースネームスペースとデスティネーションネームスペースを設定します

セキュリティを確保するために、ネームスペース間共有では、ソースネームスペースの所有者、クラスタ管理

者、および宛先ネームスペースの所有者によるコラボレーションとアクションが必要です。ユーザロールは各手順で指定します。

手順

1. ソースネームスペースの所有者： pvc(`pvc1`を作成します）（`namespace2。注釈を使用して、デスティネーションネームスペースとの共有権限を付与します。 `shareToNamespace`

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

Astra TridentがPVとバックエンドのNFSストレージボリュームを作成

- カンマ区切りリストを使用して、複数の名前空間にPVCを共有できます。たとえば、`trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`です。
- を使用して、すべてのネームスペースと共有できます `*`。例えば、`trident.netapp.io/shareToNamespace: *`
- PVCはいつでも更新してアノテーションを含めることができます `shareToNamespace`。

2. *クラスタ管理者：*カスタムロールとkubeconfigを作成して、デスティネーションネームスペースの所有者にTridentVolumeReference CRを作成する権限を付与します。
3. *デスティネーションネームスペースの所有者：*ソースネームスペースを参照するTridentVolumeReference CRをデスティネーションネームスペースに作成します `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 宛先ネームスペース所有者：(pvc2`宛先ネームスペースにPVCを作成(`namespace2）。注釈を使用して送信元PVCを指定します。 shareFromPVC

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

宛先PVCのサイズは、送信元PVCのサイズ以下である必要があります。

結果

Astra TridentはデスティネーションPVCのアノテーションを読み取り shareFromPVC、ソースPVストレージリソースを共有する独自のストレージリソースのない下位ボリュームとしてデスティネーションPVを作成します。宛先PVCとPVは、通常どおりバインドされているように見えます。

共有ボリュームを削除

複数のネームスペースで共有されているボリュームは削除できます。Tridentが、ソースネームスペースのボリュームへのアクセスを削除し、ボリュームを共有する他のネームスペースへのアクセスを維持します。ボリュームを参照するすべてのネームスペースが削除されると、Astra Tridentによってボリュームが削除されます。

下位ボリュームのクエリに使用 tridentctl get

ユーティリティを使用する[tridentctl`と、コマンドを実行して従属ボリュームを取得できます `get。詳細については、リンク:../ Trident -reference/tridentctl.htmlコマンドとオプション]を参照して[`tridentctl`ください。

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

フラグ：

- `-h, --help：ボリュームのヘルプ。
- --parentOfSubordinate string：クエリを下位のソースボリュームに制限します。

- `--subordinateOf string`:クエリをボリュームの下位に限定します。

制限事項

- Astra Tridentでは、デスティネーションネームスペースが共有ボリュームに書き込まれるのを防ぐことはできません。共有ボリュームのデータの上書きを防止するには、ファイルロックなどのプロセスを使用する必要があります。
- または `shareFromNamespace`注釈を削除したり、CRを削除したりし `TridentVolumeReference`て、送信元PVCへのアクセスを取り消すことはできません `shareToNamespace`。アクセスを取り消すには、下位PVCを削除する必要があります。
- Snapshot、クローン、およびミラーリングは下位のボリュームでは実行できません。

詳細情報

ネームスペース間のボリュームアクセスの詳細については、次の資料を参照してください。

- にアクセスします"ネームスペース間でのボリュームの共有：ネームスペース間のボリュームアクセスを許可する場合は「Hello」と入力します"。
- のデモをご覧ください "ネットアップTV"。

CSI トポロジを使用します

Astra Tridentでは、を使用して、Kubernetesクラスタ内のノードを選択的に作成して接続できます "CSI トポロジ機能"。

概要

CSI トポロジ機能を使用すると、領域およびアベイラビリティゾーンに基づいて、ボリュームへのアクセスをノードのサブセットに制限できます。現在、クラウドプロバイダは、 Kubernetes 管理者がゾーンベースのノードを生成できるようになっています。ノードは、リージョンによって異なるアベイラビリティゾーンに配置することも、リージョンによって配置することもできます。マルチゾーンアーキテクチャでワークロード用のボリューム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、こちらを参照して "ここをクリック"ください。

Kubernetes には、 2 つの固有のボリュームバインドモードがあります。

- `VolumeBindingMode`をに設定する `Immediate`と、Astra Tridentはトポロジを認識せずにボリュームを作成します。ボリュームバインディングと動的プロビジョニングは、 PVC が作成されるときに処理されます。これはデフォルト `VolumeBindingMode`であり、トポロジの制約を適用しないクラスタに適しています。永続ボリュームは、要求元ポッドのスケジュール要件に依存することなく作成されます。
- `VolumeBindingMode`をに設定する `WaitForFirstConsumer`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。

- を実行するKubernetesクラスタ"サポートされるKubernetesバージョン"

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

- クラスタ内のノードには、トポロジ対応と `topology.kubernetes.io/zone`を示すラベルを付ける必要があり(`topology.kubernetes.io/region`ます。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、サポートされているゾーンとリージョンのリストを表すオプションのブロックを運ぶことができます `supportedTopologies` 。ストレージクラスがそのようなバックエンドを使用する場合、ボリュームは、サポートされているリージョン / ゾーンでスケジュールされているアプリケーションから要求された場合にのみ作成されます。

バックエンド定義の例を次に示します。

YAML

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

JSON

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

`supportedTopologies`は、バックエンドごと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。

ストレージプールごとにも定義できます `supportedTopologies`。次の例を参照してください。

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-backend-us-central1
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
- topology.kubernetes.io/region: us-central1
  topology.kubernetes.io/zone: us-central1-a
- topology.kubernetes.io/region: us-central1
  topology.kubernetes.io/zone: us-central1-b
storage:
- labels:
    workload: production
  supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-a
- labels:
    workload: dev
  supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-b
```

この例では `region`、ラベルと `zone`ラベルはストレージプールの場所を表しています。 `topology.kubernetes.io/region` `topology.kubernetes.io/zone`ストレージプールの消費元を指定します。

手順 **2** ：トポロジを認識するストレージクラスを定義する

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

次の例を参照してください。

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

前述のStorageClass定義では、 `volumeBindingMode`がに設定されて `WaitForFirstConsumer`います。この `StorageClass` で要求された `PVC` は、ポッドで参照されるまで処理されません。およびに、 `allowedTopologies`使用するゾーンとリージョンを示します。StorageClassは `netapp-san-us-east1`、上記で定義したバックエンドにPVCを作成します `san-backend-us-east1`。

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。

次の例を参照して `spec`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、リージョンに存在するノードでポッドをスケジュールし、ゾーンまたは `us-east1-b`ゾーンに存在する任意のノードから選択する `us-east1-a`ようにKubernetesに指示し `us-east1`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含める `supportedTopologies`

既存のバックエンドを更新して、使用の `tridentctl backend update`リストを含めることができます `supportedTopologies`。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。

詳細情報

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

スナップショットを操作します

永続ボリューム（PV）のKubernetesボリュームSnapshotを使用すると、ボリュームのポイントインタイムコピーを作成できます。Astra Tridentを使用して作成したボリュームのSnapshotの作成、Astra Trident外で作成したSnapshotのインポート、既存のSnapshotから新しいボリュームの作成、Snapshotからボリュームデータをリカバリできます。

概要

ボリュームSnapshotは、、 `ontap-nas-flexgroup` `ontap-san`、 `ontap-san-economy` `solidfire-san`、、、でサポートされます `ontap-nas`。 `gcp-cvs`、および `azure-netapp-files`ドライバ。

開始する前に

スナップショットを操作するには、外部スナップショットコントローラとカスタムリソース定義（CRD）が必要です。Kubernetesオーケストレーションツール（例：Kubeadm、GKE、OpenShift）の役割を担っています。

Kubernetesディストリビューションにスナップショットコントローラと CRDが含まれていない場合は、を参照してくださいボリュームSnapshotコントローラの導入。

GKE環境でオンデマンドボリュームスナップショットを作成する場合は、スナップショットコントローラを作成しないでください。GKEでは、内蔵の非表示のスナップショットコントローラを使用します。

ボリューム **Snapshot** を作成します

手順

1. を作成し `VolumeSnapshotClass`ます。詳細については、を参照してください"ボリュームSnapshotクラス"。
   - はAstra Trident CSIドライバを示してい `driver`ます。
   - `deletionPolicy`には、または `Retain`を指定できます `Delete`。に設定する `Retain`と、オブジェクトが削除されても、ストレージクラスタの基盤となる物理Snapshotが保持され `VolumeSnapshot`ます。

     例

     ```
     cat snap-sc.yaml
     apiVersion: snapshot.storage.k8s.io/v1
     kind: VolumeSnapshotClass
     metadata:
       name: csi-snapclass
     driver: csi.trident.netapp.io
     deletionPolicy: Delete
     ```

2. 既存のPVCのスナップショットを作成します。

   例

   - 次に、既存のPVCのスナップショットを作成する例を示します。

     ```
     cat snap.yaml
     apiVersion: snapshot.storage.k8s.io/v1
     kind: VolumeSnapshot
     metadata:
       name: pvc1-snap
     spec:
       volumeSnapshotClassName: csi-snapclass
       source:
         persistentVolumeClaimName: pvc1
     ```

   - この例では、というPVCのボリュームSnapshotオブジェクトを作成し `pvc1`、Snapshotの名前をに設定して `pvc1-snap`います。VolumeSnapshotはPVCに似ており、実際のSnapshotを表すオブジェクトに関連付けられて `VolumeSnapshotContent`います。

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

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

- VolumeSnapshotのオブジェクト `pvc1-snap`を説明することで特定できます `VolumeSnapshotContent`。は `Snapshot Content Name`、このSnapshotを提供するVolumeSnapshotContentオブジェクトを識別します。パラメータは、 `Ready To Use`スナップショットを使用して新しいPVCを作成できることを示します。

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

ボリューム**Snapshot**から**PVC**を作成

を使用して、という名前のVolumeSnapshotをデータのソースとして使用してPVCを作成 `<pvc-name>`できます `dataSource`。作成された PVC は、ポッドに接続して、他の PVC と同様に使用できます。

> PVCはソースボリュームと同じバックエンドに作成されます。を参照してください "KB：Trident PVCスナップショットからPVCを作成することは代替バックエンドではできない"。

次に、をデータソースとして使用してPVCを作成する例を示し `pvc1-snap`ます。

```
cat pvc-from-snap.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

ボリューム**Snapshot**のインポート

Astra Tridentでは、がサポートされます"Kubernetesの事前プロビジョニングされたSnapshotプロセス"。クラスタ管理者は、Astra Tridentの外部で作成されたオブジェクトの作成やSnapshotのインポートを行うことができ `VolumeSnapshotContent`ます。

開始する前に

Astra TridentでSnapshotの親ボリュームが作成またはインポートされている必要があります。

手順

1. *クラスタ管理者：*バックエンドSnapshotを参照するオブジェクトを作成します `VolumeSnapshotContent`。これにより、Astra TridentでSnapshotワークフローが開始されます。
   - にバックエンドスナップショットの名前を `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`指定します `annotations`。
   - で指定し `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name> snapshotHandle`ます。この情報は、呼び出しで外部`Snapshot`者から`Astra Trident`に提供される唯一の情報です `ListSnapshots`。

     CRの名前の制約により、は `<volumeSnapshotContentName>`バックエンドスナップショット名と常に一致しません。

     例

     次の例では、バックエンドスナップショットを参照するオブジェクトを `snap-01`作成し `VolumeSnapshotContent`ます。

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

2. *クラスタ管理者：*オブジェクトを参照するCR `VolumeSnapshotContent`を作成します `VolumeSnapshot`。これにより、指定された名前空間でを使用するためのアクセスが要求され`VolumeSnapshot`ます。

   例

   次の例では、という名前 `import-snap-content`を参照する `VolumeSnapshotContent`という名前のCRを `import-snap`作成します `VolumeSnapshot`。

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

3. *内部処理（アクション不要）：*外部スナップショットは、新しく作成されたを認識して `VolumeSnapshotContent`呼び出しを実行します `ListSnapshots`。Astra Tridentがを作成 `TridentSnapshot`
   - 外部スナップショットは、をに `readyToUse`設定し、 `VolumeSnapshot`をに `true`設定し `VolumeSnapshotContent`ます。
   - Tridentが戻ります `readyToUse=true`。
4. *任意のユーザー：*を作成し `PersistentVolumeClaim`て、新しいを参照します `VolumeSnapshot`。 `spec.dataSource`（または `spec.dataSourceRef`）の名前は名前です `VolumeSnapshot`。

   例

次に、という名前の `import-snap`を参照するPVCを作成する例を示し `VolumeSnapshot`ます。

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

Snapshotを使用したボリュームデータのリカバリ

デフォルトでは、ドライバと `ontap-nas-economy`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ため、snapshotディレクトリは非表示になってい `ontap-nas`ます。ディレクトリがスナップショットからデータを直接リカバリできるようにし `.snapshot`ます。

ボリュームを以前のSnapshotに記録されている状態にリストアするには、ボリュームSnapshotリストアONTAP CLIを使用します。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

Snapshotコピーをリストアすると、既存のボリューム設定が上書きされます。Snapshotコピーの作成後にボリュームデータに加えた変更は失われます。

Snapshotが関連付けられている**PV**を削除する

スナップショットが関連付けられている永続ボリュームを削除すると、対応する Trident ボリュームが「削除状態」に更新されます。ボリュームSnapshotを削除してAstra Tridentボリュームを削除します。

ボリューム**Snapshot**コントローラの導入

Kubernetesディストリビューションにスナップショットコントローラと CRDが含まれていない場合は、次のように導入できます。

手順

1. ボリュームのSnapshot作成

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

2. スナップショットコントローラを作成します。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

必要に応じて、名前空間を開い `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`て更新し `namespace`ます。

関連リンク

- "ボリューム Snapshot"
- "ボリュームSnapshotクラス"

Astra Tridentの管理と監視

Astra Trident をアップグレード

Astra Trident をアップグレード

24.02リリース以降、Astra Tridentのリリースサイクルは4カ月になり、メジャーリリースは暦年に3回提供されます。新しいリリースは、以前のリリースに基づいて構築され、新機能、パフォーマンスの強化、バグの修正、および改善が提供されます。ネットアップでは、Astra Tridentの新機能を活用するために、1年に1回以上アップグレードすることを推奨しています。

アップグレード前の考慮事項

最新リリースの Astra Trident にアップグレードする際は、次の点を考慮してください。

- 特定のKubernetesクラスタ内のすべてのネームスペースには、Astra Tridentインスタンスを1つだけインストールする必要があります。
- Astra Trident 23.07以降では、v1ボリュームSnapshotが必要です。アルファSnapshotまたはベータSnapshotはサポートされなくなりました。
- でCloud Volumes Service for Google Cloudを作成している場合"CVS サービスタイプ"は、Astra Trident 23.01からのアップグレード時にまたは `zoneredundantstandardsw`サービスレベルを使用するようにバックエンド構成を更新する必要があります `standardsw`。バックエンドでを更新しない `serviceLevel`と、ボリュームで障害が発生する可能性があります。詳細については、を参照してください"CVSサービスタイプのサンプル" 。
- アップグレードする際は、を `StorageClasses`提供して`Astra Trident`で使用することが重要です `parameter.fsType`。既存のボリュームを停止することなく、削除や再作成を実行できます `StorageClasses`。
  - これは、SANボリュームを適用するための要件です "セキュリティコンテキスト"。
  - https://github.com/ NetApp / Trident /tree/master/storage-installer/sample-input Trident [sample input^] ディレクトリには、https://github.com/ NetApp / Trident /blob/master/storage-class-samples/storage-class-basic.yaml.templ^] Tridentやlink：https://github.com/ NetApp / Trident /blob/master/sample-input/storage-class-sample-input/storage-input/storage-class-samples/storage[`storage-class-basic.yaml.templ` Trident[`storage-class-bronze-default.yaml`
  - 詳細については、を参照してください "既知の問題"。

ステップ**1**：バージョンを選択します

Astra Tridentのバージョンには、日付ベースの命名規則が適用され `YY.MM`ます。「YY」は年の最後の2桁、「MM」は月です。ドットリリースは規則に従い `YY.MM.X`ます。「X」はパッチレベルです。アップグレード前のバージョンに基づいて、アップグレード後のバージョンを選択します。

- インストールされているバージョンの4リリースウィンドウ内にある任意のターゲットリリースに直接アップグレードできます。たとえば、23.04（または任意の23.04 DOTリリース）から24.06に直接アップグレードできます。
- 4つのリリースウィンドウ以外のリリースからアップグレードする場合は、複数の手順でアップグレード

を実行します。4リリースのウィンドウに適合する最新リリースにアップグレードするには、アップグレード元ののののアップグレード手順を使用し "以前のバージョン" ます。たとえば、22.01を実行していて、24.06にアップグレードする場合は、次の手順を実行します。

a. 22.07から23.04への最初のアップグレード。

b. その後、23.04から24.06にアップグレードします。

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

ステップ**2:**元のインストール方法を決定します

Astra Tridentの最初のインストールに使用したバージョンを確認するには、次の手順を実行します。

1. ポッドの検査に使用し `kubectl get pods -n trident`ます。
   - オペレータポッドがない場合は、を使用してAstra Tridentをインストールしました `tridentctl`。
   - オペレータポッドがある場合、Astra Tridentは手動またはHelmを使用してインストールされています。
2. オペレータポッドがある場合は、を使用 `kubectl describe torc`してAstra TridentがHelmを使用してインストールされているかどうかを確認します。
   - Helmラベルがある場合は、Helmを使用してAstra Tridentがインストールされています。
   - Helmラベルがない場合は、Astra TridentをTridentオペレータを使用して手動でインストールしています。

ステップ**3**：アップグレード方法を選択します

通常は、最初のインストールと同じ方法でアップグレードする必要がありますが、可能です。"インストール方法を切り替えます"Tridentをアップグレードする方法は2つあります。

- "Tridentオペレータを使用してアップグレード"

> オペレータとアップグレードする前に、を確認することをお勧めします"オペレータのアップグレードワークフローについて理解する"。

*

オペレータにアップグレードしてください

オペレータのアップグレードワークフローについて理解する

Tridentオペレータを使用してAstra Tridentをアップグレードする前に、アップグレード中に発生するバックグラウンドプロセスを理解しておく必要があります。これには、Tridentコントローラ、コントローラポッドとノードポッド、およびローリング更新を可能にするノードデーモンセットに対する変更が含まれます。

Tridentオペレータのアップグレード処理

Astra Tridentのインストールとアップグレードに必要なものの1つ"Tridentオペレータを使用するメリット"は、既存のマウントボリュームを停止することなく、Astra TridentとKubernetesのオブジェクトを自動的に処理することです。このようにして、Astra Tridentはダウンタイムなしでアップグレードをサポートできます。"ローリング更新"TridentオペレータはKubernetesクラスタと通信して次のことを行います。

- Trident Controller環境とノードデーモンセットを削除して再作成します。
- TridentコントローラポッドとTridentノードポッドを新しいバージョンに置き換えます。
  - 更新されていないノードは、残りのノードの更新を妨げません。
  - ボリュームをマウントできるのは、Trident Node Podを実行しているノードだけです。

KubernetesクラスタのAstra Tridentアーキテクチャの詳細については、を参照してください"Astra Tridentのアーキテクチャ"。

オペレータのアップグレードワークフロー

Tridentオペレータを使用してアップグレードを開始すると、次の処理が実行されます。

1. Trident演算子*：
   a. 現在インストールされているAstra Tridentのバージョン（version_n_）を検出します。
   b. CRD、RBAC、Trident SVCなど、すべてのKubernetesオブジェクトを更新
   c. version_n_用のTrident Controller環境を削除します。
   d. version_n+1_用のTrident Controller環境を作成します。
2. * Kubernetes *は、_n+1_用にTridentコントローラポッドを作成します。
3. Trident演算子*：
   a. _n_のTridentノードデーモンセットを削除します。オペレータは、Node Podが終了するのを待たない。
   b. _n+1_のTridentノードデーモンセットを作成します。
4. * Kubernetes * Trident Node Pod_n_を実行していないノードにTridentノードポッドを作成します。これにより、1つのノードに複数のTrident Node Pod（バージョンに関係なく）が存在することがなくなります。

Trident OperatorまたはHelmを使用してAstra Tridentのインストールをアップグレード

Astra Tridentは、Tridentオペレータを使用して手動またはHelmを使用してアップグレードできます。Tridentオペレータのインストールから別のTridentオペレータのインストールにアップグレードすることも、インストールからTridentオペレータのバージョンにアップグレードすることもできます `tridentctl`。Trident Operatorのインストールをアップグレードする前にを参照してください"アップグレード方法を選択します"。

手動インストールのアップグレード

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

ネームスペースを対象としたオペレータ（バージョン20.07~20.10）を使用してインストールされたAstra Tridentからアップグレードするには、Astra Tridentのアップグレード手順を使用します"インストールされているバージョン"。

タスクの内容

Tridentにはバンドルファイルが用意されています。このファイルを使用して、オペレータをインストールしたり、Kubernetesバージョンに対応する関連オブジェクトを作成したりできます。

- クラスタでKubernetes 1.24を実行している場合は、を使用し "Bundle_pre_1_25.yaml"ます。
- クラスタでKubernetes 1.25以降を実行している場合は、を使用します "bundle_post_1_25.yaml"。

開始する前に

を実行しているKubernetesクラスタを使用していることを確認し"サポートされるKubernetesバージョン"ます。

手順

1. Astra Tridentのバージョンを確認します。

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

2. 現在の Astra Trident インスタンスのインストールに使用した Trident オペレータを削除たとえば、23.07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 23.07.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

3. 属性を使用して初期インストールをカスタマイズした場合 `TridentOrchestrator`は、オブジェクトを編集してインストールパラメータを変更できます `TridentOrchestrator`。これには、ミラーリングされたTridentおよびCSIイメージレジストリをオフラインモードに指定したり、デバッグログを有効にしたり、イメージプルシークレットを指定したりするための変更が含まれます。
4. 環境に適したバンドルYAMLファイルを使用してAstra Tridentをインストールします（_YAMLは、または使用している<bundle.yaml>
`bundle_pre_1_25.yaml` `bundle_post_1_25.yaml` のバージョンに基づいています）。たとえば、Astra Trident 24.06をインストールする場合は、次のコマンドを実行します。

```
kubectl create -f 24.06.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

Helmインストールのアップグレード

Astra Trident Helmのインストールをアップグレードできます。

Astra Tridentがインストールされ`helm upgrade`ているKubernetesクラスタを1.24から1.25以降にアップグレードする場合は、クラスタをアップグレードする前に、値.yamlをに`true`設定するかコマンドに追加するよう`--set excludePodSecurityPolicy=true`に更新する必要があります。`excludePodSecurityPolicy`

手順

1. を使用する"Helmを使用したAstra Tridentのインストール"と、を使用してワンステップでアップグレードできます`helm upgrade trident netapp-trident/trident-operator --version 100.2406.0`。Helmリポジトリを追加しなかった場合、またはHelmリポジトリを使用してアップグレードできない場合は、次の手順を実行します。
   a. から最新のAstra Tridentリリースをダウンロードし"GitHubの_Assets_sectionを参照してください"ます。
   b. コマンドを使用し`helm upgrade`ます。は、`trident-operator-24.06.0.tgz`アップグレード先のバージョンを反映しています。

```
helm upgrade <name> trident-operator-24.06.0.tgz
```

初期インストール時にカスタムオプションを設定した場合（TridentおよびCSIイメージのプライベートなミラーレジストリの指定など）は、を使用してコマンドを追加し`helm upgrade`、これらのオプションがアップグレードコマンドに含まれていることを確認します。含まれていない場合は、`--set`値がデフォルトにリセットされます。

2. を実行し`helm list`て、チャートとアプリのバージョンの両方がアップグレードされたことを確認します。を実行し`tridentctl logs`てデバッグメッセージを確認します。

インストールから**Trident Operator**へのアップグレード`tridentctl`

インストールからTrident Operatorを最新リリースにアップグレードできます`tridentctl`。既存のバックエンドとPVCは自動的に使用可能になります。

インストール方法を切り替える前に、を参照してください"インストール方法を切り替える"。

手順

1. 最新の Astra Trident リリースをダウンロード

```
# Download the release required [24.060.0]
mkdir 24.06.0
cd 24.06.0
wget
https://github.com/NetApp/trident/releases/download/v24.06.0/trident-
installer-24.06.0.tar.gz
tar -xf trident-installer-24.06.0.tar.gz
cd trident-installer
```

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

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

3. クラスタを対象としたオペレータを同じネームスペースに導入します。

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

serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created

#Examine the pods in the Trident namespace
NAME                                READY   STATUS    RESTARTS   AGE
trident-controller-79df798bdc-m79dc   6/6     Running   0          150d
trident-node-linux-xrst8              2/2     Running   0          150d
trident-operator-5574dbbc68-nthjv     1/1     Running   0          1m30s
```

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

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

kubectl create -f deploy/crds/tridentorchestrator_cr.yaml

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

5. Tridentが目的のバージョンにアップグレードされたことを確認

```
kubectl describe torc trident | grep Message -A 3

Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v24.06.0
```

tridentctl を使用してアップグレードします

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 `tridentctl`。

タスクの内容

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。

開始する前に

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

手順

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

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

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

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

Tridentctlを使用したAstra Tridentの管理

には "Trident インストーラバンドル"、Astra Tridentに簡単にアクセスできるコマンドラインユーティリティが含まれてい `tridentctl`ます。十分な権限を持つKubernetesユーザは、この権限を使用してAstra Tridentをインストールしたり、Astra Tridentポッドを含むネームスペースを管理したりできます。

コマンドとグローバルフラグ

を実行して使用可能なコマンドのリストを取得 `tridentctl`したり、任意のコマンドにフラグを追加して特定のコマンドのオプションとフラグのリストを取得したり `--help`できます `tridentctl help。

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

Astra Trident `tridentctl`ユーティリティでは、次のコマンドとグローバルフラグがサポートされます。

コマンド

create
Astra Tridentにリソースを追加

delete
Astra Tridentから1つ以上のリソースを削除します。

get
Astra Tridentから1つ以上のリソースを入手します。

help
任意のコマンドに関するヘルプ。

images
Astra Tridentが必要とするコンテナイメージの表を出力します。

import
既存のリソースをAstra Tridentにインポート

install
Astra Trident をインストール

logs
Astra Tridentからログを出力

send
Astra Tridentからリソースを送信

uninstall
Astra Tridentをアンインストールします。

update
Astra Tridentでリソースを変更

update backend state
バックエンド処理を一時的に中断します。

upgrade
Astra Tridentでリソースをアップグレード

version
Astra Tridentのバージョンを出力します。

グローバルフラグ

-d、 --debug
デバッグ出力。

-h、 --help
のヘルプ `tridentctl`。

-k、 --kubeconfig string
コマンドをローカルまたはKubernetesクラスタ間で実行するパスを指定します `KUBECONFIG`。

または、変数をエクスポートして特定のKubernetesクラスタをポイントし、そのクラスタに対してコマンドを実行する `tridentctl`ことも できます `KUBECONFIG`。

-n、 --namespace string
Astra Trident導入のネームスペース。

-o、 --output string
出力形式。JSON の 1 つ | yaml | name | wide | ps （デフォルト）。

-s、 --server string
Astra Trident RESTインターフェイスのアドレス/ポート。

Trident REST インターフェイスは、 127.0.0.1 （ IPv4 の場合）または [::1] （ IPv6 の場合）のみをリスンして処理するように設定できます。

コマンドオプションとフラグ

作成

コマンドを使用し `create`て、Astra Tridentにリソースを追加します。

```
tridentctl create [option]
```

オプション
`backend`：Astra Tridentにバックエンドを追加します。

削除

コマンドを使用し `delete`て、Astra Tridentから1つ以上のリソースを削除します。

```
tridentctl delete [option]
```

オプション
`backend`：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。
`snapshot`：Astra Tridentから1つ以上のボリュームSnapshotを削除します。

`storageclass`：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。

取得

コマンドを使用し `get`て、Astra Tridentから1つ以上のリソースを取得します。

```
tridentctl get [option]
```

オプション

`backend`：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Snapshotを取得します。
`storageclass`：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。

フラグ

`-h、 --help`：ボリュームのヘルプ。
`--parentOfSubordinate string`：クエリを下位のソースボリュームに制限します。
`--subordinateOf string`:クエリをボリュームの下位に限定します。

イメージ

フラグを使用して、Astra Tridentが必要とするコンテナイメージの表を出力します `images`。

```
tridentctl images [flags]
```

フラグ

`-h、 --help`：画像のヘルプ。
`-v、 --k8s-version string`：Kubernetesクラスタのセマンティックバージョン。

ボリュームをインポートします

コマンドを使用し `import volume`て、既存のボリュームをAstra Tridentにインポートします。

```
tridentctl import volume <backendName> <volumeName> [flags]
```

エイリアス

`volume、 v`

フラグ

`-f,: --filename string` YAMLまたはJSON PVCファイルへのパス。
`-h、 --help`：ボリュームのヘルプ。
`--no-manage`：PV/PVCのみを作成します。ボリュームのライフサイクル管理を想定しないでください。

インストール

フラグを使用し `install`てAstra Tridentをインストール

```
tridentctl install [flags]
```

フラグ

`--autosupport-image string`：AutoSupportテレメトリのコンテナイメージ（デフォルトは「NetApp / Trident AutoSupport：<current-version>」）。
`--autosupport-proxy string`: AutoSupportテレメトリを送信するためのプロキシのアドレス/ポート。
`--enable-node-prep`：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`:何もインストールせずにYAMLファイルを生成します。
`-h、 --help`：インストールのヘルプ。
`--http-request-timeout`：TridentコントローラのREST APIのHTTP要求タイムアウトを上書きします（デフォルトは1m30秒）。
`--image-registry string`:内部イメージレジストリのアドレス/ポート。
`--k8s-timeout duration`：すべてのKubernetes処理のタイムアウト（デフォルトは3m0）。
`--kubelet-dir string`: kubeletの内部状態のホストの場所(デフォルトは/var/lib/kubelet)。
`--log-format string`：Astra Tridentのロギング形式（テキスト、JSON）（デフォルトは「text」）。
`--pv string`：Astra Tridentで使用する従来のPVの名前。makes sure this doesn't exist（デフォルトは「Trident」）
`--pvc string`：Astra Tridentで使用される従来のPVCの名前。makes sure this doesn't exist（デフォルトは「Trident」）
`--silence-autosupport`: AutoSupportバンドルをNetAppに自動的に送信しないでください(デフォルトはTRUE)。
`--silent`:インストール中にMOST出力を無効にします。
`--trident-image string`：インストールするAstra Tridentイメージ。
`--use-custom-yaml`:セットアップディレクトリに存在する既存のYAMLファイルを使用します。
`--use-ipv6`：Astra Tridentの通信にIPv6を使用します。

ログ

フラグを使用してAstra Tridentからログを出力します `logs`。

```
tridentctl logs [flags]
```

フラグ

`-a、 --archive`：特に指定がないかぎり、すべてのログを含むサポートアーカイブを作成します。
`-h、 --help`：ログのヘルプ。
`-l、 --log string`：表示するAstra Tridentのログ。Trident | auto | Trident - operator | allのいずれか（デフォルトは「auto」）。
`--node string`：ノードポッドログの収集元となるKubernetesノード名。
`-p,: --previous`以前のコンテナインスタンスが存在する場合は、そのインスタンスのログを取得します。
`--sidecars`:サイドカーコンテナのログを取得します。

送信

コマンドを使用し `send`て、Astra Tridentからリソースを送信します。

```
tridentctl send [option]
```

オプション

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

アンインストール

フラグを使用し `uninstall`てAstra Tridentをアンインストールします。

```
tridentctl uninstall [flags]
```

フラグ

`-h, --help`:アンインストールのヘルプ。
`--silent`:アンインストール中にほとんどの出力を無効にします。

更新

Astra Tridentでリソースを変更するには、コマンドを使用し `update`ます。

```
tridentctl update [option]
```

オプション

`backend`：Astra Tridentのバックエンドを更新します。

バックエンドの状態を更新

バックエンド処理を一時停止または再開するには、コマンドを使用し `update backend state`ます。

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

考慮すべきポイント

- TridentBackendConfig（tbc）を使用してバックエンドを作成した場合、ファイルを使用してバックエンドを更新することはできません `backend.json` 。
- がtbcに設定されている場合 `userState` は、コマンドを使用して変更することはできません `tridentctl update backend state <backend-name> --user-state suspended/normal` 。
- tbcで設定されたvia tridentctlを再び設定するには `userState` 、 `userState` tbcからフィールドを削除する必要があります。これは、コマンドを使用して実行でき `kubectl edit tbc` ます。フィールドを削除したら `userState` 、コマンドを使用してバックエンドのを変更できます `tridentctl update backend state` `userState` 。
- を使用して `tridentctl update backend state` を変更し `userState`ます。またはファイルを使用して更新することもでき `userState` `TridentBackendConfig` `backend.json` ます。これにより、バックエンドの完全な再初期化がトリガーされ、時間がかかる場合があります。

  フラグ

  `-h,: --help`バックエンド状態のヘルプ。
  `--user-state`：バックエンド処理を一時停止するには、に設定します `suspended`。バックエンド処理を再開するには、をに設定し `normal`ます。に設定されている場合 `suspended`：

- `AddVolume` `Import Volume` 一時停止しています。
- `CloneVolume`、 `ResizeVolume`、 `PublishVolume`、 `UnPublishVolume`、、 `CreateSnapshot` `GetSnapshot` `RestoreSnapshot`、、 `DeleteSnapshot`、 `RemoveVolume`、、 `GetVolumeExternal` `ReconcileNodeAccess` 引き続き使用できます。

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

- 例： *

JSON

ファイルを使用してを更新するには、次の手順を実行し `userState` `backend.json` ます。

1. ファイルを編集して `backend.json` 、値が「中断」に設定されたフィールドを含め `userState` ます。
2. コマンドと更新されたファイルへのパスを使用して、バックエンドを更新し `tridentctl backend update` `backend.json` ます。

   例： `tridentctl backend update -f /<path to backend JSON file>/backend.json`

```
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "<redacted>",
    "svm": "nas-svm",
    "backendName": "customBackend",
    "username": "<redacted>",
    "password": "<redacted>",
    "userState": "suspended",
}
```

YAML

tbcが適用されたら、コマンドを使用して編集できます `kubectl edit <tbc-name> -n <namespace>` 。次に、オプションを使用してバックエンド状態をsuspendに更新する例を示し `userState: suspended` ます。

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-ontap-nas
spec:
  version: 1
  backendName: customBackend
  storageDriverName: ontap-nas
  managementLIF: <redacted>
  svm: nas-svm
userState: suspended
  credentials:
    name: backend-tbc-ontap-nas-secret
```

バージョン

フラグを使用して `version`、および実行中のTridentサービスのバージョンを出力し `tridentctl`ます。

```
tridentctl version [flags]
```

フラグ

`--client`:クライアントバージョンのみ(サーバーは必要ありません)。
`-h, --help`:バージョンのヘルプ。

Astra Trident を監視

Astra Tridentは、Astra Tridentのパフォーマンス監視に使用できるPrometheus指標エンドポイントのセットを提供します。

概要

Astra Trident が提供する指標を使用すると、次のことが可能になります。

- Astra Trident の健常性と設定を保持処理が成功した方法と、想定どおりにバックエンドと通信できるかどうかを調べることができます。
- バックエンドの使用状況の情報を調べて、バックエンドでプロビジョニングされているボリュームの数や消費されているスペースなどを確認します。
- 利用可能なバックエンドにプロビジョニングされたボリュームの量のマッピングを維持します。
- パフォーマンスを追跡する。Astra Trident がバックエンドと通信して処理を実行するのにどれくらいの時間がかかるかを調べることができます。

デフォルトでは、Tridentの指標はエンドポイントの `/metrics`ターゲットポートで公開され `8001`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。にすることも、としてPrometheusを実行することもでき "ネイティブアプリケーション"ます "コンテナ化された Prometheus 環境"。

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。ここで "ブログ"は、Astra TridentでPrometheusとGrafanaを使用して指標を取得する方法について説明します。このブログでは、Kubernetesクラスタのオペレータとしてprometheusを実行する方法と、Astra Tridentの指標を取得するためのServiceMonitorの作成について説明しています。

手順 2 ： Prometheus ServiceMonitor を作成します

Trident指標を使用するには、サービスを監視してポートでリスン `metrics`するPrometheusサービスモニタを作成する必要があります `trident-csi`。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、サービスから返されたメトリックを取得し `trident-csi`、特にサービスのエンドポイントを検索し `metrics`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。

Astra Tridentから直接利用できる指標に加えて、Kubeletは独自の指標エンドポイントを介して多くの指標を公開して `kubelet_volume_*`います。Kubelet では、接続されているボリュームに関する情報、およびポッドと、それが処理するその他の内部処理を確認できます。を参照してください "ここをクリック"。

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

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

次に、 PrompQL クエリーのいくつかを示します。

Trident の健常性情報を取得

- **Astra Trident** からの **HTTP 2XX** 応答の割合

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

- **Astra Trident** からのステータスコードによる **REST** 応答の割合

```
(sum (trident_rest_ops_seconds_total_count) by (status_code)  / scalar
(sum (trident_rest_ops_seconds_total_count))) * 100
```

- **Astra Trident** によって実行された処理の平均時間（ミリ秒）

```
sum by (operation)
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by
(operation)
(trident_operation_duration_milliseconds_count{success="true"})
```

Astra Trident の使用状況に関する情報を入手

- 平均体積サイズ

```
trident_volume_allocated_bytes/trident_volume_count
```

- 各バックエンドによってプロビジョニングされた合計ボリューム容量

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

個々のボリュームの使用状況を取得する

これは、 kubelet 指標も収集された場合にのみ有効になります。

- 各ボリュームの使用済みスペースの割合

```
kubelet_volume_stats_used_bytes / kubelet_volume_stats_capacity_bytes *
100
```

Astra Trident AutoSupport の計測データ

デフォルトでは、 Astra Trident は Prometheus 指標と基本バックエンド情報を毎日定期的にネットアップに送信します。

- Astra TridentからNetAppへのPrometheus指標と基本的なバックエンド情報の送信を停止するには、Astra Tridentのインストール時にフラグを渡します。 `--silence-autosupport`
- Astra Tridentでは、経由でコンテナログをNetAppサポートにオンデマンドで送信することもできます `tridentctl send autosupport`。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、NetAppを承認する必要がありますhttps://www.netapp.com/company/legal/privacy-policy/["プライバシーポリシー"^]。
- 指定しないと、 Astra Trident は過去 24 時間からログを取得します。
- フラグを使用してログの保持期間を指定できます `--since`。例： `tridentctl send autosupport --since=1h`。収集された情報は、Astra Tridentと一緒にインストールされたコンテナを介して送信され`trident-autosupport`ます。コンテナイメージはから入手できます "Trident AutoSupport の略"。
- Trident AutoSupport は、個人情報（ PII ）や個人情報を収集または送信しません。Tridentコンテナイメー

ジ自体には適用されないが付属して "EULA" います。データのセキュリティと信頼に対するネットアップの取り組みについて詳しくは、こちらをご覧 "ここをクリック"ください。

Astra Trident から送信されるペイロードの例を次に示します。

```
---
items:
- backendUUID: ff3852e1-18a5-4df4-b2d3-f59f829627ed
  protocol: file
  config:
    version: 1
    storageDriverName: ontap-nas
    debug: false
    debugTraceFlags:
    disableDelete: false
    serialNumbers:
    - nwkvzfanek_SN
    limitVolumeSize: ''
  state: online
  online: true
```

- AutoSupport メッセージは、ネットアップの AutoSupport エンドポイントに送信されます。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、フラグを使用できます `--image-registry`。
- 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。これを行うには、を使用し `tridentctl install --generate-custom-yaml`てYAMLファイルを作成し、にコンテナの `trident-deployment.yaml`引数を `trident-autosupport`追加し `--proxy-url`ます。

Astra Trident の指標を無効化

無効メトリクスが報告されないようにするには、(フラグを使用して)カスタムYAMLを生成し、それらを編集して、 `--metrics`コンテナに対してフラグが呼び出されないように `trident-main`する必要があります `--generate-custom-yaml`。

Astra Trident をアンインストール

Astra Tridentのアンインストールには、Astra Tridentのインストールと同じ方法を使用する必要があります。

タスクの内容

- アップグレード、依存関係の問題、アップグレードの失敗または不完全な完了後に見つかったバグの修正が必要な場合は、Astra Tridentをアンインストールし、該当する手順を使用して以前のバージョンを再インストールする必要があります"バージョン"。これは、以前のバージョンに_downgrade_toを実行するための唯一の推奨方法です。
- アップグレードと再インストールを簡単に行うため、Astra Tridentをアンインストールしても、Astra Tridentで作成されたCRDや関連オブジェクトは削除されません。Astra Tridentとそのすべてのデータを完

全に削除する必要がある場合は、を参照してください"Astra TridentとCRDを完全に削除"。

開始する前に

Kubernetesクラスタの運用を停止する場合は、Astra Tridentで作成されたボリュームを使用するすべてのアプリケーションをアンインストールする前に削除する必要があります。これにより、PVCが削除される前にKubernetesノードで非公開になります。

元のインストール方法を決定する

Astra Tridentは、インストール時と同じ方法でアンインストールする必要があります。アンインストールする前に、Astra Tridentの最初のインストールに使用したバージョンを確認します。

1. ポッドの検査に使用し `kubectl get pods -n trident`ます。
   - オペレータポッドがない場合は、を使用してAstra Tridentをインストールしました `tridentctl`。
   - オペレータポッドがある場合、Astra Tridentは手動またはHelmを使用してインストールされています。
2. オペレータポッドがある場合は、を使用 `kubectl describe tproc trident`してAstra TridentがHelmを使用してインストールされているかどうかを確認します。
   - Helmラベルがある場合は、Helmを使用してAstra Tridentがインストールされています。
   - Helmラベルがない場合は、Astra TridentをTridentオペレータを使用して手動でインストールしています。

Tridentオペレータのインストールをアンインストールする

Tridentオペレータのインストールは手動でアンインストールすることも、Helmを使用してアンインストールすることもできます。

手動インストールのアンインストール

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

1. **CR**を編集し `**TridentOrchestrator**`てアンインストールフラグを設定：

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

   フラグがに設定されて `true`いる場合、 `uninstall`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。

2. 削除 **`TridentOrchestrator`**：Astra Tridentの導入に使用したCRを削除することで `TridentOrchestrator`、オペレータにTridentをアンインストールするよう指示します。オペレータがの削除処理を行い `TridentOrchestrator`、Astra Tridentの導入とデーモンセットの削除に進み、インストール時に作成したTridentポッドが削除されます。

```
kubectl delete -f deploy/<bundle.yaml> -n <namespace>
```

Helmインストールのアンインストール

Helmを使用してAstra Tridentをインストールした場合は、を使用してアンインストールできます `helm uninstall`。

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

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

インストールのアンインストール `tridentctl`

CRDと関連オブジェクトを除くAstra Tridentに関連付けられているすべてのリソースを削除するには、のコマンドを `tridentctl`使用し `uninstall`ます。

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

Trident for Docker が必要です

導入の前提条件

Trident を導入するには、必要なプロトコルをホストにインストールして設定しておく必要があります。

要件を確認します

- 導入環境がすべての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のバージョンが古い場合は、を参照してください "インストールまたは更新します"。

```
docker --version
```

- プロトコルの前提条件がホストにインストールおよび設定されていることを確認します。

NFSツール

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

RHEL 8以降

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

Ubuntu

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

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

iSCSIツール

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

RHEL 8以降

1. 次のシステムパッケージをインストールします。

```
sudo yum install -y lsscsi iscsi-initiator-utils sg3_utils device-
mapper-multipath
```

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

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

3. スキャンを手動に設定：

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

4. マルチパスを有効化：

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

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

5. および `multipathd`が実行されていることを確認し `iscsid`ます。

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

6. 有効にして開始 `iscsi`：

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

Ubuntu

1. 次のシステムパッケージをインストールします。

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

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

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

3. スキャンを手動に設定：

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

4. マルチパスを有効化：

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

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

5. とが `multipath-tools`有効で実行されていることを確認し `open-iscsi`ます。

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

NVMeツール

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

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

RHEL 9

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

Ubuntu

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

Astra Trident を導入

Docker向けAstra Tridentは、NetAppストレージプラットフォーム向けのDockerエコシステムと直接統合できます。ストレージプラットフォームから Docker ホストまで、ストレージリソースのプロビジョニングと管理をサポートします。また、将来プラットフォームを追加するためのフレームワークもサポートします。

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）

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
```

バージョンが古い場合は、 "インストール環境をインストールまたは更新します"。

手順

1. 構成ファイルを作成し、次のオプションを指定します。
   - `config`:デフォルトのファイル名はです `config.json`が、ファイル名とともにオプションを指定することで、任意の名前を使用できます `config`。構成ファイルは、ホストシステムのディレクトリに配置する必要があります `/etc/netappdvp`。
   - `log-level`：ログレベル(`debug`、 `info` `warn`、、 `error` `fatal`を指定します）。デフォルトはです `info`。
   - `debug`:デバッグロギングを有効にするかどうかを指定します。デフォルトは false です。true の場合、ログレベルを上書きします。
     i. 構成ファイルの場所を作成します。

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

     ii. 構成ファイルを作成します

```
cat << EOF > /etc/netappdvp/config.json
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1"
}
EOF
```

2. マネージドプラグインシステムを使用して Astra Trident を起動を、使用しているプラグインのバージョン（xxx.xx.x）に置き換えます `<version>`。

```
docker plugin install --grant-all-permissions --alias netapp
netapp/trident-plugin:<version> config=myConfigFile.json
```

3. 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）

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/v24.10.0/trident-
installer-24.06.0.tar.gz
tar zxf trident-installer-24.06.0.tar.gz
```

   b. ビンパス内の場所に移動します。

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

c. 構成ファイルの場所を作成します。

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

d. 構成ファイルを作成します

```
cat << EOF > /etc/netappdvp/ontap-nas.json
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1"
}
EOF
```

2. バイナリを配置して構成ファイルを作成したら、目的の構成ファイルを使用してTridentデーモンを起動します。

```
sudo trident --config=/etc/netappdvp/ontap-nas.json
```

指定されていない場合、ボリュームドライバのデフォルト名は「NetApp」です。

デーモンが開始されたら、 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

systemdベースのシステム用のサンプルユニットファイルは、 `contrib/trident.service.example`Gitリポジトリにあります。RHELでファイルを使用するには、次の手順を実行します。

1. ファイルを正しい場所にコピーします。

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

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

```
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たびに`コマンドを実行して `systemctl daemon-reload`変更を認識します

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。アップグレードプロセスでは、 `docker volume`プラグインへのコマンドが正常に実行されず、プラグインが再度実行されるまでアプリケーションはボリュームをマウントできません。ほとんどの場合、これは秒の問題です。

アップグレード

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

4. プラグインを有効にします。

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

5. プラグインが有効になっていることを確認します。

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

6. ボリュームが表示されることを確認します。

```
docker volume ls
DRIVER              VOLUME NAME
netapp:latest       my_volume
```

> 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"。エラーが発生した場合は、まずプラグインを無効にしてからプラグインを削除し、追加のconfigパラメータを渡して必要なバージョンのAstra Tridentをインストールします。 `docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant-all -permissions config=config.json`

アンインストール

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ドライバ名を指定する標準コマンドを使用して、ボリュームの作成、クローニング、削除を簡単に実行でき `docker volume`ます。

ボリュームの作成

- 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
```

anyを指定しない場合は"オプション"、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）で指定できます。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
```

ドライバを使用している場合、 `solidfire-san`上記の例ではボリュームを削除およびパージします。

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

ボリュームのクローニング

、 `ontap-san`、 `solidfire-san`、および `gcp-cvs storage drivers`を使用する場合は `ontap-

nas、Astra Tridentでボリュームをクローニングできます。ドライバまたは `ontap-nas-economy`ドライバを使用している場合、 `ontap-nas-flexgroup`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

```
docker volume inspect firstVolume

[
    {
        "Driver": "ontap-nas",
        "Labels": null,
        "Mountpoint": "/var/lib/docker-volumes/ontap-
nas/netappdvp_firstVolume",
        "Name": "firstVolume",
        "Options": {},
        "Scope": "global",
        "Status": {
            "Snapshots": [
                {
                    "Created": "2017-02-10T19:05:00Z",
                    "Name": "hourly.2017-02-10_1505"
                }
            ]
        }
    }
]

docker volume create -d ontap-nas --name clonedVolume -o from=firstVolume
clonedVolume

docker volume rm clonedVolume
docker volume create -d ontap-nas --name volFromSnap -o from=firstVolume
-o fromSnapshot=hourly.2017-02-10_1505
volFromSnap

docker volume rm volFromSnap
```

外部で作成されたボリュームにアクセス

外部で作成したブロックデバイス（またはそのクローン）には、パーティションがなく、かつファイルシステムがAstra Tridentでサポートされている場合にのみ、Trident **を使用してコンテナからアクセスできます（例：`ext4`-フォーマット `/dev/sdc1`はAstra Trident経由ではアクセスできません）。

ドライバ固有のボリュームオプション

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

ボリューム作成処理では、これらのオプションを簡単に使用できます。CLI処理中にoperatorを使用してオプ

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

ONTAP ボリュームのオプション

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

オプション	説明
`size`	ボリュームのサイズ。デフォルトは 1GiB です。
`spaceReserve`	ボリュームをシンプロビジョニングまたはシックプロビジョニングします。デフォルトはシンです。有効な値は、 `none`（thin provisioned）と `volume`（thick provisioned）です。
`snapshotPolicy`	Snapshot ポリシーが目的の値に設定されます。デフォルトはで、 `none`ボリュームのSnapshotは自動的に作成されません。ストレージ管理者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。ボリューム内の任意のディレクトリを参照することで、Snapshotに保存されているデータをリカバリでき `.snapshot`ます。
`snapshotReserve`	これにより、 Snapshot リザーブの割合が希望する値に設定されます。デフォルト値は no で、 Snapshot ポリシーを選択した場合は ONTAP によって snapshotReserve が選択されます（通常は 5% ）。 Snapshot ポリシーがない場合は 0% が選択されます。構成ファイルのすべての ONTAP バックエンドに対して snapshotReserve のデフォルト値を設定できます。また、この値は、 ONTAP-NAS-エコノミー を除くすべての ONTAP バックエンドでボリューム作成オプションとして使用できます。
`splitOnClone`	ボリュームをクローニングすると、そのクローンが原因 ONTAP によって親から即座にスプリットされます。デフォルトはです `false`。クローンボリュームのクローニングは、作成直後に親からクローンをスプリットする方法を推奨します。これは、ストレージ効率化の効果がまったくないためです。たとえば、空のデータベースをクローニングしても時間は大幅に短縮されますが、ストレージはほとんど削減されません。そのため、クローンはすぐにスプリットすることを推奨します。

オプション	説明
`encryption`	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです。`false`このオプションを使用するには、クラスタでNVE のライセンスが設定され、有効になっている必要があります。 NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームがに有効になります。 詳細については、を参照してください"Astra TridentとNVEおよびNAEの相互運用性"。
`tieringPolicy`	ボリュームに使用する階層化ポリシーを設定します。これにより、アクセス頻度の低いコールドデータをクラウド階層に移動するかどうかが決まります。

以下は、 NFS * のみ * 用の追加オプションです。

オプション	説明
`unixPermissions`	これにより、ボリューム自体の権限セットを制御できます。デフォルトでは、権限はまたは番号0755に設定され`---rwxr-xr-x`、`root`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。
`snapshotDir`	このをに設定する`true`と、 `.snapshot`ボリュームにアクセスするクライアントからディレクトリが表示されます。デフォルト値はで `false`、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。 `.snapshot`公式のMySQLイメージなど、一部のイメージは、ディレクトリが表示されているときに想定どおりに機能しません `.snapshot`。
`exportPolicy`	ボリュームで使用するエクスポートポリシーを設定します。デフォルトはです `default`。
`securityStyle`	ボリュームへのアクセスに使用するセキュリティ形式を設定します。デフォルトはです `unix`。有効な値は`unix`とです `mixed`。

以下の追加オプションは、 iSCSI * のみ * 用です。

オプション	説明
`fileSystemType`	iSCSI ボリュームのフォーマットに使用するファイルシステムを設定します。デフォルトはです `ext4`。有効な値は `ext3`、 `ext4`および `xfs`です。
`spaceAllocation`	このをに設定する `false`と、`LUN`のスペース割り当て機能が無効になります。デフォルト値はです `true`。ボリュームのスペースが不足してボリューム内のLUNへの書き込みを受け付けられない場合、ONTAPはホストに通知します。また、このオプションを使用すると、ホストでデータが削除されたときにONTAPでスペースが自動的に再生されます。

例

以下の例を参照してください。

- 10GiBのボリュームを作成します。

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

- 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の場合、 `none`Tridentは0%のスナップショット予約を使用します。

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

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

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

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

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

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

- Snapshot ポリシーを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、そのボリュームに関連付けられているQoSポリシーが命名規則を使用して指定され `-o type=service_level`ます。

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

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

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

例

QoS 定義を含む次のサンプル構成ファイルを参照してください。

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

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

- 10GiB の Gold ボリュームを作成します。

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

- 100GiB Bronze ボリュームを作成します。

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

ログを収集します

トラブルシューティングに役立つログを収集できます。ログの収集方法は、 Docker プラグインの実行方法によって異なります。

トラブルシューティング用にログを収集する

手順

1. 推奨される管理プラグイン方式（コマンドを使用）でAstra Tridentを実行している場合は `docker plugin`、次のように確認します。

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

標準的なロギングレベルでは、ほとんどの問題を診断できます。十分でない場合は、デバッグロギングを有効にすることができます。

2. デバッグロギングをイネーブルにするには、デバッグロギングをイネーブルにしてプラグインをインストールします。

```
docker plugin install netapp/trident-plugin:<version> --alias <alias>
debug=true
```

または、プラグインがすでにインストールされている場合にデバッグログを有効にします。

```
docker plugin disable <plugin>
docker plugin set <plugin> debug=true
docker plugin enable <plugin>
```

3. ホストでバイナリ自体を実行している場合、ログはホストのディレクトリにあり `/var/log/netappdvp`ます。デバッグログを有効にするには、プラグインを実行するタイミングを指定します `-debug。

一般的なトラブルシューティングのヒント

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

```
Error response from daemon: dial unix /run/docker/plugins/<id>/netapp.sock:
```

```
connect: no such file or directory
```

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

- コンテナへのPVのマウントに問題がある場合は、がインストールされて実行されていることを確認して `rpcbind`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、が実行されているかどうかを確認します `rpcbind`。rpcbindサービスのステータスを確認するには、または同等のを実行し `systemctl status rpcbind`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には、コンテナ化されたプラグインのオプションを使用して異なる名前を付けるか、 `--volume-driver`ホスト上でTridentをインスタンス化するときにオプションを使用して異なる名前を付けることが重要です `--alias`。

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

1. エイリアスと構成ファイルを指定して、最初のインスタンスを起動します。

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

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

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

3. ドライバ名としてエイリアスを指定するボリュームを作成します。

   たとえば、 gold ボリュームの場合：

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

   たとえば、 Silver ボリュームの場合：

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

従来の（バージョン **1.12** 以前）の場合の手順

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

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

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

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

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

   たとえば、 NFS の場合：

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

   たとえば、 iSCSI の場合：

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

ストレージ構成オプション

Astra Trident 構成で使用できる設定オプションを確認してください。

グローバル構成オプション

以下の設定オプションは、使用するストレージプラットフォームに関係なく、すべての Astra Trident 構成に適用されます。

オプション	説明	例
`version`	構成ファイルのバージョン番号	`1`
`storageDriverName`	ストレージドライバの名前	`ontap-nas ontap-san`、、`ontap-nas-economy`、、`ontap-nas-flexgroup solidfire-san`

オプション	説明	例
`storagePrefix`	ボリューム名のオプションのプレフィックス。デフォルト：`netappdvp_`	`staging_`
`limitVolumeSize`	ボリュームサイズに関するオプションの制限。デフォルト：""（強制なし）	`10g`

Elementバックエンドには（デフォルトを含めて）使用しない `storagePrefix`でください。デフォルトでは、 `solidfire-san`ドライバはこの設定を無視し、プレフィックスは使用しません。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。この `size`オプションは、すべてのコントローラタイプで使用できます。デフォルトのボリュームサイズの設定方法の例については、 ONTAP の設定に関するセクションを参照してください。

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

ONTAP構成

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

オプション	説明	例
`managementLIF`	ONTAP 管理 LIF の IP アドレス。Fully Qualified Domain Name （ FQDN ；完全修飾ドメイン名）を指定できます。	`10.0.0.1`

オプション	説明	例
`dataLIF`	プロトコル LIF の IP アドレス。 • ONTAP NASドライバ*：を指定することをお勧めします `dataLIF`。指定しない場合は、Astra TridentがSVMからデータLIFを取得します。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。 • ONTAP SANドライバ*: iSCSIには指定しないでくださいAstra Tridentは、を使用して"ONTAP の選択的LUNマップ"、マルチパスセッションの確立に必要なiSCI LIFを検出します。が明示的に定義されている場合は、警告が生成され`dataLIF`ます。	`10.0.0.2`
`svm`	使用する Storage Virtual Machine（管理 LIF がクラスタ LIF である場合は必須）	`svm_nfs`
`username`	ストレージデバイスに接続するユーザ名	`vsadmin`
`password`	ストレージ・デバイスに接続するためのパスワード	`secret`
`aggregate`	プロビジョニング用のアグリゲート（オプション。設定する場合は SVM に割り当てる必要があります）。ドライバの場合 `ontap-nas-flexgroup`、このオプションは無視されます。SVM に割り当てられたすべてのアグリゲートを使用して FlexGroup ボリュームがプロビジョニングされます。	`aggr1`
`limitAggregateUsage`	オプション。使用率がこの割合を超えている場合は、プロビジョニングを失敗させます	`75%`

オプション	説明	例
nfsMountOptions	NFS マウントオプションのきめ細かな制御。デフォルトは「 -o nfsvers=3 」です。ドライバと **`ontap-nas-economy`**ドライバでのみ使用でき **`ontap-nas`**ます。 "ここでは、 NFS ホストの設定情報を参照してください"です。	-o nfsvers=4
igroupName	Astra Tridentでは、ノードごとにASを `netappdvp`作成、管理し `igroups`ます。 この値は変更したり省略したりすることはできません。 ドライバーでのみ使用可能 **ontap-san**。	netappdvp
limitVolumeSize	要求可能な最大ボリュームサイズ。	300g
qtreesPerFlexvol	FlexVol あたりの最大 qtree 数は [50 、 300] の範囲で指定する必要があります。デフォルトは 200 です。 *ドライバの場合 ontap-nas-economy、このオプションを使用すると、FlexVolあたりの最大qtree数*をカスタマイズできます。	300
sanType	*ドライバでのみサポートされてい `ontap-san`ます。*iSCSIまたは `nvme`NVMe/TCPの選択に使用し `iscsi`ます。	`iscsi`空白の場合
limitVolumePoolSize	* `ontap-san-economy`および `ontap-san-economy`ドライバでのみサポートされています。*ONTAP ONTAP NASエコノミードライバおよびONTAP SANエコノミードライバでFlexVolサイズを制限します。	300g

作成するすべてのボリュームでデフォルトのオプションを指定しなくても済むようになっています。

オプション	説明	例
spaceReserve	スペースリザベーションモード（ none `シンプロビジョニング）または ` volume（シック）	none

オプション	説明	例
snapshotPolicy	使用するSnapshotポリシー。デフォルトは none	none
snapshotReserve	Snapshotリザーブの割合。デフォルトはONTAP のデフォルトをそのまま使用する場合はです	10
splitOnClone	作成時に親からクローンをスプリットします。デフォルトは false	false
encryption	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです。`false`このオプションを使用するには、クラスタでNVE のライセンスが設定され、有効になっている必要があります。 NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームがに有効になります。 詳細については、を参照してください"Astra TridentとNVEおよびNAEの相互運用性"。	正しい
unixPermissions	プロビジョニングされたNFSボリュームのNASオプション。デフォルトは 777	777
snapshotDir	ディレクトリにアクセスするためのNASオプション .snapshot。デフォルトは false	true
exportPolicy	NFSエクスポートポリシーで使用するNASオプション。デフォルトは default	default
securityStyle	プロビジョニングされたNFSボリュームにアクセスするためのNASオプション。 NFSのサポート mixed`と `unix`セキュリティ形式デフォルトはです `unix。	unix
fileSystemType	ファイルシステムタイプを選択するためのSANオプション。デフォルトは ext4	xfs
tieringPolicy	使用する階層化ポリシー。デフォルトは none。`snapshot-only`ONTAP 9 .5より前のSVM-DR構成の場合	none

スケーリングオプション

ドライバと `ontap-san`ドライバを使用すると、 `ontap-nas`DockerボリュームごとにONTAP FlexVolが作成されます。ONTAP 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ボリュームの要件がこの制限内に収まる場合は、 `ontap-nas`FlexVolで提供される追加機能（Dockerボリューム単位のSnapshotやクローニングなど）を考慮して、ドライバがNASソリューションとして推奨されます。

FlexVolの制限で対応できない数のDockerボリュームが必要な場合は、または `ontap-san-economy`ドライバを選択します `ontap-nas-economy`。

`ontap-nas-economy`ドライバは、自動的に管理されるFlexVolのプール内にONTAP qtreeとしてDockerボリュームを作成します。qtree の拡張性は、クラスタノードあたり最大 10 、 000 、クラスタあたり最大 2 、 40 、 000 で、一部の機能を犠牲にすることで大幅に向上しています。この `ontap-nas-economy`ドライバは、Dockerボリューム単位のSnapshotやクローニングをサポートしていません。

この `ontap-nas-economy`ドライバは現在、Docker Swarmではサポートされていません。これは、Swarmが複数のノード間でボリューム作成をオーケストレーションしないためです。

`ontap-san-economy`ドライバは、自動的に管理されるFlexVolの共有プール内にONTAP LUNとしてDockerボリュームを作成します。この方法により、各 FlexVol が 1 つの LUN に制限されることはなく、 SAN ワークロードのスケーラビリティが向上します。ストレージアレイに応じて、 ONTAP はクラスタあたり最大 16384 個の LUN をサポートします。このドライバは、ボリュームが下位の LUN であるため、 Docker ボリューム単位の Snapshot とクローニングをサポートします。

ドライバを選択する `ontap-nas-flexgroup`と、数十億個のファイルを含むペタバイト規模まで拡張可能な単一ボリュームへの並列処理を強化できます。FlexGroup のユースケースとしては、 AI / ML / DL 、ビッグデータと分析、ソフトウェアのビルド、ストリーミング、ファイルリポジトリなどが考えられます。Trident 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FlexGroupメンバー/ボリュームの最小サイズは100GiB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については、を参照してください "NetApp FlexGroup Volume Best Practices and Implementation Guide"。

同じ環境で高度な機能と大規模な拡張を実現するには、を使用してDocker Volume Pluginの複数のインスタンスを実行し、を使用して別の `ontap-nas-economy`インスタンスを実行し `ontap-nas`ます。

ONTAP 構成ファイルの例

<code> ONTAP NAS </code>ドライバのNFSの例

```
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1",
    "defaults": {
      "size": "10G",
      "spaceReserve": "none",
      "exportPolicy": "default"
    }
}
```

<code> ONTAP - NAS - FlexGroup </code>ドライバでのNFSの例

```
{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "password",
    "defaults": {
      "size": "100G",
      "spaceReserve": "none",
      "exportPolicy": "default"
    }
}
```

<code> ONTAP - nas-economy </code>ドライバでの**NFS**の例

```
{
    "version": 1,
    "storageDriverName": "ontap-nas-economy",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1"
}
```

<code> ONTAP SAN </code>ドライバの**iSCSI**の例

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1",
    "igroupName": "netappdvp"
}
```

<code> ONTAP SANエコノミー**</code>**ドライバでの**NFS**の例

```
{
    "version": 1,
    "storageDriverName": "ontap-san-economy",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi_eco",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1",
    "igroupName": "netappdvp"
}
```

<code> ONTAP SAN </code>ドライバの NVMe/TCP の例

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

Element ソフトウェアの設定

Element ソフトウェア（ NetApp HCI / SolidFire ）を使用する場合は、グローバルな設定値のほかに、以下のオプションも使用できます。

オプション	説明	例
`Endpoint`	\https：//<login>：<password>@<mvip>/ JSON -RPC /<element-version>	https://admin:admin@192.168.160.3/json-rpc/8.0
`SVIP`	iSCSI の IP アドレスとポート	10.0.0.7 ： 3260
`TenantName`	使用する SolidFire テナント（見つからない場合に作成）	`docker`
`InitiatorIFace`	iSCSI トラフィックをデフォルト以外のインターフェイスに制限する場合は、インターフェイスを指定します	`default`
`Types`	QoS の仕様	以下の例を参照してください

オプション	説明	例
`LegacyNamePrefix`	アップグレードされた Trident インストールのプレフィックス。1.3.2 より前のバージョンのTridentを使用していて、既存のボリュームでアップグレードを実行した場合は、volume-nameメソッドでマッピングされた古いボリュームにアクセスするためにこの値を設定する必要があります。	`netappdvp-`

この `solidfire-san`ドライバはDocker Swarmをサポートしていません。

Element ソフトウェア構成ファイルの例

```
{
    "version": 1,
    "storageDriverName": "solidfire-san",
    "Endpoint": "https://admin:admin@192.168.160.3/json-rpc/8.0",
    "SVIP": "10.0.0.7:3260",
    "TenantName": "docker",
    "InitiatorIFace": "default",
    "Types": [
        {
            "Type": "Bronze",
            "Qos": {
                "minIOPS": 1000,
                "maxIOPS": 2000,
                "burstIOPS": 4000
            }
        },
        {
            "Type": "Silver",
            "Qos": {
                "minIOPS": 4000,
                "maxIOPS": 6000,
                "burstIOPS": 8000
            }
        },
        {
            "Type": "Gold",
            "Qos": {
                "minIOPS": 6000,
                "maxIOPS": 8000,
                "burstIOPS": 10000
            }
        }
    ]
}
```

既知の問題および制限事項

Astra Trident と Docker を使用する際の既知の問題と制限事項について説明しています。

Trident Docker Volume Plugin を旧バージョンから **20.10** 以降にアップグレードすると、該当するファイルエラーまたはディレクトリエラーなしでアップグレードが失敗します。

回避策

1. プラグインを無効にします。

```
docker plugin disable -f netapp:latest
```

2. プラグインを削除します。

```
docker plugin rm -f netapp:latest
```

3. 追加のパラメータを指定してプラグインを再インストールし `config`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"です。

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の動作方法とドライバと `ontap-san`ドライバの機能により、 `ontap-nas`これらの制限内で動作できるのはこれらのドライバだけです。

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。

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し、リソース共有の障壁となります。たとえば、あるネームスペースの PVC を別のネームスペースから使用することはできません。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。を "ストレージクォータメカニズム"使用すると、グローバルなストレージクラス固有の容量とオブジェクト数の消費制限をネームスペース単位で実装できます。さらに、を使用する "範囲制限"と、PVC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に収まるようになります。

これらの値はネームスペース単位で定義されます。つまり、各ネームスペースに、リソースの要件に応じた値を定義する必要があります。の詳細については、こちらを参照してください "クォータの活用方法"。

ストレージ構成

ネットアップポートフォリオの各ストレージプラットフォームには、コンテナ化されたアプリケーションやそうでないアプリケーションに役立つ独自の機能があります。

プラットフォームの概要

Trident は ONTAP や Element と連携1 つのプラットフォームが他のプラットフォームよりもすべてのアプリケーションとシナリオに適しているわけではありませんが、プラットフォームを選択する際には、アプリケーションのニーズとデバイスを管理するチームを考慮する必要があります。

使用するプロトコルに対応したホストオペレーティングシステムのベースラインベストプラクティスに従う必要があります。必要に応じて、アプリケーションのベストプラクティスを適用する際に、バックエンド、ストレージクラス、 PVC の設定を利用して、特定のアプリケーションのストレージを最適化することもできます。

ONTAP と Cloud Volumes ONTAP のベストプラクティス

Trident 向けに ONTAP と Cloud Volumes ONTAP を設定するためのベストプラクティスをご確認ください。

次に示す推奨事項は、 Trident によって動的にプロビジョニングされたボリュームを消費するコンテナ化されたワークロード用に ONTAP を設定する際のガイドラインです。それぞれの要件を考慮し、環境内で適切かどうかを評価する必要があります。

Trident 専用の **SVM** を使用

Storage Virtual Machine （ SVM ）を使用すると、 ONTAP システムのテナントを分離し、管理者が分離できます。SVM をアプリケーション専用にしておくと、権限の委譲が可能になり、リソース消費を制限するためのベストプラクティスを適用できます。

SVM の管理には、いくつかのオプションを使用できます。

- バックエンド構成でクラスタ管理インターフェイスを適切なクレデンシャルとともに指定し、 SVM 名を指定します。
- ONTAP System Manager または CLI を使用して、 SVM 専用の管理インターフェイスを作成します。
- NFS データインターフェイスで管理ロールを共有します。

いずれの場合も、インターフェイスは DNS にあり、 Trident の設定時には DNS 名を使用する必要があります。これにより、ネットワーク ID を保持しなくても SVM-DR などの一部の DR シナリオが簡単になります。

専用の管理 LIF または共有の管理 LIF を SVM に使用する方法は推奨されませんが、ネットワークセキュリティポリシーを選択した方法と一致させる必要があります。いずれにせよ、最大限の柔軟性を確保するためには、管理LIFにDNS経由でアクセスできるようにする必要があります。これをTridentと組み合わせて使用する必要があります "SVM-DR"。

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。正確な制限を確認するには、使用しているプラットフォームおよびONTAPバージョンに対応したを参照してください "NetApp Hardware Universe"。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`ドライバと `ontap-san`ドライバは、作成される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-economy`ドライバは、200 PVSごとに約1つのFlexVolumeを作成します（50～300の間で設定可能）。 `ontap-san-economy`ドライバは、100 PVSごとに約1つのFlexVolumeを作成します（50～200の間で設定可能）。Trident がストレージシステム上の使用可能なボリュームをすべて消費しないようにするには、 SVM に制限を設定する必要があります。コマンドラインから実行できます。

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

の値 `max-volumes`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`この値は、個 々 のONTAPノードではなく、ONTAPクラスタ内のすべてのノードにプロビジョニングされたボリュームの合計です。その結果、 ONTAP クラスタノードの 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、SVMに1つのノードからしか割り当てられていない場合や、一方のノードから割り当てられたアグリゲートを（容量などの理由で）プロビジョニングできない場合は "アグリゲート"、Tridentでプロビジョニングされるすべてのボリュームのターゲットにもう一方のノードがなります。つまり、の値に達する前にそのノードのボリューム制限に達する可能性があり、その結果、Tridentとそのノードを使用するその他のボリューム処理の両方に影響が及ぶ可能性があり`max-volumes`ます。* クラスタ内の各ノードのアグリゲートを、 Trident が使用する SVM に同じ番号で確実に割り当てることで、この状況を回避できます。 *

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、定義でパラメータを `backend.json`使用し`limitVolumeSize`ます。

ストレージアレイでボリュームサイズを制御するだけでなく、 Kubernetes の機能も利用する必要があります。

Tridentで作成される**FlexVol**の最大サイズを制限する

ONTAPドライバSAN-EconomyドライバおよびONTAP NAS-Economyドライバのプールとして使用されるFlexVolの最大サイズを設定するには、 `limitVolumePoolSize` `backend.json`定義でパラメータを使用します。

双方向 **CHAP** を使用するように **Trident** を設定します

バックエンド定義で CHAP イニシエータとターゲットのユーザ名とパスワードを指定し、 Trident を使用して SVM で CHAP を有効にすることができます。バックエンド構成のパラメータを使用して `useCHAP`、TridentはCHAPを使用してONTAPバックエンドのiSCSI接続を認証します。

SVM QoS ポリシーを作成して使用します

SVM に適用された ONTAP QoS ポリシーを使用すると、 Trident でプロビジョニングされたボリュームが使用できる IOPS の数が制限されます。これにより、コンテナがTrident SVMの外部のワークロードに影響を及ぼすのを防ぎ、制御不能にすることができます "Bully を防止します"。

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。

- キーはバックエンド構成のブロックに `defaults`設定する必要があります `qosPolicy`。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"この種の侵害を防ぐように設計されています。ただし、特権コンテナという例外が 1 つあります。

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。これらの機能はデフォルトでは拒否されないため、を使用して無効にして "ポッドセキュリティポリシー"ください。

Kubernetes と外部ホストの両方からアクセスが必要なボリュームでは、 Trident ではなく管理者が導入した PV で、ストレージを従来の方法で管理する必要があります。これにより、 Kubernetes と外部ホストの両方が切断され、ボリュームを使用していない場合にのみ、ストレージボリュームが破棄されます。また、カスタムエクスポートポリシーを適用して、 Kubernetes クラスタノードおよび Kubernetes クラスタの外部にある

ターゲットサーバからのアクセスを可能にすることもできます。

専用のインフラノード（OpenShiftなど）や、ユーザアプリケーションをスケジュールできない他のノードを導入する場合は、ストレージリソースへのアクセスをさらに制限するために別 々 のエクスポートポリシーを使用する必要があります。これには、これらのインフラノードに導入されているサービス（ OpenShift Metrics サービスや Logging サービスなど）のエクスポートポリシーの作成と、非インフラノードに導入されている標準アプリケーションの作成が含まれます。

専用のエクスポートポリシーを使用します

Kubernetes クラスタ内のノードへのアクセスのみを許可するエクスポートポリシーが各バックエンドに存在することを確認する必要があります。Tridentはエクスポートポリシーを自動的に作成、管理できます。これにより、 Trident はプロビジョニング対象のボリュームへのアクセスを Kubernetes クラスタ内のノードに制限し、ノードの追加や削除を簡易化します。

また、エクスポートポリシーを手動で作成し、各ノードのアクセス要求を処理する 1 つ以上のエクスポートルールを設定することもできます。

- ONTAP CLIコマンドを使用し `vserver export-policy create`て、エクスポートポリシーを作成します。
- 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 `vserver export-policy rule create`。

これらのコマンドを実行すると、データにアクセスできる Kubernetes ノードを制限できます。

アプリケーション**SVM**で無効にする `showmount`

この `showmount`機能を使用すると、`NFS`クライアントが`SVM`に照会して使用可能な`NFS`エクスポートのリストを確認できます。`Kubernetes`クラスタに導入されたポッドは、データ`LIF`に対してコマンドを実行し、使用可能なマウント（アクセスできないマウントを含む）のリストを受け取ることができます `showmount -e`。これだけではセキュリティ上の妥協ではありませんが、権限のないユーザが NFS エクスポートに接続するのを阻止する可能性のある不要な情報が提供されます。

SVMレベルのONTAP CLIコマンドを使用して無効にする必要があり `showmount`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	最小値	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。QoSと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 のサービス品質"。

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。で最新バージョンを検索し "NetApp ライブラリ"ます。

- ONTAP *
- "NFSベストプラクティスおよび実装ガイド"
- "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特定のガイドラインがあるわけではありません。NetAppチームと協力し、を使用して最新のドキュメントを見つけることが重要 "NetApp ライブラリ"です。

Astra Trident を統合

Astra Tridentを統合するには、設計とアーキテクチャに関する次の要素を統合する必要があります。ドライバの選択と導入、ストレージクラスの設計、仮想プールの設計、永続的ボリューム要求（PVC）によるストレージプロビジョニング、ボリューム運用、Astra Tridentを使用したOpenShiftサービスの導入。

ドライバの選択と展開

ストレージシステム用のバックエンドドライバを選択して導入します。

ONTAP バックエンドドライバ

ONTAP バックエンドドライバは、使用されるプロトコルと、ストレージシステムでのボリュームのプロビジ

ョニング方法によって異なります。そのため、どのドライバを展開するかを決定する際には、慎重に検討する必要があります。

アプリケーションに共有ストレージを必要とするコンポーネント（同じ PVC にアクセスする複数のポッド）がある場合、 NAS ベースのドライバがデフォルトで選択されますが、ブロックベースの iSCSI ドライバは非共有ストレージのニーズを満たします。アプリケーションの要件と、ストレージチームとインフラチームの快適さレベルに基づいてプロトコルを選択してください。一般的に、ほとんどのアプリケーションでは両者の違いはほとんどないため、共有ストレージ（複数のポッドで同時にアクセスする必要がある場合）が必要かどうかに基づいて判断することがよくあります。

使用可能なONTAP バックエンドドライバは次のとおりです。

- `ontap-nas`：プロビジョニングされた各PVは、完全なONTAP FlexVolです。
- `ontap-nas-economy`：プロビジョニングされた各PVはqtreeであり、FlexVolあたりのqtree数は設定可能です（デフォルトは200）。
- `ontap-nas-flexgroup`：各PVがフルONTAP FlexGroupとしてプロビジョニングされ、SVMに割り当てられているすべてのアグリゲートが使用されます。
- `ontap-san`：プロビジョニングされた各PVは、専用のFlexVol内のLUNです。
- `ontap-san-economy`：プロビジョニングされた各PVはLUNであり、FlexVolあたりのLUN数は設定可能です（デフォルトは100）。

3 つの NAS ドライバの間で選択すると、アプリケーションで使用できる機能にいくつかの影響があります。

次の表では、 Astra Trident からすべての機能が提供されるわけではありません。一部の機能は、プロビジョニング後にストレージ管理者が適用する必要があります。上付き文字の脚注は、機能やドライバごとに機能を区別します。

ONTAP NASドライバ	スナップショット	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	Yesfootnote: 5[]	はい	Yesfootnote: 1[]	はい	Yesfootnote: 1[]
`ontap-nas-economy`	Yesfootnote: 3[]	Yesfootnote: 3[]	Yesfootnote: 5[]	はい	Yesfootnote: 3[]	はい	Yesfootnote: 3[]
`ontap-nas-flexgroup`	Yesfootnote: 1[]	いいえ	Yesfootnote: 5[]	はい	Yesfootnote: 1[]	はい	Yesfootnote: 1[]

Astra Trident は、 ONTAP 向けに 2 つの SAN ドライバを提供しています。このドライバの機能は次のとおりです。

ONTAP SANドライバ	スナップショット	クローン	マルチアタッチ	双方向**CHAP**	**QoS**	サイズ変更	レプリケーション
`ontap-san`	はい	はい	Yesfootnote: 4[]	はい	Yesfootnote: 1[]	はい	Yesfootnote: 1[]
`ontap-san-economy`	はい	はい	Yesfootnote: 4[]	はい	Yesfootnote: 3[]	はい	Yesfootnote: 3[]

> 脚注：1[]：Astra Tridentで管理されない出典：2[]：Astra Tridentで管理されるが、PVの詳細ではない脚注：3[]：Astra Tridentで管理されず、PVの詳細ではない出典：4[]：rawブロックボリュームでサポートYes [5]：Astra Tridentでサポート

PV に細分化されていない機能は FlexVol 全体に適用され、 PVS （共有 FlexVol 内の qtree または LUN ）にはすべて共通のスケジュールが適用されます。

上記の表からわかるように、との `ontap-nas-economy`機能の大部分は同じです。 `ontap-nas`ただし、スケジュールをPV単位で制御する機能が制限されるため、 `ontap-nas-economy`ディザスタリカバリやバックアップ計画に特に影響する可能性があります。ONTAPストレージでPVCクローン機能を活用したい開発チームでは、 `ontap-san`ドライバのまたはを `ontap-san-economy`使用している場合にのみ可能 `ontap-nas`です。

> `solidfire-san`ドライバはPVCをクローニングすることもできます。

Cloud Volumes ONTAP バックエンドドライバ

Cloud Volumes ONTAP は、ファイル共有や NAS および SAN プロトコル（ NFS 、 SMB / CIFS 、 iSCSI ）を提供するブロックレベルストレージなど、さまざまなユースケースでデータ制御とエンタープライズクラスのストレージ機能を提供します。Cloud Volume ONTAPと互換性があるドライバは `ontap-nas`、、 `ontap-nas-economy` `ontap-san` `ontap-san-economy`です。Cloud Volume ONTAP for Azure と Cloud Volume ONTAP for GCP に該当します。

ONTAP バックエンドドライバ用の**Amazon FSX**

Amazon FSx for NetApp ONTAPを使用すると、AWSにデータを格納する際のシンプルさ、即応性、セキュリティ、拡張性を活用しながら、使い慣れたNetAppの機能、パフォーマンス、管理機能を活用できます。FSx for ONTAPは、多くのONTAPファイルシステム機能と管理APIをサポートしています。Cloud Volume ONTAPと互換性があるドライバは `ontap-nas`、、 `ontap-nas-economy`、 `ontap-nas-flexgroup` `ontap-san` `ontap-san-economy`です。

NetApp HCI / SolidFireバックエンドドライバ

`solidfire-san`NetApp HCI / SolidFireプラットフォームで使用されるドライバは、管理者がQoS制限に基づいてElementバックエンドをTrident用に設定するのに役立ちます。Tridentでプロビジョニングするボリュームに特定のQoS制限を設定するようにバックエンドを設計する場合は、バックエンドファイルでパラメータを使用し `type`ます。管理者は、パラメータを使用して、ストレージに作成できるボリュームサイズを制限することもできます `limitVolumeSize`。現時点では、ボリュームサイズ変更やボリュームレプリケーションなどのElementストレージ機能は、ドライバを使用してサポートされていませ `solidfire-san`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	スナップショット	クローン	マルチアタッチ	CHAP（CHAP）	QoS	サイズ変更	レプリケーション
`solidfire-san`	はい	はい	Yesfootnote: 2[]	はい	はい	はい	Yesfootnote: 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 バックエンドドライバ

Astra Tridentはドライバを使用して `azure-netapp-files`サービスを管理し"Azure NetApp Files"ます。

このドライバとその設定方法の詳細については、を参照してください"Azure NetApp Files 向けの Trident バックエンド構成"。

Azure NetApp Files ドライバ	スナップショット	クローン	マルチアタッチ	QoS	展開表示	レプリケーション
`azure-netapp-files`	はい	はい	はい	はい	はい	Yesfootnote: 1[]

> 脚注：はい脚注： 1[] ： Astra Trident で管理されていません

Google Cloudバックエンドドライバ上のCloud Volumes Service

Astra Tridentでは、ドライバを使用して `gcp-cvs`Google CloudのCloud Volumes Serviceにリンクします。

> `gcp-cvs`ドライバは仮想プールを使用してバックエンドを抽象化し、`Astra Trident`でボリュームの配置を決定できるようにします。管理者がファイルに仮想プールを定義し`backend.json`ます。ストレージクラスには、ラベルで仮想プールを識別するセレクタが使用されます。

- バックエンドに仮想プールが定義されている場合、Astra Tridentは、その仮想プールが制限されているGoogle Cloudストレージプール内にボリュームを作成しようとします。
- バックエンドに仮想プールが定義されていない場合、Astra Tridentは、リージョン内の使用可能なストレージプールからGoogle Cloudストレージプールを選択します。

Astra TridentでGoogle Cloudバックエンドを設定するには、バックエンドファイルで、 `apiRegion`を`apiKey`指定する必要があります `projectNumber`。プロジェクト番号はGoogle Cloudコンソールで確認できます。APIキーは、Google CloudでCloud Volumes Service のAPIアクセスを設定するときに作成したサービスアカウントの秘密鍵ファイルから取得されます。

Google Cloudのサービスタイプとサービスレベルに関するCloud Volumes Serviceの詳細については、を参照してください"CVS for GCPのAstra Tridentサポートについてご確認ください"。

Cloud Volumes Service for Google Cloudドライバ	スナップショット	クローン	マルチアタッチ	QoS	展開表示	レプリケーション
`gcp-cvs`	はい	はい	はい	はい	はい	CVS -パフォーマンスサービスタイプでのみ利用できます。

レプリケーションに関する注意事項

- レプリケーションはAstra Tridentで管理されていません。
- クローンは、ソースボリュームと同じストレージプールに作成されます。

ストレージクラスの設計

Kubernetes ストレージクラスオブジェクトを作成するには、個々のストレージクラスを設定して適用する必要があります。このセクションでは、アプリケーション用のストレージクラスの設計方法について説明します。

特定のバックエンド使用率

フィルタリングは、特定のストレージクラスオブジェクト内で使用でき、そのストレージクラスで使用するストレージプールまたはプールのセットを決定します。ストレージクラスでは、、 `additionalStoragePools`、またはその両方の `excludeStoragePools`3セットのフィルタを設定できます `storagePools`。

パラメータを使用 `storagePools`すると、指定した属性に一致するプールだけにストレージを制限できます。`additionalStoragePools`パラメータを使用して、Astra Tridentでプロビジョニングに使用する一連のプールを、属性とパラメータで選択した一連のプールとともに拡張し `storagePools`ます。どちらか一方のパラメータを単独で使用することも、両方を使用して、適切なストレージプールセットが選択されていることを確認することもできます。

`excludeStoragePools`パラメータは、属性に一致するリストされた一連のプールを具体的に除外するために使用します。

QoSポリシーをエミュレートします

QoSポリシーをエミュレートするようにストレージクラスを設計する場合は、属性をまたは `ssd`にし `hdd`てストレージクラスを作成します `media`。ストレージクラスで指定された属性に基づいて `media`、Tridentはメディア属性に一致するサービスまたは `ssd`アグリゲートを提供する適切なバックエンドを選択し `hdd`、ボリュームのプロビジョニングを特定のアグリゲートに転送します。そのため、Premiumという属性が設定され `ssd`たストレージクラスを作成し `media`、Premium QoSポリシーに分類できるようにします。メディア属性を「 hdd 」に設定し、標準の QoS ポリシーとして分類できる、別のストレージクラス標準を作成できます。また、ストレージクラスの「 IOPS 」属性を使用して、 QoS ポリシーとして定義できる Element アプライアンスにプロビジョニングをリダイレクトすることもできます。

特定の機能に基づいてバックエンドを利用する

ストレージクラスは、シンプロビジョニングとシックプロビジョニング、 Snapshot 、クローン、暗号化などの機能が有効になっている特定のバックエンドでボリュームを直接プロビジョニングするように設計できます。使用するストレージを指定するには、必要な機能を有効にしてバックエンドに適したストレージクラスを作成します。

仮想プール

仮想プールはすべてのAstra Tridentバックエンドで利用可能Tridentが提供する任意のドライバを使用して、任意のバックエンドに仮想プールを定義できます。

仮想プールを使用すると、管理者はストレージクラスで参照可能なバックエンド上に抽象化レベルを作成して、バックエンドにボリュームを柔軟かつ効率的に配置できます。同じサービスクラスを使用して異なるバックエンドを定義できます。さらに、同じバックエンドに異なる特性を持つ複数のストレージプールを作成することもできます。セレクタで特定のラベルを設定したストレージクラスがある場合、 Astra Trident は、ボリュームを配置するすべてのセレクタラベルに一致するバックエンドを選択します。ストレージクラスセレクタのラベルが複数のストレージプールに一致した場合、Astra Tridentがボリュームのプロビジョニングに使用するストレージクラスを1つ選択します。

仮想プールの設計

バックエンドの作成時に、一般に一連のパラメータを指定できます。管理者が、同じストレージクレデンシャルと異なるパラメータセットを使用して別のバックエンドを作成することはできませんでした。仮想プールの導入により、この問題 は軽減されました。仮想プールは、バックエンドとKubernetesストレージクラスの間に導入されたレベル抽象化です。管理者は、Kubernetes Storage Classesでセレクターとして参照できるラベルとともにパラメータをバックエンドに依存しない方法で定義できます。仮想プールは、サポートされているすべてのネットアップバックエンドにAstra Tridentを使用して定義できます。リストには、 SolidFire / NetApp HCI 、 ONTAP 、 GCP 上の Cloud Volumes Service 、 Azure NetApp Files が含まれます。

> ⓘ 仮想プールを定義する場合は、バックエンド定義で既存の仮想プールの順序を変更しないことをお勧めします。また、既存の仮想プールの属性を編集または変更したり、新しい仮想プールを定義したりしないことを推奨します。

さまざまなサービスレベル/**QoS**のエミュレート

サービスクラスをエミュレートするための仮想プールを設計できます。Cloud Volume Service for Azure NetApp Files の仮想プール実装を使用して、さまざまなサービスクラスをセットアップする方法を見ていきましょう。Azure NetApp Filesバックエンドには、異なるパフォーマンスレベルを表す複数のラベルを設定します。アスペクトを適切なパフォーマンスレベルに設定し `servicelevel`、各ラベルの下にその他の必要なアスペクトを追加します。次に、異なる仮想プールにマッピングするさまざまなKubernetesストレージクラスを作成します。フィールドを使用して `parameters.selector`、各StorageClassはボリュームのホストに使用できる仮想プールを呼び出します。

特定の一連の側面を割り当てます

特定の側面を持つ複数の仮想プールは、単一のストレージバックエンドから設計できます。そのためには、バックエンドに複数のラベルを設定し、各ラベルに必要な側面を設定します。次に、異なる仮想プールにマッピングする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 `parameters.selector`ます。バックエンドでプロビジョニングされるボリュームには、選択した仮想プールに定義された設定が適用されます。

ストレージプロビジョニングに影響する **PVC** 特性

要求されたストレージクラスを超えたパラメータの中には、PVCを作成する際にAstra Tridentプロビジョニングの判断プロセスに影響するものがあります。

アクセスモード

PVC 経由でストレージを要求する場合、必須フィールドの 1 つがアクセスモードです。必要なモードは、ストレージ要求をホストするために選択されたバックエンドに影響を与える可能性があります。

Astra Trident は、次のマトリックスで指定されたアクセス方法で使用されているストレージプロトコルと一致するかどうかを試みます。これは、基盤となるストレージプラットフォームに依存しません。

	ReadWriteOnce コマンドを使用します	**ReadOnlyMany**	**ReadWriteMany**
iSCSI	はい	はい	○（ Raw ブロック）
NFS	はい	はい	はい

NFS バックエンドが設定されていない Trident 環境に送信された ReadWriteMany PVC が要求された場合、ボリュームはプロビジョニングされません。このため、リクエスタは、アプリケーションに適したアクセスモードを使用する必要があります。

ボリューム操作

永続ボリュームの変更

永続ボリュームとは、 Kubernetes で変更不可のオブジェクトを 2 つだけ除いてです。再利用ポリシーとサイズは、いったん作成されると変更できます。ただし、これにより、ボリュームの一部の要素がKubernetes以外で変更されることが防止されるわけではありません。特定のアプリケーション用にボリュームをカスタマイズしたり、誤って容量が消費されないようにしたり、何らかの理由でボリュームを別のストレージコントローラに移動したりする場合に便利です。

> 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。そのためには、PersistentVolumeClaimを作成し、ボリュームの作成元となるSnapshotとしてを指定します `datasource`。Astra Trident がこの PVC を処理するには、 Snapshot にデータが存在するボリュームを作成します。この機能を使用すると、複数のリージョン間でデータを複製したり、テスト環境を作成したり、破損

した本番ボリューム全体を交換したり、特定のファイルとディレクトリを取得して別の接続ボリュームに転送したりできます。

クラスタ内でボリュームを移動します

ストレージ管理者は、 ONTAP クラスタ内のアグリゲート間およびコントローラ間で、ストレージ利用者への無停止でボリュームを移動できます。この処理は、デスティネーションアグリゲートが Trident が使用している SVM からアクセス可能なアグリゲートであるかぎり、 Astra Trident または Kubernetes クラスタには影響しません。この点が重要なのは、アグリゲートが SVM に新たに追加された場合、 Astra Trident に再追加してバックエンドを更新する必要があることです。これにより、 Astra Trident が SVM のインベントリを再作成し、新しいアグリゲートが認識されるようになります。

ただし、バックエンド間でのボリュームの移動は Astra Trident では自動ではサポートされていません。これには、同じクラスタ内の SVM 間、クラスタ間、または別のストレージプラットフォーム上の SVM 間が含まれます（たとえストレージシステムが Trident から Astra に接続されている場合でも）。

ボリュームが別の場所にコピーされた場合、ボリュームインポート機能を使用して現在のボリュームを Astra Trident にインポートできます。

ボリュームを展開します

Astra Trident は、 NFS と iSCSI PVS のサイズ変更をサポートしています。これにより、ユーザは Kubernetes レイヤを介してボリュームのサイズを直接変更できます。ボリュームを拡張できるのは、 ONTAP 、 SolidFire / NetApp HCI 、 Cloud Volumes Service バックエンドなど、主要なすべてのネットアップストレージプラットフォームです。あとで拡張できるようにするには、ボリュームに関連付けられているStorageClassでをに `true`設定し `allowVolumeExpansion`ます。永続的ボリュームのサイズを変更する必要がある場合は、永続的ボリューム要求で必要なボリュームサイズになるようにアノテーションを編集します `spec.resources.requests.storage`。Tridentによって、ストレージクラスタ上のボリュームのサイズが自動的に変更されます。

既存のボリュームを **Kubernetes** にインポートする

Volume Import では、既存のストレージボリュームを Kubernetes 環境にインポートできます。これは、現在、、 `ontap-nas-flexgroup` `solidfire-san`、、 `azure-netapp-files`および `gcp-cvs`ドライバでサポートされて `ontap-nas`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とドライバを使用する場合 `solidfire-san`は、コマンドを使用し `tridentctl import volume <backend-name> <volume-name> -f /path/pvc.yaml`て既存のボリュームをKubernetesにインポートし、Astra Tridentで管理します。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。

ドライバまたは `gcp-cvs`ドライバを使用している場合 `azure-netapp-files`は、コマンドを使用し `tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml`て、Astra Tridentで管理する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を自動的に追加（および必要に応じて上書き）します。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"netapp.io のコマンドです"て"ブログ"います。

ロギングサービス

他のOpenShiftサービスと同様に、ロギングサービスは、Playbookに提供されるインベントリファイル（ホスト）から提供される設定パラメータを使用してAnsibleを使用して導入されます。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 NFSサーバにはこのような問題がないため、ロギング環境を簡単にバックアップできます。ロギングサービスには最終的にどちらかのプロトコルを選択する必要がありますが、両方のプロトコルがネットアッププラットフォームを使用する場合に適していることと、 NFS を使用する理由がないことを確認してください。

ログサービスでNFSを使用する場合は、インストーラが失敗しないように `true`Ansible変数を設定する必要があり `openshift_enable_unsupported_configurations`ます。

開始する

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の展開を選択した場合は、変数をに `true`指定する `openshift_logging_use_ops`と、サービスの2つのインスタンスが作成されます。操作のロギングインスタンスを制御する変数には「 ops 」が含まれ、アプリケーションのインスタンスには含まれません。

基盤となるサービスで正しいストレージが使用されるようにするには、導入方法に応じてAnsible変数を設定することが重要です。それぞれの導入方法のオプションを見てみましょう。

次の表には、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を示します。展開に応じて、レビュー、設定、および使用する必要がある他のオプションを見つけることができます"Red Hat OpenShift のロギングに関するドキュメント"。

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	詳細
openshift_logging_storage_kind	インストーラによってロギングサービス用のNFS PVが作成されるようにするには、をに設定し `nfs`ます。
openshift_logging_storage_host	NFS ホストのホスト名または IP アドレス。仮想マシンのデータ LIF に設定してください。
openshift_logging_storage_nfs_directory	NFS エクスポートのマウントパス。たとえば、ボリュームがとしてジャンクションされている場合 `/openshift_logging`は、そのパスを変数に使用します。
openshift_logging_storage_volume_name	作成するPVの名前（例： pv_ose_logs）。
openshift_logging_storage_volume_size	NFSエクスポートのサイズ（例：） 100Gi。

OpenShift クラスタがすでに実行中で、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	詳細
openshift_logging_es_pvc_dynamic	動的にプロビジョニングされたボリュームを使用する場合は true に設定します。
openshift_logging_es_pvc_storage_class_name	PVC で使用されるストレージクラスの名前。
openshift_logging_es_pvc_size	PVC で要求されたボリュームのサイズ。
openshift_logging_es_pvc_prefix	ロギングサービスで使用される PVC のプレフィックス。
openshift_logging_es_ops_pvc_dynamic	opsロギングインスタンスに動的にプロビジョニングされたボリュームを使用するには、をに設定し `true`ます。
openshift_logging_es_ops_pvc_storage_class_name	処理ロギングインスタンスのストレージクラスの名前。
openshift_logging_es_ops_pvc_size	処理インスタンスのボリューム要求のサイズ。
openshift_logging_es_ops_pvc_prefix	ops インスタンス PVC のプレフィックス。

ロギングスタックを導入します

初期の OpenShift インストールプロセスの一部としてロギングを導入する場合、標準の導入プロセスに従うだけで済みます。Ansible は、必要なサービスと OpenShift オブジェクトを構成および導入して、 Ansible が完了したらすぐにサービスを利用できるようにします。

ただし、最初のインストール後に導入する場合は、コンポーネントプレイブックを Ansible で使用する必要があります。このプロセスは、OpenShiftのバージョンによって若干変更される場合がありますので、お使いのバージョンに合わせてお読みください"Red Hat OpenShift Container Platform 3.11 のドキュメント"。

指標サービス

この指標サービスは、 OpenShift クラスタのステータス、リソース利用率、可用性に関する重要な情報を管理

者に提供します。ポッドの自動拡張機能にも必要であり、多くの組織では、チャージバックやショーバックのアプリケーションに指標サービスのデータを使用しています。

ロギングサービスや OpenShift 全体と同様に、 Ansible を使用して指標サービスを導入します。また、ロギングサービスと同様に、メトリクスサービスは、クラスタの初期セットアップ中、またはコンポーネントのインストール方法を使用して運用後に導入できます。次の表に、指標サービスに永続的ストレージを設定する際に重要となる変数を示します。

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	詳細
`openshift_metrics_storage_kind`	インストーラによってロギングサービス用のNFS PVが作成されるようにするには、をに設定し `nfs`ます。
`openshift_metrics_storage_host`	NFS ホストのホスト名または IP アドレス。これは SVM のデータ LIF に設定されている必要があります。
`openshift_metrics_storage_nfs_directory`	NFS エクスポートのマウントパス。たとえば、ボリュームがとしてジャンクションされている場合 `/openshift_metrics`は、そのパスを変数に使用します。
`openshift_metrics_storage_volume_name`	作成するPVの名前（例： `pv_ose_metrics`）。
`openshift_metrics_storage_volume_size`	NFSエクスポートのサイズ（例：） `100Gi`。

OpenShift クラスタがすでに実行中で、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	詳細
`openshift_metrics_cassandra_pvc_prefix`	メトリック PVC に使用するプレフィックス。
`openshift_metrics_cassandra_pvc_size`	要求するボリュームのサイズ。
`openshift_metrics_cassandra_storage_type`	指標に使用するストレージのタイプ。適切なストレージクラスを使用して PVC を作成するには、 Ansible に対してこれを dynamic に設定する必要があります。
`openshift_metrics_cassanda_pvc_storage_class_name`	使用するストレージクラスの名前。

指標サービスを導入する

ホスト / インベントリファイルに適切な Ansible 変数を定義して、 Ansible でサービスを導入します。OpenShift インストール時に導入する場合は、 PV が自動的に作成されて使用されます。コンポーネントプレイブックを使用して導入する場合は、OpenShiftのインストール後にAnsibleによって必要なPVCが作成され、Astra Tridentによってストレージがプロビジョニングされたあとにサービスが導入されます。

上記の変数と導入プロセスは、 OpenShift の各バージョンで変更される可能性があります。使用しているバージョンを確認し、環境に合わせて構成されるようにして"RedHat OpenShift 導入ガイド"ください。

データ保護とディザスタリカバリ

Astra TridentとAstra Tridentを使用して作成されたボリュームの保護とリカバリのオプションについて説明します。永続性に関する要件があるアプリケーションごとに、データ保護とリカバリの戦略を用意しておく必要があります。

Astra Tridentのレプリケーションとリカバリ

災害発生時にAstra Tridentをリストアするバックアップを作成できます。

Astra Tridentのレプリケーション

Astra Tridentは、Kubernetes CRDを使用して独自の状態の格納と管理を行い、Kubernetesクラスタetcdを使用してメタデータを格納します。

手順

1. を使用してKubernetesクラスタetcdをバックアップし"Kubernetes：etcdクラスタのバックアップ"ます。
2. バックアップアーティファクトをFlexVolに配置します。

> FlexVolが配置されているSVMを別のSVMへのSnapMirror関係で保護することを推奨します。

Astra Tridentのリカバリ

Kubernetes CRDとKubernetesクラスタetcd Snapshotを使用して、Astra Tridentをリカバリできます。

手順

1. デスティネーションSVMから、Kubernetes etcdデータファイルと証明書が格納されているボリュームを、マスターノードとしてセットアップするホストにマウントします。
2. Kubernetesクラスタに関連する必要なすべての証明書をにコピーし、etcdメンバーファイルを `/var/lib/etcd`にコピーします `/etc/kubernetes/pki`。
3. を使用して、etcdバックアップからKubernetesクラスタをリストアします"Kubernetes：etcdクラスタのリストア"。
4. を実行し `kubectl get crd`てすべてのTridentカスタムリソースが稼働していることを確認し、Tridentオブジェクトを取得してすべてのデータが使用可能であることを確認します。

SVMレプリケーションとリカバリ

Astra Tridentではレプリケーション関係を設定できませんが、ストレージ管理者はを使用してSVMをレプリケートできます "ONTAP SnapMirror"。

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。

タスクの内容

SnapMirror SVMレプリケーション機能を使用する場合は、次の点を考慮してください。

- SVM-DRを有効にしたSVMごとに、個別のバックエンドを作成する必要があります。
- SVM-DRをサポートするバックエンドにレプリケーション不要のボリュームをプロビジョニングしないように、必要な場合にのみレプリケートされたバックエンドを選択するようにストレージクラスを設定します。
- アプリケーション管理者は、レプリケーションに伴う追加コストと複雑さを理解し、このプロセスを開始する前にリカバリプランを慎重に検討する必要があります。

SVMレプリケーション

を使用すると、SVMレプリケーション関係を作成できます"ONTAP：SnapMirror SVMレプリケーション"。

SnapMirrorでは、レプリケートする対象を制御するオプションを設定できます。プリフォーム時に選択したオプションを知っておく必要がAstra Tridentを使用したSVMのリカバリあります。

- "-identity-preserve true"SVMの設定全体をレプリケートします。
- "-discard-configs network"LIFと関連ネットワークの設定を除外します。
- "-identity-preserve false"ボリュームとセキュリティ設定のみをレプリケートします。

Astra Tridentを使用した**SVM**のリカバリ

Astra Trident では、 SVM の障害は自動では検出されない。災害が発生した場合、管理者は新しいSVMへのTridentフェイルオーバーを手動で開始できます。

手順

1. スケジュールされた実行中のSnapMirror転送をキャンセルし、レプリケーション関係を解除し、ソースSVMを停止してからSnapMirrorデスティネーションSVMをアクティブ化します。
2. を指定した場合は `-identity-preserve false`、 `-discard-config network`SVMレプリケーションの設定時に、`Trident`バックエンド定義ファイルでと `dataLIF`を更新します `managementLIF`。
3. Tridentバックエンド定義ファイルにが存在することを確認します `storagePrefix`。このパラメータは変更できません。省略する `storagePrefix`と、バックエンドの更新が失敗します。
4. 次のコマンドを使用して、必要なすべてのバックエンドを更新して新しいデスティネーションSVM名を反映します。

```
./tridentctl update backend <backend-name> -f <backend-json-file> -n
<namespace>
```

5. または `discard-config network`を指定した場合は `-identity-preserve false`、すべてのアプリケーションポッドをバウンスする必要があります。

> を指定した場合、 `-identity-preserve true`デスティネーションSVMがアクティブ化されると、Astra Tridentでプロビジョニングされたすべてのボリュームがデータの提供を開始します。

ボリュームのレプリケーションとリカバリ

Astra TridentではSnapMirrorレプリケーション関係を設定できませんが、ストレージ管理者はAstra Tridentで作成されたボリュームをレプリケートするためにを使用できます"ONTAPのSnapMirrorレプリケーションとリカバリ"。

リカバリしたボリュームは、を使用してAstra Tridentにインポートできます"tridentctlボリュームインポート"。

> インポートは、、 `ontap-san-economy`、またはの `ontap-flexgroup-economy`ドライバではサポートされていません `ontap-nas-economy`。

Snapshotによるデータ保護

次のコマンドを使用してデータを保護およびリストアできます。

- 永続ボリューム（PV）のKubernetesボリュームSnapshotを作成するための外部のSnapshotコントローラとCRD。

  "ボリューム Snapshot"

- ONTAP Snapshot：ボリュームの内容全体のリストア、または個 々 のファイルまたはLUNのリカバリに使用します。

  "ONTAPスナップショット"

Astra Control Centerアプリケーションのレプリケーション

Astra Controlを使用すると、SnapMirrorの非同期レプリケーション機能を使用して、データやアプリケーションの変更をクラスタ間でレプリケートできます。

"Astra Control：SnapMirrorテクノロジを使用してアプリケーションをリモートシステムにレプリケート"

セキュリティ

セキュリティ

ここに記載された推奨事項を参考に、Astra Tridentのインストールを安全に行ってください。

Astra Trident を独自のネームスペースで実行

アプリケーション、アプリケーション管理者、ユーザ、および管理アプリケーションが Astra Trident オブジェクト定義またはポッドにアクセスしないようにして、信頼性の高いストレージを確保し、悪意のあるアクティビティをブロックすることが重要です。

他のアプリケーションやユーザをAstra Tridentから分離するには、必ず独自のKubernetesネームスペースにAstra Tridentをインストールして(`trident`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は管理者のみに許可し、アプリケーションへのアクセスを許可する必要があります `tridentctl。

ONTAP SAN バックエンドで CHAP 認証を使用します

Astra Tridentでは、ONTAP SANワークロードのCHAPベースの認証がサポートされます（ドライバと ontap-san-economy`ドライバを使用 `ontap-san）。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は、Astra Tridentで双方向CHAPを設定し、でCHAPのユーザ名とシークレットを管理できます tridentctl。Astra TridentがONTAPバックエンドでCHAPを設定する方法については、を参照してください""。

NetApp HCI および SolidFire バックエンドで CHAP 認証を使用します

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。インストールされたAstra Tridentは、CHAPシークレットを管理し、それぞれのPVのCRオブジェクトに格納し `tridentvolume`ます。PVを作成すると、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。

> Astra Tridentで作成されるボリュームは、どのボリュームアクセスグループにも関連付けられません。

NVEおよびNAEでAstra Tridentを使用する

NetApp ONTAP は、保管データの暗号化を提供し、ディスクが盗難、返却、転用された場合に機密データを保護します。詳細については、を参照してください "NetAppボリューム暗号化の設定の概要"。

- NAEがバックエンドで有効になっている場合は、Astra TridentでプロビジョニングされたすべてのボリュームがNAEに対応します。
- バックエンドでNAEが有効になっていない場合、バックエンド構成でNVE暗号化フラグをに設定しないかぎり、Astra Tridentでプロビジョニングされたボリュームは NVE対応になり `false`ます。

> NAE対応バックエンドのAstra Tridentで作成されるボリュームは、NVEまたはNAEで暗号化されている必要があります。
>
> - Tridentバックエンド構成でNVE暗号化フラグをに設定すると、NAE暗号化を無効にして、ボリューム単位で特定の暗号化キーを使用でき `true`ます。
> - NAE対応バックエンドでNVE暗号化フラグをに設定する false`と、NAE対応ボリュームが作成されます。NVE暗号化フラグをに設定してNAE暗号化を無効にすることはできません `false。

- Astra TridentでNVEボリュームを手動で作成するには、NVE暗号化フラグを明示的にに設定し `true`ます。

バックエンド構成オプションの詳細については、以下を参照してください。

- "ONTAP SANの構成オプション"

- "ONTAP NASの構成オプション"

Linux Unified Key Setup（LUKS；統合キーセットアップ）

Linux Unified Key Setup（LUKS；ユニファイドキーセットアップ）を有効にして、Astra Trident上のONTAP SANおよびONTAP SANエコノミーボリュームを暗号化できます。Astra Tridentは、LUKS暗号化ボリュームのパスフレーズローテーションとボリューム拡張をサポートしています。

Astra Tridentでは、LUKS暗号化ボリュームでAES-XTS-plain64暗号化とモードが使用されます（の推奨）"NIST"。

開始する前に

- ワーカーノードにはcryptsetup 2.1以上（3.0よりも下位）がインストールされている必要があります。詳細については、を参照してください"Gitlab: cryptsetup"。
- パフォーマンス上の理由から、ワーカーノードでAdvanced Encryption Standard New Instructions（AES-NI）をサポートすることを推奨します。AES-NIサポートを確認するには、次のコマンドを実行します。

```
grep "aes" /proc/cpuinfo
```

何も返されない場合、お使いのプロセッサはAES-NIをサポートしていません。AES-NIの詳細については、を参照してください"Intel：Advanced Encryption Standard Instructions（AES-NI）"。

LUKS暗号化を有効にします

ONTAP SANおよびONTAP SANエコノミーボリュームでは、Linux Unified Key Setup（LUKS；Linux統合キーセットアップ）を使用して、ボリューム単位のホスト側暗号化を有効にできます。

手順

1. バックエンド構成でLUKS暗号化属性を定義します。ONTAP SANのバックエンド構成オプションの詳細については、を参照してください"ONTAP SANの構成オプション"。

```
"storage": [
    {
        "labels":{"luks": "true"},
        "zone":"us_east_1a",
        "defaults": {
            "luksEncryption": "true"
        }
    },
    {
        "labels":{"luks": "false"},
        "zone":"us_east_1a",
        "defaults": {
            "luksEncryption": "false"
        }
    },
]
```

2. LUKS暗号化を使用してストレージプールを定義する場合に使用し `parameters.selector`ます。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: csi.trident.netapp.io
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

3. LUKSパスフレーズを含むシークレットを作成します。例：

```
kubectl -n trident create -f luks-pvc1.yaml
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: A
  luks-passphrase: secretA
```

制限事項

LUKSで暗号化されたボリュームは、ONTAP の重複排除と圧縮を利用できません。

LUKSボリュームをインポートするためのバックエンド構成

LUKSボリュームをインポートするには、バックエンドでをに(`true`設定する必要があります `luksEncryption`。このオプションを指定 `luksEncryption`すると、(`false`次の例に示すように、ボリュームがLUKS準拠(`true`かどうかがAstra Tridentに通知されます。

```
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: trident_svm
username: admin
password: password
defaults:
  luksEncryption: 'true'
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'
```

LUKSボリュームをインポートするためのPVC設定

LUKSボリュームを動的にインポートするには、 `trident.netapp.io/luksEncryption` `true`次の例に示すように、アノテーションをに設定し、LUKS対応のストレージクラスをPVCに含めます。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: luks-pvc
  namespace: trident
  annotations:
    trident.netapp.io/luksEncryption: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: luks-sc
```

LUKSパスフレーズをローテーションします

LUKSのパスフレーズをローテーションしてローテーションを確認できます。

パスフレーズは、ボリューム、Snapshot、シークレットで参照されなくなることを確認するまで忘れないでください。参照されているパスフレーズが失われた場合、ボリュームをマウントできず、データが暗号化されたままアクセスできなくなることがあります。

タスクの内容

LUKSパスフレーズのローテーションは、ボリュームをマウントするポッドが、新しいLUKSパスフレーズの指定後に作成されたときに行われます。新しいポッドが作成されると、Astra TridentはボリュームのLUKSパスフレーズをシークレット内のアクティブなパスフレーズと比較します。

- ボリュームのパスフレーズがシークレットでアクティブなパスフレーズと一致しない場合、ローテーションが実行されます。
- ボリュームのパスフレーズがシークレット内のアクティブなパスフレーズと一致する場合、 `previous-luks-passphrase`パラメータは無視されます。

手順

1. および `node-publish-secret-namespace`StorageClassパラメータを追加します `node-publish-secret-name`。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-san
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/backendType: "ontap-san"
  csi.storage.k8s.io/node-stage-secret-name: luks
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-publish-secret-name: luks
  csi.storage.k8s.io/node-publish-secret-namespace: ${pvc.namespace}
```

2. ボリュームまたはSnapshotの既存のパスフレーズを特定します。

ボリューム

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["A"]
```

Snapshot

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["A"]
```

3. ボリュームのLUKSシークレットを更新して、新しいパスフレーズと前のパスフレーズを指定します。previous-luks-passphrase`前のパスフレーズと一致することを確認します `previous-luke-passphrase-name。

```
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: B
  luks-passphrase: secretB
  previous-luks-passphrase-name: A
  previous-luks-passphrase: secretA
```

4. ボリュームをマウントする新しいポッドを作成します。これはローテーションを開始するために必要です。
5. パスフレーズがローテーションされたことを確認します。

ボリューム

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["B"]
```

Snapshot

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["B"]
```

結果

パスフレーズは、ボリュームとSnapshotに新しいパスフレーズのみが返されたときにローテーションされました。

> たとえば、2つのパスフレーズが返された場合、 `luksPassphraseNames: ["B", "A"]`ローテーションは不完全です。回転を完了するために、新しいポッドをトリガできます。

ボリュームの拡張を有効にします

LUKS暗号化ボリューム上でボリューム拡張を有効にできます。

手順

1. 機能ゲート（ベータ1.25以降）を有効にします CSINodeExpandSecret。詳細については、を参照して

ください "Kubernetes 1.25：CSIボリュームのノードベースの拡張にシークレットを使用します" 。

2. および `node-expand-secret-namespace`StorageClassパラメータを追加します `node-expand-secret-name。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: csi.trident.netapp.io
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-expand-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-expand-secret-namespace: ${pvc.namespace}
allowVolumeExpansion: true
```

結果

ストレージのオンライン拡張を開始すると、ドライバに適切なクレデンシャルが渡されます。

知識とサポート

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。

24.02リリース以降、Astra Tridentは2月、6月、10月の4カ月ごとにリリースされます。

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。

使用して `tridentctl get`コマンドを、Astra Trident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。Prometheusエンドポイント：管理対象のバックエンド数、プロビジョニングされたボリューム数、消費されたバイト数など、Astra Tridentの処理に関する情報を収集できます。を監視および分析に使用することもできます"Cloud Insights"。

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。ユーザーエクスペリエンスと機能に関しては変更はありません。使用されるプロビジョニングツール名はです `csi.trident.netapp.io`。現在および将来のリリースで提供される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

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 "Astra Tridentのインストール方法をご確認ください"。

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、クラスタにアクセスできる任意のマシンからのリモートインストール機能がサポートされます `kubectl`。アクセスが確認されたら `kubectl`（たとえば、リモートマシンからコマンドを実行し `kubectl get nodes`て確認する）、インストール手順に従います。

Astra Trident でハイアベイラビリティを構成できますか。

Astra Tridentは、インスタンスが1つのKubernetesデプロイメント（ReplicaSet）としてインストールされるため、HAが組み込まれています。デプロイメント内のレプリカの数を増やすことはできません。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インストーラによってKubernetes ClusterRoleが作成され、KubernetesクラスタのPersistentVolume、PersistentVolumeClaim、StorageClass、およびSecretリソースに特定のアクセス権が付与されます。を参照してください "tridentctlのインストールをカスタマイズします"。

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 "tridentctlのインストールをカスタマイズします"。

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、ファイルで一意のパラメータを `setup/backend.json`指定し `StoragePrefix`ます。これは、両方のインスタンスで同じ 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. に連絡してサポートケースを提起し "NetAppのサポート"ます。

サポートログバンドルを生成するにはどうすればよいですか？

を実行すると、サポートバンドルを作成できます `tridentctl logs -a`。バンドルでキャプチャされたログに加えて、 kubelet ログをキャプチャして、 Kubernetes 側のマウントの問題を診断します。kubelet ログの取得手順は、 Kubernetes のインストール方法によって異なります。

新しい機能のリクエストを発行する必要がある場合は、どうすればよいですか。

問題を作成し "Astra Trident Github"、問題の件名と説明に*RFE*を記載します。

不具合を発生させる場所

で問題を作成し "Astra Trident Github"ます。問題に関連する必要なすべての情報とログを記録しておいてください。

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、リクエストがある場合は、AstraまたはGitHubからお問い合わせ"チャネルを外します"ください。

ストレージシステムのパスワードが変更され、**Astra Trident**が機能しなくなったため、どのようにリカバリすればよいですか？

バックエンドのパスワードをで更新し `tridentctl update backend myBackend -f </path/to_new_backend.json> -n trident`ます。この例では、をバックエンド名と `/path/to_new_backend.json`正しいファイルへのパスに `backend.json`置き換えます `myBackend`。

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。

アップグレード、依存関係の問題、またはアップグレードの失敗または不完全な実行後に見つかったバグの修正が必要な場合は、そのバージョンに固有の手順を使用して以前のバージョンを再インストールする必要があります"Astra Tridentをアンインストールします"。これは、以前のバージョンにダウングレードするための唯一の推奨方法です。

バックエンドとボリュームを管理

ONTAP バックエンド定義ファイルに管理 **LIF** とデータ **LIF** の両方を定義する必要がありますか。

管理LIFは必須です。データLIFのタイプはさまざまです。

- ONTAP SAN：iSCSIには指定しないでください。Astra Tridentは、を使用して"ONTAP の選択的LUNマップ"、マルチパスセッションの確立に必要なiSCI LIFを検出します。が明示的に定義されている場合は、警告が生成され `dataLIF`ます。詳細については、を参照してください "ONTAP SANの設定オプションと例"。
- ONTAP NAS:を指定することを推奨します `dataLIF`。指定しない場合は、Astra TridentがSVMからデータLIFを取得します。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。詳細は、を参照してください。"ONTAP NASの設定オプションと例"

Astra Trident が **ONTAP** バックエンドに **CHAP** を設定できるか。

はい。Astra Tridentでは、ONTAPバックエンドの双方向CHAPがサポートされます。これには、バックエンド構成での設定が必要です `useCHAP=true`。

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。

管理 **LIF** とデータ **LIF** に **IPv6** アドレスを使用できますか。

Astra Tridentでは、次の機能に対してIPv6アドレスを定義できます。

- managementLIF`また `dataLIF、ONTAP NASバックエンドにも対応しています。
- managementLIF`ONTAP SANバックエンドの場合。ONTAP SANバックエンドでは指定できません `dataLIF。

Astra TridentをIPv6で機能させるには、フラグ（インストール用 tridentctl）、（Trident Operator用）、IPv6`または（Helmインストール用） `tridentTPv6`を使用してインストールする必要があります `--use-ipv6。

バックエンドの管理 **LIF** を更新できますか。

はい、コマンドを使用してバックエンドの管理LIFを更新できます tridentctl update backend。

バックエンドのデータ **LIF** を更新できるか。

データLIFの更新は、および ontap-nas-economy`でのみ実行できます `ontap-nas。

Kubernetes 向け **Astra Trident** で複数のバックエンドを作成できますか。

Astra Trident では、同じドライバまたは別々のドライバを使用して、多数のバックエンドを同時にサポートできます。

Astra Trident はバックエンドクレデンシャルをどのように保存しますか。

Astra Trident では、バックエンドのクレデンシャルを Kubernetes のシークレットとして格納します。

Astra Trident ではどのようにして特定のバックエンドを選択しますか。

バックエンド属性を使用してクラスに適したプールを自動的に選択できない場合は storagePools、および `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。

パラメータを `excludeStoragePools`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、に一致するプールをすべて削除します。

同じ種類のバックエンドが複数ある場合、 **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。

同じタイプの設定済みバックエンドが複数ある場合は、との `PersistentVolumeClaim`パラメータに基づいて適切なバックエンドがAstra Tridentによって選択され `StorageClass`ます。たとえば、ONTAPとNASのドライババックエンドが複数ある場合、Astra Tridentは、と `PersistentVolumeClaim`を組み合わせてのパラメータを照合し、と `PersistentVolumeClaim`に記載されている要件を満たすバックエンドを `StorageClass`照合し `StorageClass`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-economy`ドライバは、同じFlexVolに最大200個のqtree（50~300の間で設定可能）、クラスタノードあたり100,000個、クラスタあたり2.4M個のqtreeを作成します。エコノミードライバによって処理される新しいを入力すると、`PersistentVolumeClaim`新しいqtreeに対応できるFlexVolがすでに存在するかどうかが確認されます。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、次の例を参照して"ここをクリック"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で設定されているパラメータを使用し `exportPolicy`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。詳細については、以下を参照してください。"Astra TridentとNVEおよびNAEの相互運用性"

Trident 経由で **ONTAP** に **QoS** を実装するには、どのような方法が最適ですか。

ONTAPのQoSを実装するために使用し `StorageClasses`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を設定する必要があります StorageClass。両方が設定されている場合は、が `StorageClass`優先されます。ONTAP で次の項目を設定します。

1. で StorageClass、属性をthickに設定し `provisioningType`ます。
2. バックエンド定義ファイルで、をvolumeに設定してシックボリュームを有効にします backend spaceReserve parameter。

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。

ボリュームが **SnapMirror** データ保護（ **DP** ）モードまたはオフラインモードの間にインポートできますか。

外部ボリュームが DP モードになっているかオフラインになっている場合、ボリュームのインポートは失敗します。次のエラーメッセージが表示されます。

```
Error: could not import volume: volume import failed to get size of
volume: volume <name> was not found (400 Bad Request) command terminated
with exit code 1.
Make sure to remove the DP mode or put the volume online before importing
the volume.
```

リソースクォータをネットアップクラスタに変換する方法

Kubernetes ストレージリソースクォータは、ネットアップストレージの容量があるかぎり機能します。容量不足が原因で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、フィーチャーゲートが有効になっていることを確認し `VolumeSnapshotDataSource`ます。

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。

現時点では `ontap-nas`、、 `ontap-nas-flexgroup` `ontap-san`、 `ontap-san-economy` `solidfire-san`、、 `gcp-cvs`、および `azure-netapp-files`バックエンドドライバ。

ONTAP を使用して **Astra Trident** でプロビジョニングしたボリュームの **Snapshot** バックアップを作成する方法を教えてください。

これは、、 `ontap-san`、および `ontap-nas-flexgroup`ドライバで使用でき `ontap-nas`ます。ドライバの `ontap-san-economy`を`FlexVol`レベルで指定することもできます `snapshotPolicy`。

これはドライバでも使用できますが、qtreeレベルではなく、FlexVolレベルで使用でき `ontap-nas-economy`ます。Astra Tridentでプロビジョニングされたボリュームの Snapshotを作成できるようにするには、バックエンドパラメータオプションをONTAPバックエンドで定義されている目的のSnapshotポリシーに設定し `snapshotPolicy`ます。ストレージコントローラで作成された Snapshot は 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属性を設定することで、Astra TridentでSnapshotコピーの格納用に特定の割合のディスクスペースをリザーブできます `snapshotReserve`。を設定し、 `snapshotReserve`バックエンド定義ファイルでスナップショット予約の割合が設定されている場合は `snapshotPolicy`、バックエンドファイルで指定されている割合に従って設定され `snapshotReserve`ます。パーセンテージ番号が指定されていない場合 `snapshotReserve`、ONTAPはデフォルトでスナップショット予約のパーセンテージを5とします。この `snapshotPolicy`オプションをnoneに設定すると、Snapshotリザーブの割合は0に設定されます。

ボリュームの **Snapshot** ディレクトリに直接アクセスしてファイルをコピーできますか。

はい。Tridentによってプロビジョニングされるボリューム上のsnapshotディレクトリには、バックエンド定義ファイルでパラメータを設定することでアクセスできます `snapshotDir`。

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。

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。

全般的なトラブルシューティング

- Tridentポッドが適切に起動しない場合（たとえば、Tridentポッドが使用可能なコンテナが3つ未満でフェーズで停止した `ContainerCreating`場合）は、実行中で `kubectl -n trident describe deployment trident`、追加の分析情報が得られます。 `kubectl -n trident describe pod trident--**`kubeletログ（たとえば、経由）を取得すること `journalctl -xeu kubelet`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installパラメータにフラグを渡して、Tridentのデバッグモードを有効にしてみて `-d`ください。

  次に、を使用してデバッグが設定されていることを確認し、 `./tridentctl logs -n trident`ログでを検索し `level=debug msg`ます。

  オペレータとともにインストールされます

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

  すべての Trident ポッドが再起動されます。これには数秒かかることがあります。これを確認するには、の出力にある「age」列を確認し `kubectl get pod -n trident`ます。

  Astra Trident 20.07および20.10では、の代わりに `torc`使用します `tprov`。

  Helm とともにインストールされます

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

  tridentctl を使用してインストールされます

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

- バックエンド定義にを含めることで、各バックエンドのデバッグログを取得することもできます `debugTraceFlags`。たとえば、TridentログでAPI呼び出しとメソッドトラバーサルを取得するためにを指定します `debugTraceFlags: {“api”:true, “method”:true,}`。既存のバックエンドはで構成 `tridentctl backend update`できます `debugTraceFlags`。
- RedHat CoreOSを使用している場合は、ワーカーノードでが有効になっていて、デフォルトで起動されていることを確認して `iscsid`ください。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。
- でTridentを使用するときによく発生する問題 "Azure NetApp Files"は、権限が不十分なアプリケーション登録にテナントシークレットとクライアントシークレットが含まれている場合です。Tridentの要件の一覧については、構成を参照して"Azure NetApp Files"ください。
- コンテナへのPVのマウントに問題がある場合は、がインストールされて実行されていることを確認して `rpcbind`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、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、または同等のを実行して `systemctl status rpcbind`確認できます `rpcbind`。

- 以前は機能していたにもかかわらずTridentバックエンドが状態であると報告された場合は `failed`、バックエンドに関連付けられているSVM /管理者クレデンシャルの変更が原因である可能性があります。Tridentポッドを使用してバックエンド情報を更新 `tridentctl update backend`またはバウンスすると、この問題が修正されます。
- コンテナランタイムとしてDockerを使用してTridentをインストールするときに権限の問題が発生した場合は、フラグを指定してTridentのインストールを試行し `--in cluster=false`ます。これはインストーラポッドを使用せず、ユーザーによるアクセス許可の問題を回避します `trident-installer`。
- を使用して、 `uninstall parameter <Uninstalling Trident>`実行に失敗した後のクリーンアップを実行します。デフォルトでは、スクリプトは Trident によって作成された CRD を削除しないため、実行中の導入環境でも安全にアンインストールしてインストールできます。
- 以前のバージョンのTridentにダウングレードする場合は、最初にコマンドを実行し `tridentctl uninstall`てTridentを削除します。コマンドを使用して目的のものをダウンロードし "Trident のバージョン"てインストールし `tridentctl install`ます。
- インストールが正常に完了した後、PVCがフェーズで停止した場合、 `Pending`を実行すると、 `kubectl describe pvc`TridentがこのPVCのPVのプロビジョニングに失敗した理由に関する追加情報が提供されます。

オペレータを使用したTridentの導入に失敗

オペレータを使用してTridentを展開する場合は、のステータス `TridentOrchestrator`がからに `Installed`変わります `Installing`。ステータスを確認し、オペレータが単独で回復できない場合は `Failed`、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、ステータスを確認する必要があり `TridentOrchestrator`ます。

```
kubectl describe torc trident-2
Name:         trident-2
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  trident.netapp.io/v1
Kind:         TridentOrchestrator
...
Status:
  Current Installation Params:
    IPv6:
    Autosupport Hostname:
    Autosupport Image:
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:
    Image Pull Secrets:         <nil>
    Image Registry:
    k8sTimeout:
    Kubelet Dir:
    Log Format:
    Silence Autosupport:
    Trident Image:
  Message:                      Trident is bound to another CR 'trident'
  Namespace:                    trident-2
  Status:                       Error
  Version:
Events:
  Type     Reason  Age                From                          Message
  ----     ------  ----               ----                          -------
  Warning  Error   16s (x2 over 16s)  trident-operator.netapp.io  Trident
is bound to another CR 'trident'
```

このエラーは、Tridentのインストールに使用されたがすでに存在することを示します `TridentOrchestrator`。各KubernetesクラスタはTridentのインスタンスを1つだけ持つことができるため、常に作成可能なアクティブなインスタンスが1つだけ存在するようにします `TridentOrchestrator`。

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、CRを編集する必要があります `TridentOrchestrator`。または、削除して、修正された正確な定義を使用して新しいものを作成することもできます `TridentOrchestrator`。

シヨウシテTridentヲトウニユウテキナイ `tridentctl`

何がうまくいかなかったのかを理解するために、引数を使用してインストーラを再度実行すると、`-d`デバッグモードがオンになり、問題の内容を理解するのに役立ちます。

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

問題に対処したら、次のようにインストールをクリーンアップし、コマンドを再度実行し `tridentctl install`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.
```

Astra TridentとCRDを完全に削除

Astra Tridentと作成されたCRDと関連するカスタムリソースをすべて完全に削除できます。

この操作は元に戻すことはできません。Astra Tridentを完全に新規にインストールする場合を除き、この作業は行わないでください。CRDを削除せずにAstra Tridentをアンインストールする方法については、を参照してください"Astra Trident をアンインストール"。

Trident オペレータ

Astra Tridentをアンインストールし、Tridentオペレータを使用してCRDを完全に削除するには、次の手順を実行します。

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

Helm

Astra Tridentをアンインストールし、Helmを使用してCRDを完全に削除する手順は次のとおりです。

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

<code> tridentctl </code>

Astra Tridentのアンインストール後にCRDを完全に削除するには `tridentctl`

```
tridentctl obliviate crd
```

RWX rawブロックネームスペースo Kubernetes 1.26でNVMeノードのステージング解除が失敗する

Kubernetes 1.26を実行している場合、RWX rawブロックネームスペースでNVMe/TCPを使用すると、ノードのステージング解除が失敗することがあります。次のシナリオは、障害に対する回避策を提供します。または、Kubernetesを1.27にアップグレードすることもできます。

ネームスペースとポッドが削除されました

Astra Tridentで管理されるネームスペース（NVMeの永続的ボリューム）をポッドに接続したシナリオを考えてみましょう。ネームスペースをONTAPバックエンドから直接削除すると、ポッドを削除しようとすると、ステージング解除プロセスが停止します。このシナリオは、Kubernetesクラスタやその他の機能には影響しません。

回避策

該当するノードから永続的ボリューム（そのネームスペースに対応するボリューム）をアンマウントして削除します。

ブロックされたデータ**LIF**

If you block (or bring down) all the dataLIFs of the NVMe Astra Trident backend, the unstaging process gets stuck when you attempt to delete the pod. In this scenario, you cannot run any NVMe CLI commands on the Kubernetes node.
.回避策
すべての機能を復元するには、dataLIFSを起動します。

ネームスペースマッピングが削除され

If you remove the `hostNQN` of the worker node from the corresponding subsystem, the unstaging process gets stuck when you attempt to delete the pod. In this scenario, you cannot run any NVMe CLI commands on the Kubernetes node.
.回避策
をサブシステムに再度追加し `hostNQN`ます。

サポート

NetAppは、Astra Tridentをさまざまな方法でサポートします。ナレッジベース（KB）記事やDiscordチャネルなど、24時間365日利用可能な無料のセルフサポートオプションをご用意しています。

Astra Tridentのサポートライフサイクル

Astra Tridentでは、バージョンに応じて3つのレベルのサポートが提供されます。を参照してください "定義に対するNetAppソフトウェアバージョンのサポート"。

フルサポート
: Astra Tridentは、リリース日から12カ月間フルサポートを提供します。

限定サポート
: Astra Tridentでは、リリース日から13~24カ月目に限定的なサポートを提供します。

セルフサポート
: Astra Tridentのドキュメントは、リリース日から25~36カ月間提供されます。

バージョン	フルサポート	限定サポート	セルフサポート
"24.06"	2025年6月	2026年6月	2027年6月
"24.02"	2025年2月	2026年2月	2027年2月
"23.10"	2024年10月	2025年10月	2026年10月

バージョン	フルサポート	限定サポート	セルフサポート
"23.07"	2024年7月	2025年7月	2026年7月
"23.04"	—	2025年4月	2026年4月
"23.01"	—	2025年1月	2026年1月
"22.10"	—	2024年10月	2025年10月
"22.07"	—	2024年7月	2025年7月
"22.04"	—	—	2025年4月
"22.01"	—	—	2025年1月
"21.10"	—	—	2024年10月

セルフサポート

トラブルシューティング記事の包括的なリストについては、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確認できます "ここをクリック"。

コミュニティサポート

ネットアップのAstraには、コンテナユーザ（Astra Trident開発者を含む）の活発なパブリックコミュニティがあり"チャネルを外します"ます。プロジェクトに関する一般的な質問をしたり、同じような気のある同僚と関連するトピックについて話し合うのには、この場所が最適です。

NetAppテクニカルサポート

Astra Tridentに関するサポートが必要な場合は、を使用してサポートバンドルを作成し `tridentctl logs -a -n trident`、に送信します `NetApp Support <Getting Help>`。

詳細情報

- "Astra ブログ"
- "Astra Trident のブログ"
- "Kubernetes ハブ"
- "netapp.io のコマンドです"

参考文献

Astra Trident ポート

Tridentが通信に使用するポートの詳細をご確認ください。

Astra Trident ポート

Astra Trident は次のポート経由で通信：

ポート	目的
8443	バックチャネル HTTPS
8001	Prometheus 指標エンドポイント
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、フラグを使用してインストール中に変更できます `--probe-port`。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

Astra Trident REST APIは最も簡単に操作できます"tridentctl コマンドとオプション"が、必要に応じてRESTエンドポイントを直接使用することもできます。

REST APIを使用する状況

REST APIは、Kubernetes以外の環境で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、ポッド内で実行する場合、Astra Tridentは `REST API`デフォルトでlocalhostに制限されます。この動作を変更するには、ポッド構成でAstra Tridentの引数を設定する必要があり `-address`ます。

REST APIを使用する

これらのAPIの呼び出し方法の例については、debug)(`-d`フラグを渡します。詳細については、を参照してください "Tridentctlを使用したAstra Tridentの管理"。

API は次のように機能します。

取得

GET <trident-address>/trident/v1/<object-type>

そのタイプのすべてのオブジェクトを一覧表示します。

GET <trident-address>/trident/v1/<object-type>/<object-name>

指定したオブジェクトの詳細を取得します。

投稿

POST <trident-address>/trident/v1/<object-type>

指定したタイプのオブジェクトを作成します。

- オブジェクトを作成するには JSON 構成が必要です。各オブジェクトタイプの仕様については、を参照してください"Tridentctlを使用したAstra Tridentの管理"。
- オブジェクトがすでに存在する場合、動作は一定ではありません。バックエンドが既存のオブジェクトを更新しますが、それ以外のすべてのオブジェクトタイプで処理が失敗します。

削除

DELETE <trident-address>/trident/v1/<object-type>/<object-name>

指定したリソースを削除します。

バックエンドまたはストレージクラスに関連付けられているボリュームは削除されず、削除されません。詳細については、を参照してください "Tridentctlを使用したAstra Tridentの管理"。

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

デバッグ出力を有効にします。

-loglevel <level>

ロギングレベル（debug、info、warn、error、fatal）を設定します。デフォルトは info です。

Kubernetes

-k8s_pod

このオプションまたはを使用し `-k8s_api_server`て、Kubernetesのサポートを有効にします。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を使用し `-k8s_pod`て、Kubernetesのサポートを有効にします。Trident を指定すると、セキュアでないアドレスとポートを使用して Kubernetes API サーバに接続されます。これにより、Trident をポッドの外部に導入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。安全に接続するには、オプションを使用してポッドにTridentを導入し `-k8s_pod`ます。

Docker

-volume_driver <name>

Dockerプラグインの登録時に使用するドライバ名。デフォルトはです `netapp`。

-driver_port <port-number>

UNIXドメインソケットではなく、このポートでリッスンします。

-config <file>

必須。バックエンド構成ファイルへのパスを指定する必要があります。

REST

-address <ip-or-host>

TridentのRESTサーバがリスンするアドレスを指定します。デフォルトは localhost です。localhost で聞いて Kubernetes ポッド内で実行しているときに、 REST インターフェイスにポッド外から直接アクセスすることはできません。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ために使用し `-address ""`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インターフェイスを有効にします。デフォルトは true です。

Kubernetes オブジェクトと Trident オブジェクト

リソースオブジェクトの読み取りと書き込みを行うことで、 REST API を使用して Kubernetes や Trident を操作できます。Kubernetes と Trident 、 Trident とストレージ、 Kubernetes とストレージの関係を決定するリソースオブジェクトがいくつかあります。これらのオブジェクトの中には Kubernetes で管理されるものと Trident で管理されるものがあります。

オブジェクトは相互にどのように相互作用しますか。

おそらく、オブジェクト、その目的、操作方法を理解する最も簡単な方法は、 Kubernetes ユーザからのストレージ要求を 1 回だけ処理することです。

1. ユーザは、管理者が以前に設定したKubernetesから、特定のサイズの `StorageClass`新しい要求を `PersistentVolume`作成します `PersistentVolumeClaim`。
2. Kubernetesは `StorageClass`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備えています。
3. Tridentは、同じ名前を使用して一致するものを識別し `Backends`、 `StoragePools`クラス用のボリュームのプロビジョニングに使用できるかどうかを確認し `StorageClass`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。1つは `PersistentVolume`Kubernetesでボリュームの検索、マウント、処理方法を指示するもので、もう1つはTridentで、もう1つはと実際のストレージの関係を保持するもの `PersistentVolume`です。
5. Kubernetesは、を新しいに `PersistentVolume`バインドし `PersistentVolumeClaim`ます。実行されるホスト上の`PersistentVolume`のマウントを含むポッド `PersistentVolumeClaim`。
6. ユーザは、Tridentをポイントするを使用して、既存のPVCの `VolumeSnapshotClass`を作成します `VolumeSnapshot`。
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、Snapshotを識別する方法をKubernetesに指示するを作成し `VolumeSnapshotContent`ます。
8. ユーザーは、をソースとして使用して `VolumeSnapshot`を作成できます `PersistentVolumeClaim`。
9. Tridentは必要なSnapshotを特定し、およびの `Volume`作成と同じ手順を実行します `PersistentVolume`。

Kubernetesオブジェクトの詳細については、Kubernetesドキュメントのセクションを読むことを強くお勧めします "永続ボリューム"。

`PersistentVolumeClaim`Kubernetesオブジェクト

Kubernetes `PersistentVolumeClaim`オブジェクトは、Kubernetesクラスタユーザによって行われたストレージへの要求です。

Trident では、標準仕様に加えて、バックエンド構成で設定したデフォルト設定を上書きする場合に、ボリューム固有の次のアノテーションを指定できます。

アノテーション	ボリュームオプション	サポートされているドライバ
trident.netapp.io/fileSystem	ファイルシステム	ONTAP-SAN、solidfire-san-エコノミー 構成、solidfire-san-SAN間にあるSolidFireを実現します
trident.netapp.io/cloneFromPVC	cloneSourceVolume の実行中です	ontap - NAS 、 ontap - san 、 solidfire-san-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

アノテーション	ボリュームオプション	サポートされているドライバ
trident.netapp.io/snapshotPolicy	Snapshot ポリシー	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup 、 ONTAP-SAN
trident.netapp.io/snapshotReserve	Snapshot リザーブ	ONTAP-NAS 、 ONTAP-NAS-flexgroup 、 ONTAP-SAN 、 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に再要求ポリシーが設定されている場合 `Delete`、PVが解放されると（つまり、ユーザがPVCを削除すると）、TridentはPVと元のボリュームの両方を削除します。削除操作が失敗した場合、 Trident は PV をマークします。そのような状態で操作が成功するか、 PV が手動で削除されるまで、定期的に再試行します。PVがポリシーを使用している場合 `Retain`、Tridentはポリシーを無視し、管理者がKubernetesとバックエンドからポリシーをクリーンアップすると想定します。これにより、削除前にボリュームをバックアップまたは検査できるようになります。PV を削除しても、原因 Trident で元のボリュームが削除されないことに注意してください。REST APIを使用して削除する必要があり(`tridentctl`ます）。

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。これがどのように機能するかを見て `On-Demand Volume Snapshots`ください。

Tridentには、クローンを作成するためのアノテーションとが `splitOnClone`用意されています `cloneFromPVC`。これらの注釈を使用して、CSI実装を使用せずにPVCのクローンを作成できます。

次に例を示します。ユーザがすでにというPVCを持っている場合、 `mysql`ユーザはなどの注釈を使用して `trident.netapp.io/cloneFromPVC: mysql`という新しいPVCを作成できます `mysqlclone`。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。
- ドライバと `ontap-san`ドライバを使用している `ontap-nas`場合は、と組み合わせて `trident.netapp.io/cloneFromPVC`PVCアノテーションを設定することをお勧めし `trident.netapp.io/splitOnClone`ます。 `trident.netapp.io/splitOnClone`をに設定する `true`と、Tridentは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、クローンボリュームのライフサイクルが親から完全に切り離されますが、ストレージ効率が低下することはありません。に設定または設定し `false`ない `trident.netapp.io/splitOnClone`とバックエンドのスペース消費が削減されますが、親ボリュームとクローンボリュームの間に依存関係が作成されるので、最初にクローンを削除しないかぎり親ボリュームを削除できません。クローンをスプリットするシナリオでは、空のデータベースボリュームをクローニングする方法が効果的です。このシナリオでは、ボリュームとそのクローンで使用するデータベースボリューム

のサイズが大きく異なっており、 ONTAP ではストレージ効率化のメリットはありません。

この `sample-input`ディレクトリには、Tridentで使用するPVC定義の例が含まれています。Tridentボリュームに関連するパラメータと設定の詳細については、を参照してください。

`PersistentVolume`Kubernetesオブジェクト

Kubernetes `PersistentVolume`オブジェクトは、Kubernetesクラスタで使用可能になるストレージの一部を表します。ポッドに依存しないライフサイクルがあります。

Tridentは、プロビジョニングするボリュームに基づいて自動的にオブジェクトを作成し `PersistentVolume`、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ベースを参照するPVCを作成すると `StorageClass`、Tridentは対応するストレージクラスを使用して新しいボリュームをプロビジョニングし、そのボリュームの新しいPVを登録します。プロビジョニングされたボリュームと対応する PV 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。

`StorageClass`Kubernetesオブジェクト

Kubernetes `StorageClass`オブジェクトは、の名前で指定し `PersistentVolumeClaims`、一連のプロパティを使用してストレージをプロビジョニングします。ストレージクラス自体が、使用するプロビジョニングツールを特定し、プロビジョニングツールが理解できる一連のプロパティを定義します。

管理者が作成および管理する必要がある 2 つの基本オブジェクトのうちの 1 つです。もう 1 つは Trident バックエンドオブジェクトです。

Tridentを使用するKubernetes `StorageClass`オブジェクトは次のようになります。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

これらのパラメータは Trident 固有で、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	タイプ	必須	説明
属性	[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	タイプ	値	提供	リクエスト	でサポートされます
メディア ^1	文字列	HDD 、ハイブリッド、 SSD	プールにはこのタイプのメディアが含まれています。ハイブリッドは両方を意味します	メディアタイプが指定されました	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup 、 ONTAPSAN 、 solidfire-san-SAN 、 solidfire-san-SAN のいずれかに対応しています
プロビジョニングタイプ	文字列	シン、シック	プールはこのプロビジョニング方法をサポートします	プロビジョニング方法が指定されました	シック：All ONTAP ；thin ：All ONTAP & solidfire-san-SAN

属性	タイプ	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、GCP-cvs、azure-NetApp-files、ONTAP-SAN-bエコノミー	プールはこのタイプのバックエンドに属しています	バックエンドが指定されて	すべてのドライバ
Snapshot	ブール値	true false	プールは、Snapshot を含むボリュームをサポートします	Snapshot が有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
クローン	ブール値	true false	プールはボリュームのクローニングをサポートします	クローンが有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
暗号化	ブール値	true false	プールでは暗号化されたボリュームをサポート	暗号化が有効なボリューム	ONTAP-NAS 、ONTAP-NAS-エコノミー 、ONTAP-NAS-FlexArray グループ、 ONTAP-SAN
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、単独でを使用して、特定のクラスのニーズを満たすために必要なストレージの品質をモデル化できます `attributes`。Tridentは、指定したの_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したプールを自動的に選択できない場合 `attributes`は、パラメータと `additionalStoragePools`パラメータを使用してプールをさらに絞り込んだり、特定のプールセットを選択したりできます `storagePools`。

パラメータを使用すると、指定したいずれかに一致するプールのセットをさらに制限 `attributes`でき `storagePools`ます。つまり、Tridentでは、パラメータと `storagePools`パラメータで識別されたプールの共通部分がプロビジョニングに使用され `attributes`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

パラメータを使用すると、パラメータと `storagePools`パラメータで選択し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 `attributes`でき `additionalStoragePools`ます。

パラメータを使用すると、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できます `excludeStoragePools`。このパラメータを使用すると、一致するプールがすべて削除されます。

> `storagePools`パラメータおよび `additionalStoragePools`パラメータでは、各エントリはの形式になり `<backend>:<storagePoolList>`ます。
> `<storagePoolList>`は、指定したバックエンドのストレージプールをカンマで区切ったリストです。たとえば、の値 `additionalStoragePools`はのようになります `ontapnas_192.168.1.100:aggr1,aggr2;solidfire_192.168.1.101:bronze`。これらのリストでは、バックエンド値とリスト値の両方に正規表現値を使用できます。を使用すると、バックエンドとそのプールのリストを取得できます `tridentctl get backend`。

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	タイプ	値	説明	関連するドライバ	**Kubernetes** のバージョン
FSstype （英語）	文字列	ext4、ext3、xfs	ブロックボリュームのファイルシステムのタイプ	solidfire-san-group、ontap/nas、ontap -nas-エコノミー、ontap -nas-flexgroup 、ontap -san 、ONTAP - SAN -経済性	すべて
allowVolumeExpansion の略	ブーリアン	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	タイプ	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	1.19 ~ 1.26

- パラメータは、 `fsType`SAN LUNに必要なファイルシステムタイプを制御するために使用します。さらに、Kubernetesはストレージクラスにが含まれていることを使用して `fsType`、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、が設定されている場合にのみ、ポッドのセキュリティコンテキストを `fsType`使用して制御でき `fsGroup`ます。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 `fsGroup`、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"。Kubernetesがこの値を適用する `fsGroup`のは、次の場合のみです。

  - `fsType`はストレージクラスに設定されます。
  - PVC アクセスモードは RWO です。

  NFS ストレージドライバの場合、 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。ストレージクラスを使用する `fsGroup`には、を指定する必要があり `fsType`ます。またはnull以外の任意の値に設定できます。 `nfs`

- ボリューム拡張の詳細については、を参照してください"ボリュームを展開します"。
- Tridentインストーラバンドルには、のTridentで使用するストレージクラスの定義例がいくつか用意されていますsample-input/storage-class-*.yaml。Kubernetes ストレージクラスを削除すると、対応する Trident ストレージクラスも削除されます。

`VolumeSnapshotClass`Kubernetesオブジェクト

Kubernetes `VolumeSnapshotClass`オブジェクトはに似てい `StorageClasses`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、管理者がを `VolumeSnapshotClass`定義する必要があります。ボリューム Snapshot クラスは、次の定義で作成されます。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

は driver、Kubernetesに対して、クラスのボリュームSnapshotの要求がTridentで処理されるように指定し

ます `csi-snapclass`。は、 `deletionPolicy`Snapshotを削除する必要がある場合に実行する処理を指定します。 `deletionPolicy`をに設定する `Delete`と、Snapshotを削除すると、ボリュームSnapshotオブジェクトとストレージクラスタ上の基盤となるSnapshotが削除されます。または、に設定する `Retain`と、 `VolumeSnapshotContent`物理Snapshotが保持されます。

`VolumeSnapshot`Kubernetesオブジェクト

Kubernetes `VolumeSnapshot`オブジェクトは、ボリュームのSnapshotの作成要求です。PVC がボリュームに対するユーザからの要求を表すのと同様に、ボリュームスナップショットは、ユーザが既存の PVC のスナップショットを作成する要求です。

ボリュームSnapshot要求を受信すると、Tridentはバックエンドでのボリュームの Snapshotの作成を自動的に管理し、一意のオブジェクトを作成してそのSnapshotを公開します。
`VolumeSnapshotContent`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。

`VolumeSnapshotContent`Kubernetesオブジェクト

Kubernetes `VolumeSnapshotContent`オブジェクトは、プロビジョニング済みのボリュームから取得されたSnapshotを表します。これは、に似て `PersistentVolume`おり、ストレージクラスタにプロビジョニングされたSnapshotを示します。オブジェクトと `PersistentVolume`オブジェクトと同様に、 `PersistentVolumeClaim`Snapshotが作成されると、 `VolumeSnapshotContent`オブジェクトはSnapshotの作成を要求したオブジェクトへの1対1のマッピングを保持し `VolumeSnapshot`ます。

> `VolumeSnapshotContent`オブジェクトには、`Snapshot`を一意に識別する詳細（など）が含まれます `snapshotHandle`。これは `snapshotHandle`、
> `PV`の名前とオブジェクトの名前の一意の組み合わせ `VolumeSnapshotContent`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はオブジェクトを構成し `VolumeSnapshotContent`、スナップショットをKubernetes APIに公開します。

> 通常、オブジェクトを管理する必要はありませ `VolumeSnapshotContent`ん。ただし、Astra Trident以外でを作成する場合は例外です"ボリュームSnapshotのインポート"。

`CustomResourceDefinition`Kubernetes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は、を実行して取得できます `kubectl get crds`。

カスタムリソース定義（ CRD ）と関連するオブジェクトメタデータは、 Kubernetes によってメタデータス

トアに格納されます。これにより、 Trident の独立したストアが不要になります。

Astra Tridentでは、オブジェクトを使用して `CustomResourceDefinition`、Tridentバックエンド、Tridentストレージクラス、Tridentボリュームなどの Trident オブジェクトの ID を保持します。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、を使用してバックエンドを作成する `tridentctl`と、Kubernetesで使用するために対応する `tridentbackends`CRDオブジェクトが作成されます。

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。
- コマンドを使用してTridentをアンインストールする `tridentctl uninstall`と、Tridentポッドは削除されますが、作成されたCRDはクリーンアップされません。Tridentを完全に削除してゼロから再構成する方法については、を参照してください"Trident をアンインストールします"。

Astra Trident `StorageClass`オブジェクト

Tridentは、プロビジョニングツールフィールドで指定されたKubernetesオブジェクト `csi.trident.netapp.io`に一致するストレージクラスを作成します `StorageClass`。ストレージクラス名は、そのストレージクラスが表すKubernetesオブジェクトの名前と一致し `StorageClass`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を登録すると、これらのオブジェクトが自動的に作成され `StorageClass`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デプロイメントの場合は、新しいKubernetesオブジェクトを登録するときに作成されることを想定してい `StorageClass`ます。

Astra 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 `StorageClass`オブジェクトです。

これらのオブジェクトの作成方法の詳細については、を参照してください"バックエンドの設定"。

Astra 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。

Astra Trident `Volume`オブジェクト

ボリュームは、 NFS 共有や iSCSI LUN などのバックエンドエンドエンドエンドポイントで構成される、プロビジョニングの基本単位です。Kubernetesでは、これらはに直接対応し `PersistentVolumes`ます。ボリュームを作成するときは、そのボリュームにストレージクラスが含まれていることを確認します。このクラスによって、ボリュームをプロビジョニングできる場所とサイズが決まります。

- 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	タイプ	必須	説明
バージョン	文字列	いいえ	Trident API のバージョン（「 1 」）
名前	文字列	はい	作成するボリュームの名前
ストレージクラス	文字列	はい	ボリュームのプロビジョニング時に使用するストレージクラス
サイズ	文字列	はい	プロビジョニングするボリュームのサイズ（バイト単位）
プロトコル	文字列	いいえ	使用するプロトコルの種類：「 file 」または「 block 」
インターン名	文字列	いいえ	Trident が生成した、ストレージシステム上のオブジェクトの名前
cloneSourceVolume の実行中です	文字列	いいえ	ONTAP （ NAS 、 SAN ） & SolidFire - * ：クローン元のボリュームの名前
splitOnClone	文字列	いいえ	ONTAP （ NAS 、 SAN ）：クローンを親からスプリットします
Snapshot ポリシー	文字列	いいえ	ONTAP - * ：使用する Snapshot ポリシー

属性	タイプ	必須	説明
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	ファイルシステムタイプ

ボリュームの作成時にTridentで生成され `internalName`ます。この構成は 2 つのステップで構成されます。最初に、ボリューム名の先頭にストレージプレフィックス（デフォルトまたはバックエンド構成のプレフィックス）が付加さ `trident`れ、形式の名前が生成されます。 `<prefix>-<volume-name>`その後、名前の完全消去が行われ、バックエンドで許可されていない文字が置き換えられます。ONTAPバックエンドの場合、ハイフンはアンダースコアに置き換えられます（内部名はになります `<prefix>_<volume-name>`）。Element バックエンドの場合、アンダースコアはハイフンに置き換えられます。

ボリューム構成を使用して、REST APIを使用してボリュームを直接プロビジョニングできますが、Kubernetes環境では、ほとんどのユーザが標準のKubernetesメソッドを使用することを想定してい`PersistentVolumeClaim`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。

Astra 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、これらはオブジェクトに直接対応し`VolumeSnapshotContent`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。

各 `Snapshot`オブジェクトには、次のプロパティが含まれています。

属性	タイプ	必須	説明
バージョン	文字列	はい	Trident API のバージョン（「 1 」）
名前	文字列	はい	Trident Snapshot オブジェクトの名前
インターン名	文字列	はい	ストレージシステム上の Trident Snapshot オブジェクトの名前
ボリューム名	文字列	はい	Snapshot を作成する永続的ボリュームの名前

属性	タイプ	必須	説明
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オブジェクト要求が作成されると、 `VolumeSnapshot`Tridentは元のストレージシステムにSnapshotオブジェクトを作成することで機能します。このSnapshotオブジェクトのは `internalName`、プレフィックスと `VolumeSnapshot`オブジェクトのを `UID`組み合わせることで生成され `snapshot-`ます（例： `snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660`）。`volumeName`および`volumeInternalName`は、バッキングボリュームの詳細を取得することによって読み込まれます。

Astra Trident `ResourceQuota`オブジェクト

Tridentデーモンセットは、Kubernetesで利用可能な最高の優先度クラスであるプライオリティクラスを使用して `system-node-critical`、ノードの正常なシャットダウン時にAstra Tridentがボリュームを特定してクリーンアップできるようにし、リソースへの負荷が高いクラスタでは、Tridentデーモンセットポッドが優先度の低いワークロードをプリエンプトできるようにします。

これを実現するために、Astra Tridentではオブジェクトを使用し `ResourceQuota`て、`Trident`デーモンセットで「`system-node-critical`」優先クラスを確実に満たします。導入とデーモンセットの作成の前に、`Astra Trident`がオブジェクトを検索し `ResourceQuota`、検出されなかった場合は適用します。

デフォルトのリソースクォータと優先クラスを詳細に制御する必要がある場合は、Helmチャートを使用してオブジェクトを生成または設定 `ResourceQuota`できます `custom.yaml`。

次に示すのは'ResourceQuota'オブジェクトがTridentのデマ作用を優先する例です

```
apiVersion: <version>
kind: ResourceQuota
metadata:
  name: trident-csi
  labels:
    app: node.csi.trident.netapp.io
spec:
  scopeSelector:
     matchExpressions:
       - operator : In
         scopeName: PriorityClass
         values: ["system-node-critical"]
```

リソースクォータの詳細については、を参照してください"Kubernetes：リソースクォータ"。

インストールに失敗した場合のクリーンアップ `ResourceQuota`

まれに、オブジェクトの作成後にインストールが失敗する `ResourceQuota`場合は、最初に再インストールを試行してから再インストールして"アンインストール"ください。

それでも問題が解決しない場合は、オブジェクトを手動で削除し `ResourceQuota`ます。

取り外す `ResourceQuota`

リソースの割り当てを独自に制御する場合は、コマンドを使用してAstra Tridentオブジェクトを削除できます `ResourceQuota`。

```
kubectl delete quota trident-csi -n trident
```

PODセキュリティ標準（PSS）およびセキュリティコンテキストの制約（SCC）

Kubernetesポッドのセキュリティ標準（PSS）とポッドのセキュリティポリシー（PSP）によって、権限レベルが定義され、ポッドの動作が制限されます。また、OpenShift Security Context Constraints（SCC）でも、OpenShift Kubernetes Engine固有のポッド制限を定義します。このカスタマイズを行うために、Astra Tridentはインストール時に特定の権限を有効にします。次のセクションでは、Astra Tridentによって設定された権限の詳細を説明します。

PSSは、Podセキュリティポリシー（PSP）に代わるものです。PSPはKubernetes v1.21で廃止され、v1.25で削除されます。詳細については、を参照してください"Kubernetes：セキュリティ"。

必須の**Kubernetes Security Context**と関連フィールド

権限	説明
権限があります	CSIでは、マウントポイントが双方向である必要があります。つまり、Tridentノードポッドで特権コンテナを実行する必要があります。詳細については、を参照してください "Kubernetes：マウントの伝播"。
ホストネットワーク	iSCSIデーモンに必要です。 `iscsiadm`iSCSIマウントを管理し、ホストネットワークを使用してiSCSIデーモンと通信します。
ホストIPC	NFSは'IPC（プロセス間通信）を使用して'nfsdと通信します
ホストPID	NFSで開始する必要があり `rpc-statd`ます。Astra Tridentは、ホストプロセスを照会して、NFSボリュームをマウントする前にが実行されているかどうかを確認し `rpc-statd`ます。

権限	説明

機能	この `SYS_ADMIN`機能は、特権コンテナのデフォルト機能の一部として提供されています。たとえば、`Docker`は特権コンテナに次の機能を設定します。
	`CapPrm: 0000003fffffffff`
	`CapEff: 0000003fffffffff`

Seccom	Seccompプロファイルは、権限のあるコンテナでは常に「制限なし」なので、Astra Tridentでは有効にできません。
SELinux	OpenShiftでは、特権コンテナは（「Super Privileged Container」）ドメインで実行され、非特権コンテナはドメインで実行さ `spc_t`れ `container_t`ます。では `containerd`、がインストールされていると `container-selinux`、すべてのコンテナがドメイン内で実行され `spc_t`、SELinuxが実質的に無効になります。そのため、Astra Tridentはコンテナに追加しません `seLinuxOptions`。
DAC	特権コンテナは、ルートとして実行する必要があります。CSIに必要なUNIXソケットにアクセスするために、非特権コンテナはrootとして実行されます。

PODセキュリティ標準（PSS）

ラベル	説明	デフォルト
`pod-security.kubernetes.io/enforce pod-security.kubernetes.io/enforce-version`	Tridentコントローラとノードをインストールネームスペースに登録できるようにします。ネームスペースラベルは変更しないでください。	`enforce: privileged enforce-version: <version of the current cluster or highest version of PSS tested.>`

名前空間ラベルを変更すると、ポッドがスケジュールされず、「Error creating：…」または「Warning：trident-csi-…」が表示される場合があります。この場合は、のネームスペースラベルが変更されていないかどうかを確認してください `privileged`。その場合は、Tridentを再インストールします。

PoDセキュリティポリシー（PSP）

フィールド	説明	デフォルト
`allowPrivilegeEscalation`	特権コンテナは、特権昇格を許可する必要があります。	`true`
`allowedCSIDrivers`	TridentはインラインCSIエフェメラルボリュームを使用しません。	空

フィールド	説明	デフォルト
allowedCapabilities	権限のないTridentコンテナにはデフォルトよりも多くの機能が必要ないため、特権コンテナには可能なすべての機能が付与されます。	空
allowedFlexVolumes	Tridentではを使用できない"FlexVolドライバ"ため、これらのボリュームは許可されるボリュームのリストに含まれていません。	空
allowedHostPaths	Tridentノードポッドでノードのルートファイルシステムがマウントされるため、このリストを設定してもメリットはありません。	空
allowedProcMountTypes	Tridentはいずれも使用しません ProcMountTypes。	空
allowedUnsafeSysctls	Tridentには安全でないものは必要あり `sysctls`ません。	空
defaultAddCapabilities	特権コンテナに追加する機能は必要ありません。	空
defaultAllowPrivilegeEscalation	権限の昇格は、各Tridentポッドで処理されます。	false
forbiddenSysctls	いいえは `sysctls`許可されません。	空
fsGroup	Tridentコンテナはrootとして実行されます。	RunAsAny
hostIPC	NFSボリュームをマウントするにはホストIPCが通信する必要があります nfsd	true
hostNetwork	iscsiadmには、iSCSIデーモンと通信するためのホストネットワークが必要です。	true
hostPID	ホストPIDは、がノードで実行されているかどうかを確認するために必要 `rpc-statd`です。	true
hostPorts	Tridentはホストポートを使用しません。	空
privileged	Tridentノードのポッドでは、ボリュームをマウントするために特権コンテナを実行する必要があります。	true
readOnlyRootFilesystem	Tridentノードのポッドは、ノードのファイルシステムに書き込む必要があります。	false

フィールド	説明	デフォルト
requiredDropCapabilities	Tridentノードのポッドは特権コンテナを実行するため、機能をドロップすることはできません。	none
runAsGroup	Tridentコンテナはrootとして実行されます。	RunAsAny
runAsUser	Tridentコンテナはrootとして実行されます。	runAsAny
runtimeClass	Tridentはを使用しません RuntimeClasses。	空
seLinux	現在、コンテナランタイムとKubernetesディストリビューションでSELinuxを処理する方法が異なるため、Tridentは設定されて`seLinuxOptions`いません。	空
supplementalGroups	Tridentコンテナはrootとして実行されます。	RunAsAny
volumes	Tridentポッドには、このボリュームプラグインが必要です。	hostPath, projected, emptyDir

セキュリティコンテキストの制約（SCC）

ラベル	説明	デフォルト
allowHostDirVolumePlugin	Tridentノードのポッドは、ノードのルートファイルシステムをマウントします。	true
allowHostIPC	NFSボリュームをマウントするには、ホストIPCと通信する必要があり`nfsd`ます。	true
allowHostNetwork	iscsiadmには、iSCSIデーモンと通信するためのホストネットワークが必要です。	true
allowHostPID	ホストPIDは、がノードで実行されているかどうかを確認するために必要`rpc-statd`です。	true
allowHostPorts	Tridentはホストポートを使用しません。	false
allowPrivilegeEscalation	特権コンテナは、特権昇格を許可する必要があります。	true
allowPrivilegedContainer	Tridentノードのポッドでは、ボリュームをマウントするために特権コンテナを実行する必要があります。	true

ラベル	説明	デフォルト
`allowedUnsafeSysctls`	Tridentには安全でないものは必要あり `sysctls`ません。	`none`
`allowedCapabilities`	権限のないTridentコンテナにはデフォルトよりも多くの機能が必要ないため、特権コンテナには可能なすべての機能が付与されます。	空
`defaultAddCapabilities`	特権コンテナに追加する機能は必要ありません。	空
`fsGroup`	Tridentコンテナはrootとして実行されます。	`RunAsAny`
`groups`	このSCCはTridentに固有で、ユーザにバインドされています。	空
`readOnlyRootFilesystem`	Tridentノードのポッドは、ノードのファイルシステムに書き込む必要があります。	`false`
`requiredDropCapabilities`	Tridentノードのポッドは特権コンテナを実行するため、機能をドロップすることはできません。	`none`
`runAsUser`	Tridentコンテナはrootとして実行されます。	`RunAsAny`
`seLinuxContext`	現在、コンテナランタイムとKubernetesディストリビューションでSELinuxを処理する方法が異なるため、Tridentは設定されて `seLinuxOptions`いません。	空
`seccompProfiles`	特権のあるコンテナは常に「閉鎖的」な状態で実行されます。	空
`supplementalGroups`	Tridentコンテナはrootとして実行されます。	`RunAsAny`
`users`	このSCCをTridentネームスペースのTridentユーザにバインドするエントリが1つあります。	N/A
`volumes`	Tridentポッドには、このボリュームプラグインが必要です。	`hostPath, downwardAPI, projected, emptyDir`

法的通知

法的通知では、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、それを所有する各社の商標である場合があります。

"https://www.netapp.com/company/legal/trademarks/"

特許

NetAppが所有する特許の最新リストは、次のサイトで参照できます。

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

Astra TridentのNetAppソフトウェアで使用されているサードパーティの著作権とライセンスは、各リリースのnoticesファイルで確認できます https://github.com/NetApp/trident/。

著作権に関する情報

商標に関する情報