Astra Trident 24.02ドキュメント

Astra Trident

NetApp
April 18, 2024

This PDF was generated from https://docs.netapp.com/ja-jp/trident/index.html on April 18, 2024. Always check docs.netapp.com for the latest.

目次

Astra Trident 24.02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。

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

24.02の新機能

拡張機能

- Cloud Identityのサポートが追加されました。
  - ANF-AzureワークロードIDを持つAKは、クラウドIDとして使用されます。
  - FSxN-AWS IAMロールを持つEKSがクラウドIDとして使用されます。
- EKSコンソールからEKSクラスタにアドオンとして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（"問題#864"）。
- ONTAPドライバにFSxパーソナリティを追加して、AWS IAMやSecretsManagerとの統合を可能にし、Astra Tridentでバックアップ付きのFSxボリュームを削除できるようにしました（"問題#453"）。

Kubernetes

- Kubernetes 1.29のサポートを追加。

の修正

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

非推奨

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

23.10の変更点

の修正

- 要求された新しいサイズがontap-nasおよびontap-nas-flexgroupストレージドライバの合計ボリュームサイズよりも小さい場合、ボリュームの拡張が修正されました（"問題#834"）。
- ontap-nasおよびontap-nas-flexgroupストレージドライバのインポート時にボリュームの使用可能なサイズ

のみを表示するための固定ボリュームサイズ（"問題#722"）。

- ONTAP-NAS-EconomyのFlexVol名変換が修正されました。
- ノードのリブート時に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。

拡張機能

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

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

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（"問題#816"）。
- use-rest = trueを指定してONTAPドライバを使用すると、クォータのサイズが修正されました。
- ONTAP-SAN-EconomyでLUNクローンを固定作成
- パブリッシュ情報フィールドを元に戻す `rawDevicePath` 終了： `devicePath`;データの取り込みとリカバリのためのロジックを追加(場合によっては) `devicePath` フィールド。

拡張機能

Kubernetes

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

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

の修正

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

拡張機能

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

- Windowsおよびarm64ホストのクロスプラットフォームビルドサポートをMakefileに追加しました。build.mdを参照してください。

非推奨

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

23.01.1の変更点

の修正

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

23.01の変更点

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

の修正

- Kubernetes：Helm（"問題#783、#794"）。

拡張機能

Kubernetes

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

Astra Trident

- ONTAP 対応のAmazon FSXを使用したSMBボリュームのONTAP NASストレージドライバへのサポートが追加されました。
- SMBボリュームの使用時のNTFS権限のサポートが追加されました。
- CVSサービスレベルを使用したGCPボリュームのストレージプールのサポートが追加されました。
- FlexGroupをONTAP-NAS-flexgroupストレージドライバで作成する際のflexgroupAggregateListのオプション使用がサポートされるようになりました。

- 複数のFlexVolを管理する場合の、ONTAPとNASの両方に対応したストレージドライバのパフォーマンスが向上しました。
- すべてのONTAP NASストレージドライバに対してデータLIFの更新を有効にしました。
- Trident DeploymentとDemonSetの命名規則を更新し、ホストノードOSを反映させました。

非推奨

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

22.10の変更

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

> **Astra Tridentに関する重要な情報22.10**
>
> - TridentでKubernetes 1.25がサポートされるようになりました。Kubernetes 1.25にアップグレードする前に、Astra Tridentを22.10にアップグレードする必要があります。
> - Astra Tridentは、SAN環境でマルチパス構成を厳密に使用するよう強制し、推奨値をに設定するようになりました `find_multipaths: no` multipath.confファイル内。
>
> 非マルチパス構成またはを使用 `find_multipaths: yes` または `find_multipaths: smart` multipath.confファイルの値が原因でマウントが失敗します。Tridentはの使用を推奨しています `find_multipaths: no` 21.07リリース以降

の修正

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

拡張機能

- Kubernetes：
  - Kubernetes 1.25のサポートが追加されました。Kubernetes 1.25にアップグレードする前に、Astra Tridentを22.10にアップグレードする必要があります。

  - Trident Deployment and DemonSet用に別々のServiceAccount、ClusterRole、ClusterRoleBindingを追加して、今後の権限の強化を可能にしました。
  - のサポートが追加されました "ネームスペース間ボリューム共有"。
- すべてTrident `ontap-*` ストレージドライバがONTAP REST APIで機能するようになりました。
- 新しい演算子YAMLを追加しました (`bundle_post_1_25.yaml`）を使用しない場合 `PodSecurityPolicy` Kubernetes 1.25をサポートするため。
- を追加しました "LUKS暗号化ボリュームをサポートします" の場合 `ontap-san` および `ontap-san-economy` ストレージドライバ。
- Windows Server 2019ノードのサポートが追加されました。
- を追加しました "WindowsノードでのSMBボリュームのサポート" を使用する `azure-netapp-files` ストレージドライバ。
- ONTAP ドライバの自動MetroCluster スイッチオーバー検出機能が一般提供されるようになりました。

非推奨

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

2007年22月の変更

の修正

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

拡張機能

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

非推奨

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

削除します

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

ドキュメント

新しい "PODセキュリティ標準" （PSS）セクションに、インストール時にAstra Tridentによって有効化された権限の詳細が追加されました。

2004年10月22日の変更

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

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

の修正

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

拡張機能

- Trident のセキュアエンドポイントへのインバウンド接続には、 TLS 1.3 以上が必要です。（"GitHub問題 #698"）
- Trident では、セキュアなエンドポイントからの応答に HSTS ヘッダーが追加されました。

- Trident では、 Azure NetApp Files の UNIX 権限機能が自動的に有効化されるようになりました。
- * Kubernetes * ： Trident のデプロイ機能は、システムノードに不可欠な優先度クラスで実行されるようになりました。（"GitHub問題 #694"）

削除します

E シリーズドライバ（ 20.07 以降無効）が削除されました。

22.01.1 の変更

の修正

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

22.01.0 の変更

の修正

- * Kubernetes ：大規模なクラスタのノード登録バックオフ再試行時間を延長します。
- azure-NetApp-files ドライバが、同じ名前の複数のリソースによって混乱することがあるという解決済みの問題 。
- 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。

拡張機能

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

非推奨

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

21.10.1 の変更点

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

の修正

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

21.10.0 の変更点

の修正

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

拡張機能

- GCP - CVS パフォーマンスボリュームに対する「 unixPermissions 」オプションのサポートが追加されました。

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

実験的な機能強化

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

既知の問題

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

- Astra Tridentがインストールされているkubernetesクラスタを1.24から1.25以降にアップグレードする場合は、value.yamlを更新して設定する必要があります `excludePodSecurityPolicy` 終了： `true` または、を追加します `--set excludePodSecurityPolicy=true` に移動します `helm upgrade` コマンドを実行してからクラスタをアップグレードしてください。
- Astra Trident は ' ストレージクラスに fsType が指定されていないボリュームに対して ' ブランクの fsType(`fsType="") を適用するようになりましたKubernetes 1.17 以降を使用する場合、 Trident は NFS ボリュームに空の「 fsType 」を提供できます。iSCSI ボリュームの場合 ' セキュリティコンテキストを使用して fsGroup を実行するときに 'fsType' を StorageClass 上に設定する必要があります
- 複数の Astra Trident インスタンス間でバックエンドを使用する場合、各バックエンド構成ファイルの ONTAP バックエンドに異なる「 toragePrefix 」値を指定するか、 SolidFire バックエンドに異なる「 tenantname 」を使用する必要があります。Astra Trident は、 Astra Trident の他のインスタンスが作成したボリュームを検出できません。ONTAP または SolidFire バックエンドに既存のボリュームを作成しようとすると成功します。 Astra Trident は、ボリューム作成をべき等の操作として扱います。「 toragePrefix 」または「 tenantname 」が異なる場合は、同じバックエンド上に作成されたボリュームに名前の衝突が発生する可能性があります。
- Astra Trident をインストールするときに (tridentctl または Trident Operator を使用) 、 tridentctl を使用して Astra Trident を管理するときは、「 KUBECONFIG 」環境変数が設定されていることを確認する必要があります。これは 'tridentctl が機能するべき Kubernetes クラスタを示すために必要です複数の Kubernetes 環境で作業する場合は、「 KBECONFIG 」ファイルが正確にソースされていることを確認する必要があります。
- iSCSI PVS のオンラインスペース再生を実行するには、作業者ノード上の基盤となる OS がボリュームにマウントオプションを渡す必要があります。これは、「 discard 」を必要とする RHEL/RedHat CoreOS インスタンスに当てはまります "マウントオプション"; discard mountOption がに含まれていることを確認します "d4b9b9554fd820f43eae492d33e41167" オンラインブロックの破棄をサポートするため。
- Kubernetes クラスタごとに複数の Astra Trident インスタンスがある場合、 Astra Trident は他のインスタンスと通信できず、作成した他のボリュームを検出できません。そのため、 1 つのクラスタ内で複数のインスタンスを実行している場合、予期しない動作が発生したり、誤ったりすることがあります。Kubernetes クラスタごとに Trident のインスタンスが 1 つだけ必要です。
- Trident がオフラインのときに Astra Trident ベースの「 torageClass 」オブジェクトが Kubernetes から削除された場合、 Astra Trident は、対応するストレージクラスをオンラインに戻ってもデータベースから削

除しません。これらのストレージクラスは、「 tridentctl 」または REST API を使用して削除してください。

- 対応する PVC を削除する前に Astra Trident によってプロビジョニングされた PV を削除しても、 Astra Trident は自動的に元のボリュームを削除しません。tridentctl または REST API を使用してボリュームを削除してください。
- FlexGroup では、プロビジョニング要求ごとに一意のアグリゲートセットがないかぎり、同時に複数の ONTAP をプロビジョニングすることはできません。
- IPv6 経由で Astra Trident を使用する場合は、バックエンド定義内の角かっこ内に「 managementlif 」と「 datalif 」を指定する必要があります。例えば、「 [fd20 ： 8b1e ： b258 ： 2000 ： f816 ： 3eff ： feec ： 0]`` 」のようになります。

> を指定することはできません `dataLIF` ONTAP SANバックエンドの場合：Astra Trident は、使用可能なすべてのiSCSI LIFを検出し、それらを使用してマルチパスセッションを確立します。

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

詳細については、こちらをご覧ください

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

以前のバージョンのドキュメント

Astra Trident 24.02を実行していない場合、以前のリリースのドキュメントは、 "Astra Tridentのサポートライフサイクル"。

- "Astra Trident 23.10"
- "Astra Trident 23.07"
- "Astra Trident 23.04"
- "Astra Trident 23.01"
- "Trident 10月22日"
- "トライデント22.07年アストラト"
- "トライデント22.04アストラ"
- "Trident 22.01"
- "Astra Trident 21.10"
- "Astra Trident 21.07"

はじめに

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、NetAppが "Astra 製品ファミリー"。Container Storage Interface（CSI）などの業界標準のインターフェイスを使用して、コンテナ化されたアプリケーションの永続性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。

の詳細を確認してください "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、コンテナストレージインターフェイス（CSI）に準拠した動的ストレージオーケストレーションツールで、 "Kubernetes"。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。を参照してください "Astra Tridentのアーキテクチャ" を参照してください。

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。を参照してください "Astra Trident for Docker を導入" を参照してください。

> Kubernetesを初めて使用する場合は、 "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、「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 "ネットアップのテスト用ドライブ" すぐに使用できるラボイメージを使用する。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 が提供します。

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 の特長

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

Cloud Volumes ONTAP

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

Cloud Volumes Service for Google Cloud

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

Element ソフトウェア

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

NetApp HCI

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

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。

を参照してください。

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

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を単一システムとして導入 Tridentコントローラポッド および1つ以上 Tridentノードポッド Kubernetesクラスタ上で、標準のKUBSI_CSI Sidecar Containers_を使用してCSIプラグインの導入を簡素化します。 "Kubernetes CSIサイドカーコンテナ" Kubernetes Storageコミュニティが管理しています。

Kubernetes "ノードセレクタ" および "寛容さと汚れ" は、特定のノードまたは優先ノードで実行されるようにポッドを制限するために使用されます。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。

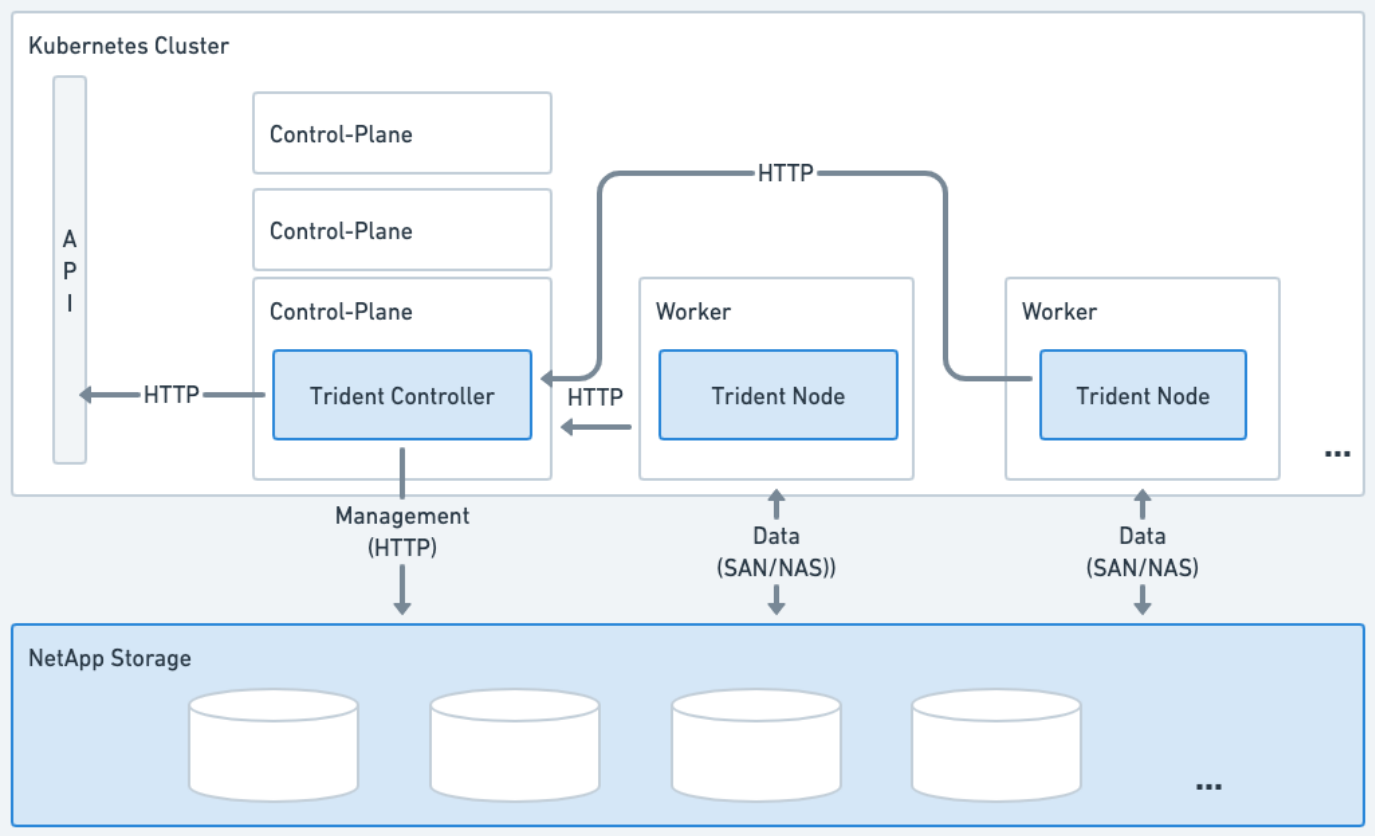


図 **1. Kubernetes**クラスタに導入される**Astra Trident**

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。

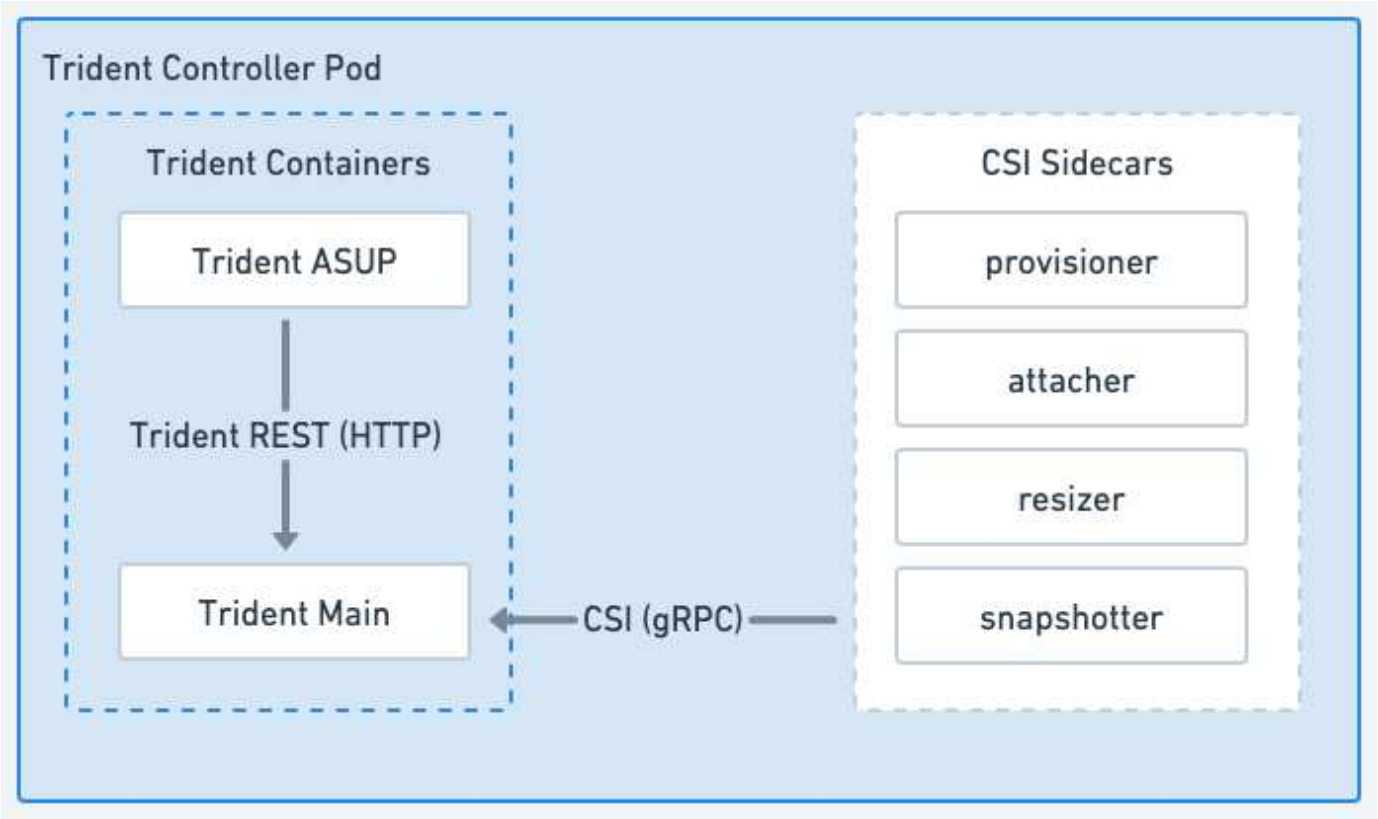


図 **2. Trident**コントローラポッドの図

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

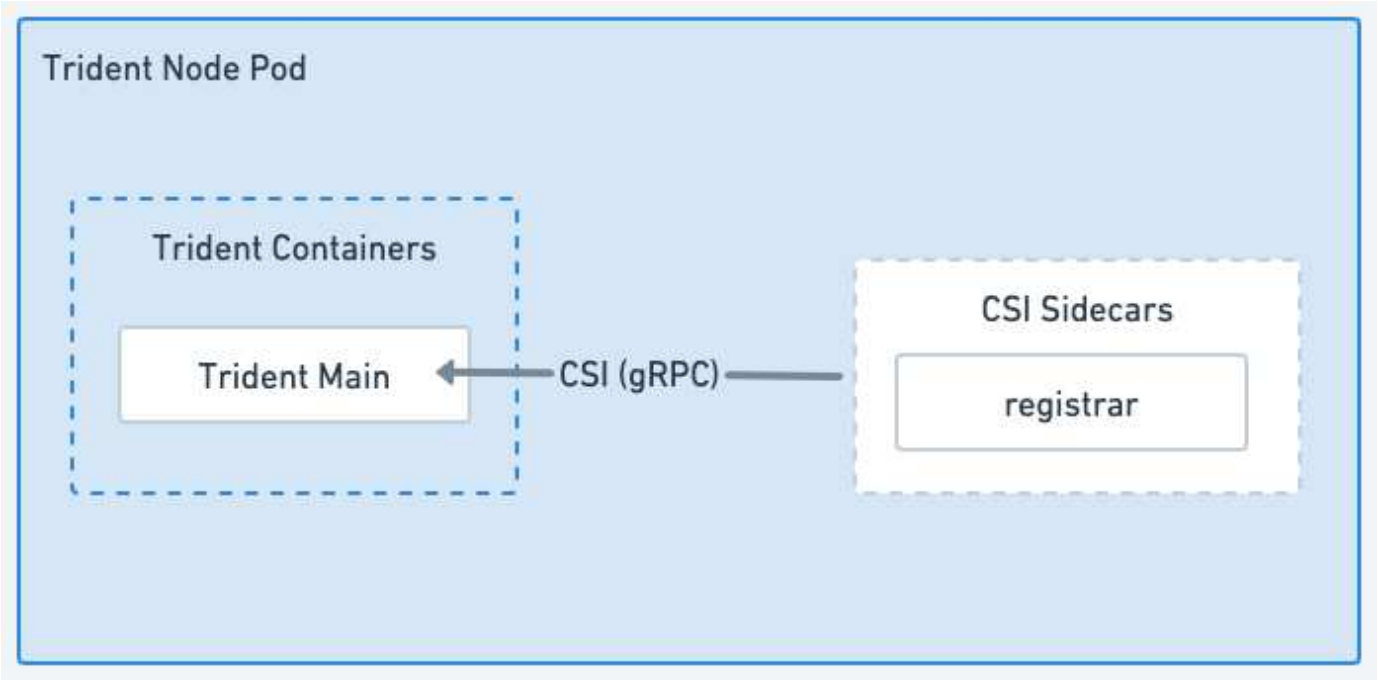


図 **3. Trident**ノードのポッド図

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

概念

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 'toragePools’additionalStoragePools'`excludeStoragePools' リストの両方が必要ですストレージクラスを作成すると、 Trident はバックエンドごとに提供される属性とプールを、ストレージクラスから要求された属性とプールと比較します。要求された属性とプール名がストレージプールの属性と名前ですべて一致した場合、 Astra Trident がそのストレージプールを、そのストレージクラスに適した一連のストレージプールに追加します。さらに、属性がストレージクラスの要求された属性のすべてまたは一部を満たしていない場合でも、 Astra Trident は、そのセットに「 additionalStoragePools 」リストにリストされているすべてのストレージプールを追加します。ストレージ・クラスのストレージ・プールを上書きして ' ストレージ・クラスに使用しないようにするには 'excludeStoragePools' リストを使用する必要がありますAstra Trident では、新しいバックエンドを追加するたびに同様のプロセスが実行され、ストレージプールが既存のストレージクラスのストレージクラスを満たしているかどうかが確認され、除外済みとマークされているストレージが削除されます。

ボリュームの作成

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

ボリューム **Snapshot**

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

ボリューム**Snapshot**の作成方法について説明します

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

ースはごくわずかです。

- ONTAP スナップショットは 'ONTAP-NAS' および「 ONTAP-SAN' 」ドライバと連携して動作する場合 ' FlexVol のポイント・イン・タイム・コピーであり ' FlexVol 自体の領域を消費しますその結果、ボリューム内の書き込み可能なスペースが、 Snapshot の作成やスケジュール設定にかかる時間を短縮できます。この問題に対処する簡単な方法の 1 つは、 Kubernetes を使用してサイズを変更することでボリュームを拡張することです。もう 1 つの方法は、不要になった Snapshot を削除することです。Kubernetes で作成されたボリューム Snapshot を削除すると、関連付けられている ONTAP Snapshot が Astra Trident から削除されます。Kubernetes で作成されていない ONTAP スナップショットも削除できます。

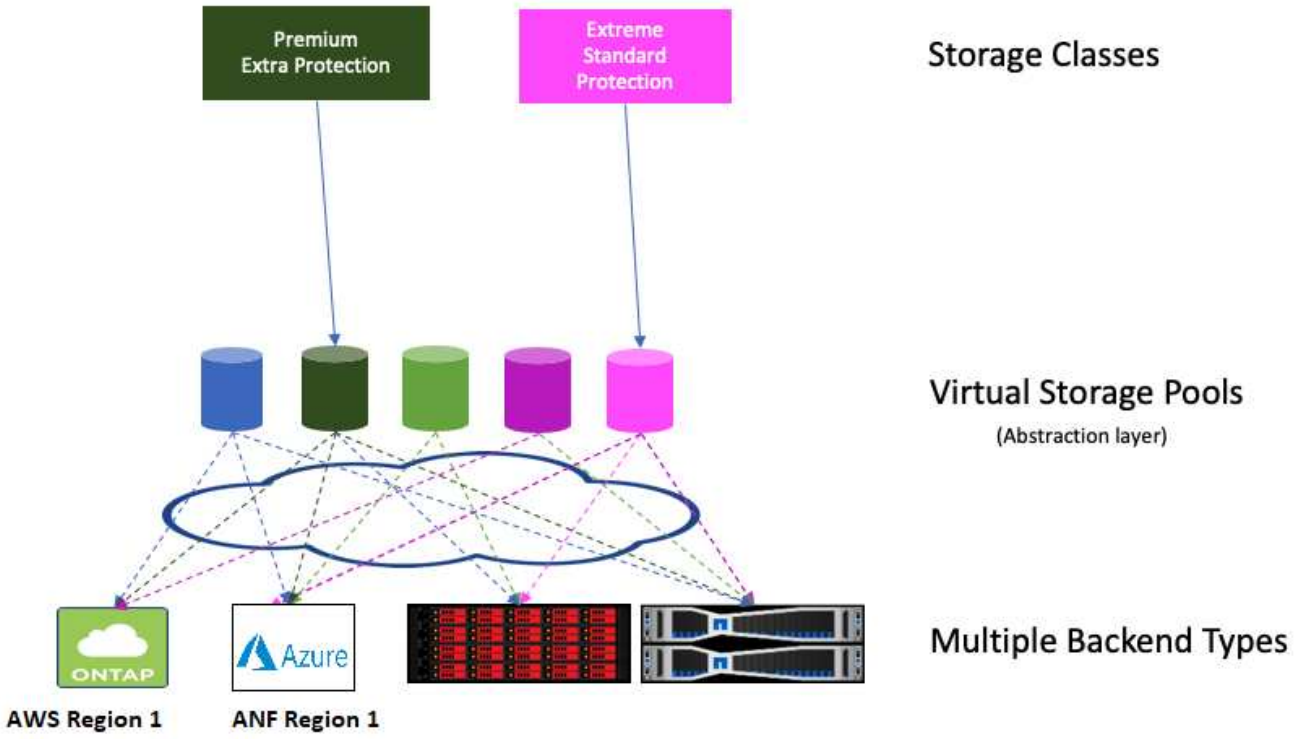
ネットアップの Trident では、ボリューム Snapshot を使用して PVS を新規作成できます。これらの Snapshot から PVS を作成するには、サポート対象の ONTAP および CVS バックエンドに対して FlexClone テクノロジを使用します。SnapshotからPVを作成する場合、元のボリュームはSnapshotの親ボリュームのFlexCloneになります。。 `solidfire-san` ドライバは、Elementソフトウェアのボリュームクローンを使用してSnapshotからPVSを作成します。ここで、 Element Snapshot からクローンを作成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

仮想プール

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

仮想プールについて説明します

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



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

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

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

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

演算子	例	プールのラベル値は次のとおりです。
=	パフォーマンス = プレミアム	一致
`!= `	パフォーマンス！ = 非常に優れています	一致しません
「 in`	場所（東部、西部）	値のセットに含まれています

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

ボリュームアクセスグループ

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

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

ボリュームアクセスグループについて学習する

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

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。 "こちらをご覧ください"。

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

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、 "Astra Trident の要件"。

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

1 **Astra Trident** をインストール

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します"

3 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。

"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

- TridentでKubernetes 1.29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は、SAN環境でマルチパス構成を厳密に使用し、推奨される値をに設定します `find_multipaths: no` multipath.confファイル内。

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

サポートされるフロントエンド（オーケストレーションツール）

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

- Anthosオンプレミス（VMware）とAnthos（ベアメタル1.16）
- Kubernetes 1.23~1.29
- OpenShift 4.10-4.14

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 1.23~1.29
- OpenShift 4.10-4.14

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、 "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 の特長
- Cloud Volumes ONTAP
- Cloud Volumes Service for GCP

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

機能の要件

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

フィーチャー（ **Feature** ）	**Kubernetes** のバージョン	フィーチャーゲートが必要ですか？
Astra Trident	1.23 ~ 1.29	いいえ
ボリューム Snapshot	1.23 ~ 1.29	いいえ
ボリューム Snapshot からの PVC	1.23 ~ 1.29	いいえ
iSCSI PV のサイズ変更	1.23 ~ 1.29	いいえ
ONTAP 双方向 CHAP	1.23 ~ 1.29	いいえ
動的エクスポートポリシー	1.23 ~ 1.29	いいえ
Trident のオペレータ	1.23 ~ 1.29	いいえ
CSI トポロジ	1.23 ~ 1.29	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 2019（AMD64）

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

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

ホストの設定

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

"ワーカーノードを準備します"

ストレージシステムの構成：

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

Kubernetes のバージョン	コンテナイメージ
v1.3.0	• Docker .io / NetApp / Trident：24.02.0 • docker.io / netapp/trident-autosupport：24.02 • registry.k8s.io/sig-storage/csi-provisioner：v3.6.0 • registry.k8s.io/sig-storage/csi-attacher：v4.4.0 • registry.k8s.io/sig-storage/csi-resizer：v1.9.0 • registry.k8s.io/sig-storage/csi-snapshotter：v6.3.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.9.0 • docker.io/netapp/trident-operator：24.02.0（オプション）

Kubernetes のバージョン	コンテナイメージ
v1.24.0	• Docker .io / NetApp / Trident：24.02.0 • docker.io / netapp/trident-autosupport：24.02 • registry.k8s.io/sig-storage/csi-provisioner：v3.6.0 • registry.k8s.io/sig-storage/csi-attacher：v4.4.0 • registry.k8s.io/sig-storage/csi-resizer：v1.9.0 • registry.k8s.io/sig-storage/csi-snapshotter：v6.3.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.9.0 • docker.io/netapp/trident-operator：24.02.0（オプション）
v1.25.0	• Docker .io / NetApp / Trident：24.02.0 • docker.io / netapp/trident-autosupport：24.02 • registry.k8s.io/sig-storage/csi-provisioner：v3.6.0 • registry.k8s.io/sig-storage/csi-attacher：v4.4.0 • registry.k8s.io/sig-storage/csi-resizer：v1.9.0 • registry.k8s.io/sig-storage/csi-snapshotter：v6.3.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.9.0 • docker.io/netapp/trident-operator：24.02.0（オプション）
v1.26.0	• Docker .io / NetApp / Trident：24.02.0 • docker.io / netapp/trident-autosupport：24.02 • registry.k8s.io/sig-storage/csi-provisioner：v3.6.0 • registry.k8s.io/sig-storage/csi-attacher：v4.4.0 • registry.k8s.io/sig-storage/csi-resizer：v1.9.0 • registry.k8s.io/sig-storage/csi-snapshotter：v6.3.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.9.0 • docker.io/netapp/trident-operator：24.02.0（オプション）

Kubernetes のバージョン	コンテナイメージ
v1.27.0	• Docker .io / NetApp / Trident：24.02.0 • docker.io / netapp/trident-autosupport：24.02 • registry.k8s.io/sig-storage/csi-provisioner：v3.6.0 • registry.k8s.io/sig-storage/csi-attacher：v4.4.0 • registry.k8s.io/sig-storage/csi-resizer：v1.9.0 • registry.k8s.io/sig-storage/csi-snapshotter：v6.3.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.9.0 • docker.io/netapp/trident-operator：24.02.0（オプション）
v1.28.0	• Docker .io / NetApp / Trident：24.02.0 • docker.io / netapp/trident-autosupport：24.02 • registry.k8s.io/sig-storage/csi-provisioner：v3.6.0 • registry.k8s.io/sig-storage/csi-attacher：v4.4.0 • registry.k8s.io/sig-storage/csi-resizer：v1.9.0 • registry.k8s.io/sig-storage/csi-snapshotter：v6.3.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.9.0 • docker.io/netapp/trident-operator：24.02.0（オプション）
v1.29.0	• Docker .io / NetApp / Trident：24.02.0 • docker.io / netapp/trident-autosupport：24.02 • registry.k8s.io/sig-storage/csi-provisioner：v4.0.0 • registry.k8s.io/sig-storage/csi-attacher：v4.5.0 • registry.k8s.io/sig-storage/csi-resizer：v1.9.3 • registry.k8s.io/sig-storage/csi-snapshotter：v6.3.3 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.10.0 • docker.io/netapp/trident-operator：24.02.0 （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。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24.02

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

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

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

作業を開始する前に

インストールパスに関係なく、次のものが必要です。

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

に慣れていない場合は "基本概念"今こそ、そのための絶好の機会です。

インストール方法を選択します

適切なインストール方法を選択します。また、に関する考慮事項についても確認しておく必要があります "メ

ソッド間を移動しています" 決定する前に。

Trident演算子を使用する

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

Tridentオペレータには次のようなメリットがあります。

** Astra Trident**オブジェクト作成****

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

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

****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。A `trident-operator-<generated-id>` ポッドが作成され、が関連付けられます `TridentOrchestrator` Astra TridentをインストールしたCR。これにより、クラスタ内にAstra Tridentのインスタンスが1つだけ存在し、そのセットアップを制御することで、インストールがべき等の状態であることを確認できます。インストールに変更が加えられると（展開またはノードのデミスタなど）、オペレータはそれらを識別し、個別に修正します。

** は、インストール済みの既存の を簡単に更新できます**

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

たとえば、 Astra Trident を有効にしてデバッグログを生成する必要があるシナリオを考えてみましょう。これを行うには、にパッチを適用します `TridentOrchestrator` をクリックして設定します `spec.debug` 終了： `true`：

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

実行後 `TridentOrchestrator` が更新され、オペレータが既存のインストールの更新とパッチを処理します。これにより、新しいポッドが作成され、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。

BlueXP（旧Cloud Manager） を使用した Kubernetesクラスタ管理

を使用 "Astra TridentでBlueXPを使用"では、最新バージョンのAstra Tridentにアップグレードし、ストレージクラスを追加して管理し、作業環境に接続し、Cloud Backup Service を使用して永続的ボリュームをバックアップすることができます。BlueXPは、Tridentオペレータを使用したAstra Tridentの導入を、手動またはHelmを使用してサポートしています。

を使用します `tridentctl`

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

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

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

インストールモードを選択します

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

標準インストール

これは、Astra Tridentをインストールする最も簡単な方法であり、ネットワークの制限を課すことのないほとんどの環境で機能します。標準インストールモードでは、必要なTridentを格納するためにデフォルトのレジストリが使用されます (`docker.io`）とCSIを参照してください (`registry.k8s.io`）イメージ。

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

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

オフラインインストール

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

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

リモートインストール

次に、リモートインストールプロセスの概要を示します。

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

メソッドとモードに基づいてプロセスを選択します

決定が終わったら、適切なプロセスを選択します。

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

インストール方法を切り替える

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

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

その他の既知の設定オプション

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

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

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

Tridentオペレータを使用してインストール

Tridentオペレータを手動で導入（標準モード）

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

Astra Tridentに関する重要な情報**24.02**

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

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

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

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

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

作業を開始する前に

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

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

1. Kubernetesのバージョンを確認します。

```
kubectl version
```

2. クラスタ管理者の権限を確認します。

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

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

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

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

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

```
wget https://github.com/NetApp/trident/releases/download/v24.02.0/trident-
installer-24.02.0.tar.gz
tar -xf trident-installer-24.02.0.tar.gz
cd trident-installer
```

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

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

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

手順**3**：**Trident**のオペレータを導入する

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

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

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

作業を開始する前に

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

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

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

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

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

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

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

手順

1. リソースを作成し、オペレータを配置します。

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

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

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

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

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

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

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

kubectl describe torc trident

Name:        trident
Namespace:
Labels:      <none>
Annotations: <none>
API Version: trident.netapp.io/v1
Kind:        TridentOrchestrator
...
Spec:
  Debug:     true
  Namespace: trident
Status:
  Current Installation Params:
    IPv6:                      false
    Autosupport Hostname:
    Autosupport Image:         netapp/trident-autosupport:24.02
    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:24.02.0
  Message:                  Trident installed  Namespace:
trident
  Status:                   Installed
  Version:                  v24.02.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

インストールを確認します。

インストールを確認するには、いくつかの方法があります。

を使用します `TridentOrchestrator` ステータス

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

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

ポッドの作成ステータスを使用する

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

```
kubectl get pods -n trident

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

を使用します `tridentctl`

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

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 24.02.0        | 24.02.0        |
+----------------+----------------+
```

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、を使用します "標準的な導入のプロセス"。

Astra Tridentに関する重要な情報**24.02**

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

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

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

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

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

作業を開始する前に

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

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

1. Kubernetesのバージョンを確認します。

```
kubectl version
```

2. クラスタ管理者の権限を確認します。

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

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

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

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

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

```
wget https://github.com/NetApp/trident/releases/download/v24.02.0/trident-
installer-24.02.0.tar.gz
tar -xf trident-installer-24.02.0.tar.gz
cd trident-installer
```

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

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

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

手順**3**：オペレータのレジストリの場所を更新します

インチ `/deploy/operator.yaml`、を更新します `image: docker.io/netapp/trident-operator:24.02.0` イメージレジストリの場所を反映します。。 "TridentとCSIの画像" 1つのレジストリまたは別のレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。例：

- `image: <your-registry>/trident-operator:24.02.0` すべての画像が1つのレジストリにある場合。

- `image: <your-registry>/netapp/trident-operator:24.02.0` TridentイメージがCSIイメージとは別のレジストリにある場合。

ステップ**4**：**Trident**オペレータを導入

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

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

作業を開始する前に

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

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

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

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

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

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

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

手順

1. リソースを作成し、オペレータを配置します。

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

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

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

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

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

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

1つのレジストリ内のイメージ

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:24.02"
tridentImage: "<your-registry>/trident:24.02.0"
```

異なるレジストリ内の画像

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

```
imageRegistry: "<your-registry>/sig-storage"
autosupportImage: "<your-registry>/netapp/trident-autosupport:24.02"
tridentImage: "<your-registry>/netapp/trident:24.02.0"
```

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

これで、を作成できます `TridentOrchestrator` Astra Tridentを導入必要に応じて、さらに行うことができます "Tridentのインストールをカスタマイズ" で属性を使用する `TridentOrchestrator` 仕様次の例は、TridentイメージとCSIイメージが異なるレジストリにあるインストールを示しています。

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

kubectl describe torc trident

Name:        trident
Namespace:
Labels:      <none>
Annotations: <none>
API Version: trident.netapp.io/v1
Kind:        TridentOrchestrator
...
Spec:
  Autosupport Image:  <your-registry>/netapp/trident-autosupport:24.02
  Debug:              true
  Image Registry:     <your-registry>/sig-storage
  Namespace:          trident
  Trident Image:      <your-registry>/netapp/trident:24.02.0
Status:
  Current Installation Params:
    IPv6:                       false
    Autosupport Hostname:
    Autosupport Image:          <your-registry>/netapp/trident-
autosupport:24.02
    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:24.02.0
  Message:                Trident installed
  Namespace:              trident
  Status:                 Installed
  Version:                v24.02.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

インストールを確認します。

インストールを確認するには、いくつかの方法があります。

を使用します `TridentOrchestrator` ステータス

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

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

ポッドの作成ステータスを使用する

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

```
kubectl get pods -n trident

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

を使用します `tridentctl`

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

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 24.02.0        | 24.02.0        |
+----------------+----------------+
```

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、を使用します "オフライン導入のプロセス"。

Astra Tridentに関する重要な情報**24.02**

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

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

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

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

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

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

作業を開始する前に

に加えて "導入の前提条件" 必要です "Helm バージョン 3"。

手順

1. Astra Trident Helmリポジトリを追加します。

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

2. 使用 `helm install` をクリックし、次の例に示すように、導入環境の名前を指定します `100.2402.0` 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 100.2402.0
--create-namespace --namespace <trident-namespace>
```

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

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

インストール中に設定データを渡す

インストール中に設定データを渡すには、次の 2 つの方法があります。

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

たとえば、のデフォルト値を変更するには、のように指定します `debug`をクリックし、次のコマンドを実行します `--set` コマンドを入力します `100.2402.0` 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 100.2402.0
--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。を参照してください [コントローラポッドとノードポッドについて] を参照してください。	
`tridentControllerPluginTolerations`	ポッドに対するKubernetesの許容範囲を上書きします。を参照してください [コントローラポッドとノードポッドについて] を参照してください。	
`tridentNodePluginNodeSelector`	ポッド用の追加のノードセレクタ。を参照してください [コントローラポッドとノードポッドについて] を参照してください。	
`tridentNodePluginTolerations`	ポッドに対するKubernetesの許容範囲を上書きします。を参照してください [コントローラポッドとノードポッドについて] を参照してください。	
「 imageRegistry 」と入力します	のレジストリを指定します `trident-operator`、`trident`、およびその他の画像。デフォルトをそのまま使用する場合は、空のままにします。	""
`imagePullPolicy`	のイメージプルポリシーを設定します `trident-operator`。	`IfNotPresent`

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

オプション	説明	デフォルト
「 tridentImage 」のように入力します	Astra Tridentのイメージを完全に上書きできます。	""
`tridentImageTag`	Astra Tridentのイメージのタグを上書きできます。	""
`tridentProbePort`	Kubernetesの活性/準備プローブに使用されるデフォルトポートを上書きできます。	""
`windows`	WindowsワーカーノードにAstra Tridentをインストールできます。	「偽」
`enableForceDetach`	強制切り離し機能を有効にできます。	「偽」
`excludePodSecurityPolicy`	オペレータポッドのセキュリティポリシーを作成から除外します。	「偽」
`cloudProvider`	をに設定します "`Azure`" AKSクラスタで管理対象IDまたはクラウドIDを使用する場合。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。	""

コントローラポッドとノードポッドについて

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

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

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

Helm（オフラインモード）を使用した**Trident**のオペレータの導入

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

Astra Tridentに関する重要な情報**24.02**

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

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

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

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

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

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

作業を開始する前に

に加えて "導入の前提条件" 必要です "Helm バージョン 3"。

手順

1. Astra Trident Helmリポジトリを追加します。

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

2. 使用 `helm install` 展開およびイメージレジストリの場所の名前を指定します。。 "TridentとCSIの画像" 1つのレジストリまたは別のレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。例では、 `100.2402.0` は、インストールするAstra Tridentのバージョンです。

1つのレジストリ内のイメージ

```
helm install <name> netapp-trident/trident-operator --version
100.2402.0 --set imageRegistry=<your-registry> --create-namespace
--namespace <trident-namespace>
```

異なるレジストリ内の画像

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

```
helm install <name> netapp-trident/trident-operator --version
100.2402.0 --set imageRegistry=<your-registry>/sig-storage --set
operatorImage=<your-registry>/netapp/trident-operator:24.02.0 --set
tridentAutosupportImage=<your-registry>/netapp/trident-
autosupport:24.02 --set tridentImage=<your-
registry>/netapp/trident:24.02.0 --create-namespace --namespace
<trident-namespace>
```

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

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

インストール中に設定データを渡す

インストール中に設定データを渡すには、次の 2 つの方法があります。

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

たとえば、のデフォルト値を変更するには、のように指定します `debug`をクリックし、次のコマンドを実行します `--set` コマンドを入力します `100.2402.0` 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 100.2402.0
--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。を参照してください "コントローラポッドとノードポッドについて" を参照してください。	
`tridentControllerPluginTolerations`	ポッドに対するKubernetesの許容範囲を上書きします。を参照してください "コントローラポッドとノードポッドについて" を参照してください。	
`tridentNodePluginNodeSelector`	ポッド用の追加のノードセレクタ。を参照してください "コントローラポッドとノードポッドについて" を参照してください。	
`tridentNodePluginTolerations`	ポッドに対するKubernetesの許容範囲を上書きします。を参照してください "コントローラポッドとノードポッドについて" を参照してください。	
「 imageRegistry 」と入力します	のレジストリを指定します `trident-operator`、`trident`、およびその他の画像。デフォルトをそのまま使用する場合は、空のままにします。	""
`imagePullPolicy`	のイメージプルポリシーを設定します `trident-operator`。	`IfNotPresent`
「 imagePullSecrets 」	のイメージプルシークレットを設定します `trident-operator`、`trident`、およびその他の画像。	
「 kubeletDir 」を参照してください	kubeletの内部状態のホスト位置を上書きできます。	`"/var/lib/kubelet"`
`operatorLogLevel`	Tridentオペレータのログレベルを次のように設定できます。`trace`、 `debug`、 `info`、 `warn`、`error`または `fatal`。	`"info"`

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

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

Tridentオペレータのインストールをカスタマイズ

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

コントローラポッドとノードポッドについて

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

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

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

設定オプション

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

このテーブルの詳細 `TridentOrchestrator` 属性。

パラメータ	説明	デフォルト
`namespace`	Astra Trident をインストールするネームスペース	`"default"`
「バグ」	Astra Trident のデバッグを有効にします	「偽」
`enableForceDetach`	`ontap-san` および `ontap-san-economy` のみ。 KubernetesのNon-Graceful Node Shutdown（NGN）と連携して、ノードに障害が発生した場合に、マウントされたボリュームを含むワークロードを新しいノードに安全に移行する機能をクラスタ管理者に提供します。	「偽」
`windows`	をに設定します `true` Windowsワーカーノードへのインストールを有効にします。	「偽」
`cloudProvider`	をに設定します `"Azure"` AKSクラスタで管理対象IDまたはクラウドIDを使用する場合。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	いいえ
`k8sTimeout`	Kubernetes 処理のタイムアウト	`30sec`
`silenceAutosupport`	AutoSupportバンドルをNetAppに送信しない 自動	「偽」
「 autosupportImage 」を参照してください	AutoSupport テレメトリのコンテナイメージ	`"netapp/trident-autosupport:24.02"`
「 autosupportProxy 」と入力します	AutoSupportを送信するためのプロキシのアドレス/ポート テレメータ	`"http://proxy.example.com:8888"`
uninstall	Astra Trident のアンインストールに使用するフラグ	「偽」
`logFormat`	Astra Trident のログ形式が使用 [text 、 JSON]	`"text"`
「 tridentImage 」のように入力します	インストールする Astra Trident イメージ	`"netapp/trident:24.02"`
「 imageRegistry 」と入力します	形式の内部レジストリへのパス `<registry FQDN>[:port][/subpath]`	`"k8s.gcr.io/sig-storage"` （Kubernetes 1.19以降） または `"quay.io/k8scsi"`
「 kubeletDir 」を参照してください	ホスト上の kubelet ディレクトリへのパス	`"/var/lib/kubelet"`

パラメータ	説明	デフォルト
`wipeout`	完全な削除を実行するために削除するリソースのリスト Astra Trident	
「imagePullSecrets」	内部レジストリからイメージをプルするシークレット	
`imagePullPolicy`	Tridentオペレータのイメージプルポリシーを設定します。有効な値は次のとおりです。 `Always` 常にイメージをプルする。 `IfNotPresent` ノード上にイメージが存在しない場合にのみ取得します。 `Never` 画像を絶対に引き出さないでください。	`IfNotPresent`
controller'PluginNodeSelector	ポッド用の追加のノードセレクタ。 の形式はと同じです `pod.spec.nodeSelector`。	デフォルトはありません。オプションです
controllerPluginTolerations	ポッドに対するKubernetesの許容範囲を上書きします。はと同じ形式です `pod.spec.Tolerations`。	デフォルトはありません。オプションです
「nodePluginNodeSelector」	ポッド用の追加のノードセレクタ。の形式はと同じです `pod.spec.nodeSelector`。	デフォルトはありません。オプションです
「nodePluginTolerations`」	ポッドに対するKubernetesの許容範囲を上書きします。はと同じ形式です `pod.spec.Tolerations`。	デフォルトはありません。オプションです

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

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

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

Kubernetesクラスタ管理者がを適用したとき `node.kubernetes.io/out-of-service=nodeshutdown:NoExecute` ノードおよびに影響を与えます `enableForceDetach` がに設定されます `true`Astra Tridentがノードのステータスを判断し、次の処理を実行します。

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

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

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

1. ノード上の古い公開パスを特定してクリーンアップします。
2. ノードがに含まれている場合 `cleanable` 状態（out-of-service taintが削除され、ノードがinになっています `Ready` 状態）。古い公開済みパスはすべてクリーンで、Astra Tridentはノードをとして再登録します `clean` 新しいボリュームのノードへの公開を許可します。

構成例

次の属性を使用できます： [設定オプション] テイギスルバアイ `TridentOrchestrator` をクリックして、インストールをカスタマイズします。

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

Astra Tridentに関する重要な情報24.02

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

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

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

を使用して**Astra Tridentをインストールします** `tridentctl`

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

作業を開始する前に

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

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

1. Kubernetesのバージョンを確認します。

```
kubectl version
```

2. クラスタ管理者の権限を確認します。

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

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

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

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

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

```
wget https://github.com/NetApp/trident/releases/download/v24.02.0/trident-
installer-24.02.0.tar.gz
tar -xf trident-installer-24.02.0.tar.gz
cd trident-installer
```

手順**2**：**Astra Trident**をインストールする

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

標準モード

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

1つのレジストリ内のイメージ

```
./tridentctl install -n trident --image-registry <your-registry>
--autosupport-image <your-registry>/trident-autosupport:24.02 --trident
-image <your-registry>/trident:24.02.0
```

異なるレジストリ内の画像

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

```
./tridentctl install -n trident --image-registry <your-registry>/sig-
storage --autosupport-image <your-registry>/netapp/trident-
autosupport:24.02 --trident-image <your-
registry>/netapp/trident:24.02.0
```

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=24.02.0
INFO Trident installation succeeded.
....
```

インストールを確認します。

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

ポッドの作成ステータスを使用する

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

```
kubectl get pods -n trident

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

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

を使用します `tridentctl`

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

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 24.02.0        | 24.02.0        |
+----------------+----------------+
```

構成例

以下に、Astra Tridentをインストールするための設定例を示します。 `tridentctl`。

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

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

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

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

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

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

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

iSCSIツール

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

NVMeツール

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

> 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。

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

Astra Tridentの自動修復機能は、次のような問題を防止します。

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

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

- iSCSIセッションがありません
- LUNが見つかりません

iSCSIツールをインストール

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

作業を開始する前に

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

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

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

> 「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。

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

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

6. 'iSCSI' を有効にして開始します

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

Ubuntu

1. 次のシステムパッケージをインストールします。

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

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

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

3. スキャンを手動に設定：

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

4. マルチパスを有効化：

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

> 「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。

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

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

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

> iSCSI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。

NVMe/TCPホリユウム

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

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

RHEL 9

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

Ubuntu

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

インストールを確認します

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

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

Astra Tridentは、 `ctrl_device_tmo` NVMeがダウンしてもパスを諦めないようにするための値。この設定は変更しないでください。

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"

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

Azure NetApp Files の特長

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 」と入力します	NFS SMB	ファイルシステム	RWO、ROX、RWX、RWOP	`nfs`、 `smb`

考慮事項

- 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ボリュームのみをサポート

AKSの管理対象ID

Astra Tridentのサポート "管理対象ID" （Azure Kubernetes Servicesクラスタの場合）。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
- Astra Tridentをインストール（以下を含む） `cloudProvider` 指定するには `"Azure"`。

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、`tridentorchestrator_cr.yaml` をクリックして設定します `cloudProvider` 終了：`"Azure"`。例：

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

Helm

次の例は、Astra Tridentセットをインストールします。 `cloudProvider` 環境変数を使用してAzureに移行 `$CP`：

```
helm install trident trident-operator-100.2402.0.tgz --create
-namespace --namespace <trident-namespace> --set cloudProvider=$CP
```

<code>tridentctl</code>

次の例では、Astra Tridentをインストールして `cloudProvider` フラグの対象 `Azure`：

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

AKSのクラウドID

クラウドIDを使用すると、Kubernetesポッドは、明示的なAzure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ID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クラスタに設定されたワークロードIDとoidc-issuer
- Astra Tridentをインストール（以下を含む） `cloudProvider` 指定するには `"Azure"` および `cloudIdentity` ワークロードIDの指定

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、`tridentorchestrator_cr.yaml` をクリックして設定します `cloudProvider` 終了： `"Azure"` をクリックして設定します `cloudIdentity` 終了： `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`。

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-
xxxxxxxxxxx"
```

次の例では、Astra Tridentとセットをインストールします。 `cloudProvider` 環境変数を使用してAzureに移行 `$CP` をクリックすると、 `cloudIdentity` 環境変数の使用 `$CI`：

```
helm install trident trident-operator-100.2402.0.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-
xxxxxxxxxxx"
```

次の例では、Astra Tridentをインストールして `cloud-provider` フラグの対象 `$CP``および `cloud-identity` 終了： `$CI`：

```
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"。

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

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

- 容量プール。を参照してください "Microsoft：Azure NetApp Files 用の容量プールを作成します"。
- Azure NetApp Files に委任されたサブネット。を参照してください "Microsoft：サブネットをAzure NetApp Files に委任します"。
- Azure NetApp Files が有効な Azure サブスクリプションのスクリプト ID 。
- `tenantID`、 `clientID`および `clientSecret` から "アプリケーション登録" Azure Active Directory で、 Azure NetApp Files サービスに対する十分な権限がある。アプリケーション登録では、次のいずれかを使用します。
  - オーナーまたは寄与者のロール "Azureで事前定義"。
  - A "カスタム投稿者ロール" をサブスクリプションレベルで選択します (`assignableScopes`)以下のアクセス許可は、Astra Tridentが必要とするものに限定されます。カスタムロールを作成したあと、"Azureポータルを使用してロールを割り当てます"。

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": []
            }
        ]
    }
}
```

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

SMBボリュームに関するその他の要件

SMBボリュームを作成するには、以下が必要です。

- Active Directoryが設定され、Azure NetApp Files に接続されています。を参照してください "Microsoft：Azure NetApp Files のActive Directory接続を作成および管理します"。
- LinuxコントローラノードとLinuxコントローラノードと少なくとも1つのWindowsワーカーノードでWindows Server 2019を実行しているKubernetesクラスタ。Astra Tridentは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみをサポート
- Azure NetApp Files がActive Directoryに対して認証できるように、Active Directoryクレデンシャルを含むAstra Tridentのシークレットが少なくとも1つ含まれています。シークレットを生成します `smbcreds`：

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

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

Azure NetApp Files バックエンド構成のオプションと例

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
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	「 azure-NetApp-files 」
backendName`	カスタム名またはストレージバックエンド	ドライバ名 + "_" + ランダムな文字
' スクリプト ID' 。	Azure サブスクリプションのサブスクリプション ID AKSクラスタで管理IDが有効になっている場合はオプションです。	
「 tenantID 」。	アプリケーション登録からのテナント ID AKSクラスタで管理IDまたはクラウドIDを使用する場合はオプションです。	
「 clientID 」。	アプリケーション登録からのクライアント ID AKSクラスタで管理IDまたはクラウドIDを使用する場合はオプションです。	
「 clientSecret 」を入力します。	アプリケーション登録からのクライアントシークレット AKSクラスタで管理IDまたはクラウドIDを使用する場合はオプションです。	

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

パラメータ	説明	デフォルト
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例 : `\{"API":false," メソッド ":true," 検出 ":true}`トラブルシューティングを行って詳細なログダンプが必要な場合を除き、このオプションは使用しないでください。	null
`nasType`	NFSボリュームまたはSMBボリュームの作成を設定オプションはです `nfs`、 `smb` またはnull。nullに設定すると、デフォルトでNFSボリュームが使用されます。	`nfs`

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。

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

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

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

ボリュームのプロビジョニング

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

パラメータ	説明	デフォルト
「 exportRule 」	新しいボリュームに対するエクスポートルール `exportRule` CIDR表記のIPv4アドレスまたはIPv4サブネットの任意の組み合わせをカンマで区切って指定する必要があります。SMBボリュームでは無視されます。	"0.0.0.0/0 "
「スナップショット方向」	.snapshot ディレクトリの表示を制御します	いいえ
「 size 」	新しいボリュームのデフォルトサイズ	" 100G "
「 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。理由 `nasType` は省略されています `nfs` デフォルトが適用され、バックエンドがNFSボリュームにプロビジョニングされます。

この構成は、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。

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
```

AKSの管理対象ID

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

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

AKSのクラウドID

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

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

容量プールフィルタを使用した特定のサービスレベル構成

このバックエンド構成では、Azureにボリュームが配置されます `eastus` の場所 `Ultra` 容量プール：Astra Tridentは、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。プールを区別するために、仮想プールのラベルを使用しました `performance`。

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

ストレージクラスの定義

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

を使用した定義の例 `parameter.selector` フィールド

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

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

SMBボリュームの定義例

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

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でフィルタリングします。 `nasType: nfs` または `nasType: null` NFSプールに対してフィルタを適用します。

バックエンドを作成します

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs
```

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

Google Cloudバックエンド用に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の特長 `gcp-cvs` クラスタと通信するドライバ。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

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

- * CVS - Performance *：デフォルトのAstra Tridentサービスタイプ。パフォーマンスが最適化されたこのサービスタイプは、パフォーマンスを重視する本番環境のワークロードに最適です。CVS -パフォーマンスサービスタイプは、サイズが100GiB以上のボリュームをサポートするハードウェアオプションです。のいずれかを選択できます "3つのサービスレベル"：
  - `standard`
  - `premium`
  - `extreme`
- * CVS *：CVSサービスタイプは、中程度のパフォーマンスレベルに制限された高レベルの可用性を提供

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

- `standardsw`
- `zoneredundantstandardsw`

必要なもの

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

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

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

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

ボリュームのプロビジョニングオプション

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

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

CVS -パフォーマンスサービスの種類の例

次の例は、CVS -パフォーマンスサービスタイプの設定例を示しています。

例 **1** ：最小限の構成

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

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

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

例2：サービスレベルの設定

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

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

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

例3：仮想プールの構成

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

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

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

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

```
  serviceLevel: standard
```

ストレージクラスの定義

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

ストレージクラスの例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-extreme-extra-protection
provisioner: 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
```

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

CVSサービスタイプの例

次の例は、CVSサービスタイプの設定例を示しています。

例1：最小構成

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

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

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

例**2**：ストレージプールの構成

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

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

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

次の手順

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs
```

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

NetApp HCI または SolidFire バックエンドを設定します

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の特長 `solidfire-san` クラスタと通信するためのストレージドライバ。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。をクリックします `Filesystem` volumeMode、Astra Tridentがボリュームを作成し、ファイルシステムを作成ファイルシステムのタイプは StorageClass で指定されます。

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされるファイルシステム
「 olidfire -san 」	iSCSI	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムがありません。raw ブロックデバイスです。
「 olidfire -san 」	iSCSI	ファイルシステム	RWO、RWOP	「 xfs 」、「 ext3 」、「 ext4 」

作業を開始する前に

Elementバックエンドを作成する前に、次の情報が必要になります。

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	常に「 solidfire-san-」
backendName`	カスタム名またはストレージバックエンド	「 iSCSI_ 」 + ストレージ（ iSCSI ） IP アドレス SolidFire
「エンドポイント」	テナントのクレデンシャルを使用する SolidFire クラスタの MVIP	
「 VIP 」	ストレージ（ iSCSI ）の IP アドレスとポート	
「ラベル」	ボリュームに適用する任意の JSON 形式のラベルのセット。	「」
「 tenantname 」	使用するテナント名（見つからない場合に作成）	
「 InitiatorIFCace 」	iSCSI トラフィックを特定のホストインターフェイスに制限します	デフォルト
UseCHAP'	CHAPを使用してiSCSIを認証します。Astra TridentはCHAPを使用	正しいです
「アクセスグループ」	使用するアクセスグループ ID のリスト	「 trident 」という名前のアクセスグループの ID を検索します。
「タイプ」	QoS の仕様	

パラメータ	説明	デフォルト
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します	“”（デフォルトでは適用されません）
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例： {"API" ： false 、 "method" ： true}	null

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

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

次の例は、 CHAP 認証を使用するバックエンドファイルと、特定の QoS 保証を適用した 3 つのボリュームタイプのモデリングを示しています。その場合 ' ストレージ・クラスを定義して 'iops`storage クラス・パラメータを使用します

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

例**2**：のバックエンドとストレージクラスの設定 `solidfire-san` 仮想プールを備えたドライバ

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

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

以下に示すバックエンド定義ファイルの例では、すべてのストレージプールに対して特定のデフォルトが設定されています。これにより、が設定されます `type` シルバー。仮想プールは、で定義されます `storage` セクション。この例では、一部のストレージプールが独自のタイプを設定し、一部のプールが上記のデフォルト値を上書きします。

```
---
version: 1
storageDriverName: solidfire-san
Endpoint: https://<user>:<password>@<mvip>/json-rpc/8.0
SVIP: "<svip>:3260"
TenantName: "<tenant>"
UseCHAP: true
Types:
- Type: Bronze
  Qos:
    minIOPS: 1000
    maxIOPS: 2000
    burstIOPS: 4000
- Type: Silver
  Qos:
    minIOPS: 4000
    maxIOPS: 6000
    burstIOPS: 8000
- Type: Gold
  Qos:
    minIOPS: 6000
    maxIOPS: 8000
    burstIOPS: 10000
type: Silver
labels:
  store: solidfire
  k8scluster: dev-1-cluster
region: us-east-1
storage:
- labels:
    performance: gold
    cost: '4'
  zone: us-east-1a
```

```
  type: Gold
- labels:
    performance: silver
    cost: '3'
  zone: us-east-1b
  type: Silver
- labels:
    performance: bronze
    cost: '2'
  zone: us-east-1c
  type: Bronze
- labels:
    performance: silver
    cost: '1'
  zone: us-east-1d
```

次のStorageClass定義は、上記の仮想プールを参照しています。を使用する `parameters.selector` 各ストレージクラスは、ボリュームのホストに使用できる仮想プールを呼び出します。ボリュームには、選択した仮想プール内で定義された要素があります。

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

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

詳細については、こちらをご覧ください

- "ボリュームアクセスグループ"

ONTAP SAN ドライバ

ONTAP SANドライバの概要

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

ONTAP SANドライバの詳細

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、 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/FC を参照してください NVMe/TCPに関するその他の考慮事項。	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし。rawブロックデバイスです

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/FC を参照してください NVMe/TCPに関するその他の考慮事項。	ファイルシステム	RWO、RWOP ROXおよびRWXは、ファイルシステムボリュームモードでは使用できません。	「 xfs 」、「 ext3 」、「 ext4 」
「 ONTAP - SAN - エコノミー」	iSCSI	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし。rawブロックデバイスです
「 ONTAP - SAN - エコノミー」	iSCSI	ファイルシステム	RWO、RWOP ROXおよびRWXは、ファイルシステムボリュームモードでは使用できません。	「 xfs 」、「 ext3 」、「 ext4 」

Astra Controlドライバの互換性

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

- 使用 `ontap-san-economy` 永続的ボリュームの使用数が次の値よりも多いと予想される場合のみ "サポートされるONTAPの制限"。
- 使用 `ontap-nas-economy` 永続的ボリュームの使用数が次の値よりも多いと予想される場合のみ "サポートされるONTAPの制限" および `ontap-san-economy` ドライバは使用できません。
- 使用しないでください `ontap-nas-economy` データ保護、ディザスタリカバリ、モビリティのニーズが予想される場合。

ユーザ権限

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

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

ONTAP内で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、 `ontap-san` 以下を含むドライバー：

- 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（24.02）

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

バックエンドに**ONTAP SAN**ドライバを設定する準備をします

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

要件

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

複数のドライバを実行し、 1 つまたは複数のドライバを参照するストレージクラスを作成することもできます。たとえば 'ONTAP-SAN' ドライバを使用する「 -dev 」クラスと 'ONTAP-SAN-エコノミー 'one を使用する「デフォルト」クラスを設定できます

すべてのKubernetesワーカーノードに適切なiSCSIツールをインストールしておく必要があります。を参照してください "ワーカーノードを準備します" を参照してください。

ONTAPバックエンドの認証

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

- credential based ：必要な権限を持つ ONTAP ユーザのユーザ名とパスワード。ONTAP バージョンとの互換性を最大限に高めるために 'admin' または vsadmin などの事前定義されたセキュリティ・ログイン・ロールを使用することを推奨します
- 証明書ベース： Astra Trident は、バックエンドにインストールされた証明書を使用して ONTAP クラスタと通信することもできます。この場合、バックエンド定義には、 Base64 でエンコードされたクライアント証明書、キー、および信頼された CA 証明書（推奨）が含まれている必要があります。

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。「 admin 」や「 vsadmin 」など、事前定義された標準的な役割を使用することをお勧めします。これにより、今後のリリースの ONTAP との互換性が今後のリリースの Astra Trident で使用される機能 API が公開される可能性があります。カスタムのセキュリティログインロールは Astra Trident で作成して使用できますが、推奨されません。

バックエンド定義の例は次のようになります。

YAML

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

JSON

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

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

証明書ベースの認証を有効にします

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

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

一般的なワークフローは次の手順で構成されます。

手順

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

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

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

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

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

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

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

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

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

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

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

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

認証方法を更新するか、クレデンシャルをローテーションして

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

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

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

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

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

双方向 **CHAP** を使用して接続を認証します

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

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

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

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

動作の仕組み

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

- SVM で CHAP をセットアップします。
  - SVMのデフォルトのイニシエータセキュリティタイプがnone（デフォルトで設定）*で、*ボリュームに既存のLUNがない場合、Astra Tridentはデフォルトのセキュリティタイプを `CHAP` CHAPイニシエータとターゲットのユーザ名およびシークレットの設定に進みます。
  - SVM に LUN が含まれている場合、 Trident は SVM で CHAP を有効にしません。これにより、SVM にすでに存在するLUNへのアクセスが制限されなくなります。
- 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「 tridentbackend 」 CRD を作成し、 CHAP シークレットとユーザ名を Kubernetes シークレットとして保存します。このバックエンドの Astra Trident によって作成されたすべての PVS がマウントされ、 CHAP 経由で接続されます。

クレデンシャルをローテーションし、バックエンドを更新

CHAP 証明書を更新するには 'backend.json ファイルの CHAP パラメータを更新しますこれには 'CHAP シークレットを更新し 'tridentctl update コマンドを使用してこれらの変更を反映する必要があります

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

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

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

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

ONTAP の**SAN**構成オプションと例

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
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	`ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`
backendName`	カスタム名またはストレージバックエンド	ドライバ名+"_"+ dataLIF
「管理 LIF 」	クラスタ管理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、次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、を参照してください。 MetroClusterの例。	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」
「重複排除	プロトコル LIF の IP アドレス。 * iSCSIには指定しないでください。* Astra Tridentが使用します "ONTAP の選択的LUNマップ" iSCSI LIF を検出するには、マルチパスセッションを確立する必要があります。の場合は警告が生成されます `dataLIF` は明示的に定義されます。 * MetroClusterの場合は省略してください。* MetroClusterの例。	SVMの派生物です
'VM'	使用する Storage Virtual Machine * MetroClusterの場合は省略してください。* MetroClusterの例。	SVM 「管理 LIF 」が指定されている場合に生成されます
「 useCHAP 」	CHAPを使用してONTAP SANドライバのiSCSIを認証します（ブーリアン）。をに設定します `true` Astra Tridentでは、バックエンドで指定されたSVMのデフォルト認証として双方向CHAPを設定して使用します。を参照してください "バックエンドにONTAP SANドライバを設定する準備をします" を参照してください。	「偽」
「 chapInitiatorSecret 」	CHAP イニシエータシークレット。「 useCHAP = TRUE 」の場合は必須	""
「ラベル」	ボリュームに適用する任意の JSON 形式のラベルのセット	""

パラメータ	説明	デフォルト
「 chapTargetInitiatorSecret 」	CHAP ターゲットイニシエータシークレット。「 useCHAP = TRUE 」の場合は必須	""
「 chapUsername 」	インバウンドユーザ名。「 useCHAP = TRUE 」の場合は必須	""
「 chapTargetUsername 」	ターゲットユーザ名。「 useCHAP = TRUE 」の場合は必須	""
「 clientCertificate 」をクリックします	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""
「 clientPrivateKey 」	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""
「 trustedCacertifate 」	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます。	""
「ユーザ名」	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 」と入力します	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。	""
'VM'	使用する Storage Virtual Machine	SVM 「管理 LIF 」が指定されている場合に生成されます
'toragePrefix'	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。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`
「 AggreglimitateUsage 」と入力します	使用率がこの割合を超えている場合は、プロビジョニングが失敗します。NetApp ONTAP バックエンドにAmazon FSXを使用している場合は、指定しないでください `limitAggregateUsage`。提供された `fsxadmin` および `vsadmin` アグリゲートの使用状況を取得し、Astra Tridentを使用して制限するために必要な権限が含まれていない。	"" （デフォルトでは適用されません）
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。また、qtreeおよびLUNに対して管理するボリュームの最大サイズを制限します。	""（デフォルトでは適用されません）
'lunsPerFlexvol	FlexVol あたりの最大 LUN 数。有効な範囲は 50 、 200 です	`100`

パラメータ	説明	デフォルト
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例： {"api"：false、"method"：true} トラブルシューティングを行い、詳細なログダンプが必要な場合を除き、は使用しないでください。	`null`
「 useREST` 」	ONTAP REST API を使用するためのブーリアンパラメータ。* テクニカルプレビュー * `useREST` は、テクニカルプレビューとして提供されています。テスト環境では、本番環境のワークロードでは推奨されません。に設定すると `true`Astra Tridentは、`ONTAP REST API`を使用してバックエンドと通信します。この機能には`ONTAP 9.11.1`以降が必要です。また、使用する`ONTAP` ログインロールにはへのアクセス権が必要です `ontap` アプリケーション：これは事前定義されたによって満たされます `vsadmin` および `cluster-admin` ロール。 `useREST` は、MetroCluster ではサポートされていません。 `useREST` はNVMe/TCPに完全修飾されています。	「偽」
`sanType`	を使用して選択 `iscsi` iSCSIの場合または `nvme`（NVMe/TCPの場合）。	`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

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

パラメータ	説明	デフォルト
「平和の配分」	space-allocation for LUN のコマンドを指定します	"正しい"
「平和のための準備」を参照してください	スペースリザベーションモード：「none」（シン）または「volume」（シック）	"なし"
「ナプショットポリシー」	使用する Snapshot ポリシー	"なし"
「 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。	""

パラメータ	説明	デフォルト
「adaptiveQosPolicy 」を参照してください	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	""
「スナップショット予約」	Snapshot 用にリザーブされているボリュームの割合	次の場合は「0」`snapshotPolicy` は「none」、それ以外の場合は「」です。
'plitOnClone	作成時にクローンを親からスプリットします	いいえ
「暗号化」	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは「false」です。このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームが NAEに有効になります。詳細については、以下を参照してください。 "Astra TridentとNVEおよびNAEの相互運用性"。	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。	""
'ecurityStyle'	新しいボリュームのセキュリティ形式	`unix`
階層ポリシー	「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」

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'
```

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

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

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

1.1 は、 Astra Trident の 10% の追加料金で、 FlexVol のメタデータに対応します。「 napshotReserve 」=5% 、 PVC 要求 =5GiB の場合、ボリュームの合計サイズは 5.79GiB 、使用可能なサイズは 5.5GiB です。volume show コマンドは ' 次の例のような結果を表示する必要があります

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

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

最小限の設定例

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

> 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、 `ontap-san` ドライバ。

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

ONTAP SAN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>
```

MetroClusterの例

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

シームレスなスイッチオーバーとスイッチバックを実現するには、 `managementLIF` を省略します。 `dataLIF` および `svm` パラメータ例：

```
---
version: 1
storageDriverName: ontap-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）がに入力されます `backend.json` およびは、クライアント証明書、秘密鍵、信頼されたCA証明書のbase64エンコード値をそれぞれ取得します。

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

双方向**CHAP**の例

次の例では、 `useCHAP` をに設定します `true`。

ONTAP SAN CHAPの例

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

ONTAP SANエコノミー**CHAP**の例

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

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
```

仮想プールを使用するバックエンドの例

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

Astra Trident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が独自の `spaceReserve`、 `spaceAllocation`および `encryption` 値、および一部のプールはデフォルト値よりも優先されます。

ONTAP SANの例

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

ONTAP SANの経済性の例

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

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

NVMe/TCPの例

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

バックエンドを **StorageClasses** にマッピングします

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

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

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

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

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

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

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

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

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

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

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

- 。 `my-test-app-sc` StorageClassはにマッピングされます `testAPP` 内の仮想プール `ontap-san` ドライバ `sanType: nvme`。これは唯一のプールサービスです。 `testApp`。

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

Trident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、 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 - エコノミー」	NFS SMB	ファイルシステム	RWO、ROX、RWX、RWOP	""、 `nfs`、 `smb`
「 ONTAP-NAS-flexgroup 」	NFS SMB	ファイルシステム	RWO、ROX、RWX、RWOP	""、 `nfs`、 `smb`

Astra Controlドライバの互換性

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

> - 使用 `ontap-san-economy` 永続的ボリュームの使用数が次の値よりも多いと予想される場合のみ "サポートされるONTAPの制限"。
> - 使用 `ontap-nas-economy` 永続的ボリュームの使用数が次の値よりも多いと予想される場合のみ "サポートされるONTAPの制限" および `ontap-san-economy` ドライバは使用できません。
> - 使用しないでください `ontap-nas-economy` データ保護、ディザスタリカバリ、モビリティのニーズが予想される場合。

ユーザ権限

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

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

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

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

ONTAP NASドライバを使用してバックエンドを設定する準備をします

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

要件

- ONTAP バックエンドすべてに対して、 Astra Trident が SVM に少なくとも 1 つのアグリゲートを割り当てておく必要があります。
- 複数のドライバを実行し、どちらか一方を参照するストレージクラスを作成できます。たとえば、を使用するGoldクラスを設定できます `ontap-nas` ドライバとを使用するBronzeクラス `ontap-nas-economy` 1つ。
- すべてのKubernetesワーカーノードに適切なNFSツールをインストールしておく必要があります。を参照してください "こちらをご覧ください" 詳細：
- Astra Tridentは、Windowsノードで実行されているポッドにマウントされたsmbボリュームのみをサポートを参照してください SMBボリュームをプロビジョニングする準備をします を参照してください。

ONTAPバックエンドの認証

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

- Credential-based：このモードでは、ONTAPバックエンドに十分な権限が必要です。事前定義されたセキュリティログインロールに関連付けられたアカウントを使用することを推奨します。例： `admin` または `vsadmin` ONTAP のバージョンとの互換性を最大限に高めるため。
- 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。「 admin 」や「 vsadmin 」など、事前定義された標準的な役割を使用することをお勧めします。これにより、今後のリリースの ONTAP との互換性が今後のリリースの Astra Trident

で使用される機能 API が公開される可能性があります。カスタムのセキュリティログインロールは Astra Trident で作成して使用できますが、推奨されません。

バックエンド定義の例は次のようになります。

YAML

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

JSON

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

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

証明書ベースの認証を有効にします

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

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

一般的なワークフローは次の手順で構成されます。

手順

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

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

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

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

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

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

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

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

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

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

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

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

認証方法を更新するか、クレデンシャルをローテーションして

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

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

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

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

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

NFS エクスポートポリシーを管理します

Astra Trident は、 NFS エクスポートポリシーを使用して、プロビジョニングするボリュームへのアクセスを制御します。

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

- Astra Trident は、エクスポートポリシー自体を動的に管理できます。このモードでは、許容可能な IP アドレスを表す CIDR ブロックのリストをストレージ管理者が指定します。Astra Trident は、この範囲に含まれるノード IP をエクスポートポリシーに自動的に追加します。または、 CIDRs が指定されていない場

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

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

エクスポートポリシーを動的に管理

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 」は「 true 」に設定されています。これは、 Astra Trident が「 vm1 」 SVM のエクスポートポリシーを作成し、「 autoExportCIDRs 」アドレスブロックを使用してルールの追加と削除を処理することを示しています。たとえば、 UUID 403b5326-842-40dB-96d0-d83fb3f4daec と「 autoExportPolicy 」が「 true 」に設定されているバックエンドは、 SVM 上に「 trident-403b5326-842-40dB-96d0-d83f3f4daec 」という名前のエクスポートポリシーを作成します。

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

この例では '192.168.0.0/24 アドレス空間が提供されていますこのアドレス範囲に含まれる Kubernetes ノードの IP が、 Astra Trident が作成するエクスポートポリシーに追加されることを示します。Astra Trident は、実行されているノードを登録すると、ノードの IP アドレスを取得し、「 autoExportCIDRs 」で提供されているアドレスブロックと照合します。IP をフィルタリングすると、 Trident が検出したクライアント IP のエクスポートポリシールールを作成し、特定したノードごとに 1 つのルールが設定されます。

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

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

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

ノードが Kubernetes クラスタに追加されて Astra Trident コントローラに登録されると、既存のバックエンドのエクスポートポリシーが更新されます（バックエンドの「 autoExportCIDRs 」に指定されたアドレス範囲に含まれる場合）。

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

以前のバックエンドの場合は、を使用してバックエンドを更新します `tridentctl update backend` では、Astra Tridentがエクスポートポリシーを自動的に管理します。これにより、バックエンドのUUID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

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

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

作業を開始する前に

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

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

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

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

手順

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

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

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

   a. 必要に応じて、共有のディレクトリパス構造を作成します。

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

b. 指定したSVMに関連付けられているSMB共有を作成します。

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

c. 共有が作成されたことを確認します。

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

> を参照してください "SMB 共有を作成" 詳細については、

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

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

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。

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	「ontap-nas」、「ontap-nas-economy」、「ontap-nas-flexgroup」、「ontap-san」、「ontap-san-economy」
backendName`	カスタム名またはストレージバックエンド	ドライバ名+"_"+ dataLIF
「管理 LIF 」	クラスタ管理 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、次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、を参照してください。 MetroClusterの例。	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」
「重複排除	プロトコル LIF の IP アドレス。 を指定することを推奨します `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、次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。* MetroClusterの例。	指定されたアドレス、または指定されていない場合はSVMから取得されるアドレス（非推奨）
'VM'	使用する Storage Virtual Machine * MetroClusterの場合は省略してください。* MetroClusterの例。	SVM 「管理 LIF 」が指定されている場合に生成されます

パラメータ	説明	デフォルト
「autoExportPolicy」を参照してください	エクスポートポリシーの自動作成と更新を有効にします[ブーリアン]。を使用する `autoExportPolicy` および `autoExportCIDRs` ネットアップのAstra Tridentなら、エクスポートポリシーを自動的に管理できます。	いいえ
「 autoExportCI` 」	KubernetesのノードIPをフィルタリングするCIDRのリスト `autoExportPolicy` が有効になります。 を使用する `autoExportPolicy` および `autoExportCIDRs` ネットアップのAstra Tridentなら、エクスポートポリシーを自動的に管理できます。	["0.0.0.0/0","::/0"]'
「ラベル」	ボリュームに適用する任意の JSON 形式のラベルのセット	""
「clientCertificate」をクリックします	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""
「clientPrivateKey」	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""
「trustedCacertifate」	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます	""
「ユーザ名」	クラスタ / SVM に接続するためのユーザ名。クレデンシャルベースの認証に使用されます	
「 password 」と入力します	クラスタ / SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます	
'toragePrefix'	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。設定後に更新することはできません	"トライデント"
「AggreglimitateUsage」と入力します	使用率がこの割合を超えている場合は、プロビジョニングが失敗します。* Amazon FSX for ONTAP * には適用されません	"" （デフォルトでは適用されません）
「limitVolumeSize」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。また、qtreeおよびLUN用に管理するボリュームの最大サイズも制限します `qtreesPerFlexvol` オプションを使用すると、FlexVol あたりの最大qtree数をカスタマイズできます。	""（デフォルトでは適用されません）
'lunsPerFlexvol	FlexVol あたりの最大 LUN 数。有効な範囲は 50 、 200 です	"100"

パラメータ	説明	デフォルト
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例： {"api"：false、"method"：true} 使用しないでください `debugTraceFlags` トラブルシューティングを実行していて、詳細なログダンプが必要な場合を除きます。	null
`nasType`	NFSボリュームまたはSMBボリュームの作成を設定オプションはです `nfs`、 `smb` またはnull。nullに設定すると、デフォルトでNFSボリュームが使用されます。	`nfs`
「nfsvMountOptions 」のように入力します	NFSマウントオプションをカンマで区切ったリスト。Kubernetes永続ボリュームのマウントオプションは通常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"
`smbShare`	Microsoft管理コンソールまたはONTAP CLIを使用して作成されたSMB共有の名前、Astra TridentでSMB共有を作成できる名前、ボリュームへの共有アクセスを禁止する場合はパラメータを空白のままにすることができます。 オンプレミスのONTAPでは、このパラメータはオプションです。 このパラメータはAmazon FSx for ONTAPバックエンドで必須であり、空にすることはできません。	`smb-share`
「useREST` 」	ONTAP REST API を使用するためのブーリアンパラメータ。* テクニカルプレビュー * `useREST` は、テクニカルプレビューとして提供されています。テスト環境では、本番環境のワークロードでは推奨されません。に設定すると `true`Astra Tridentは、`ONTAP REST API`を使用してバックエンドと通信します。この機能には`ONTAP 9.11.1`以降が必要です。また、使用する`ONTAP` ログインロールにはへのアクセス権が必要です `ontap` アプリケーション：これは事前定義されたによって満たされます `vsadmin` および `cluster-admin` ロール。`useREST` は、MetroCluster ではサポートされていません。	いいえ

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

パラメータ	説明	デフォルト
「平和の配分」	space-allocation for LUN のコマンドを指定します	"正しい"
「平和のための準備」を参照してください	スペースリザベーションモード：「none」（シン）または「volume」（シック）	"なし"
「ナプショットポリシー」	使用する Snapshot ポリシー	"なし"
「 QOSPolicy 」	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	""
「 adaptiveQosPolicy 」を参照してください	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します。経済性に影響する ONTAP - NAS ではサポートされません。	""
「スナップショット予約」	Snapshot 用にリザーブされているボリュームの割合	次の場合は「0」 `snapshotPolicy` は「none」、それ以外の場合は「」です。
'plitOnClone	作成時にクローンを親からスプリットします	いいえ
「暗号化」	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは「false」です。このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームが NAEに有効になります。詳細については、以下を参照してください。 "Astra TridentとNVEおよびNAEの相互運用性"。	いいえ
階層ポリシー	「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）
「スナップショット方向」	にアクセスする権限を管理します。 `.snapshot` ディレクトリ	いいえ
「 exportPolicy 」と入力します	使用するエクスポートポリシー	デフォルト
'ecurityStyle'	新しいボリュームのセキュリティ形式。NFSのサポート `mixed` および `unix` セキュリティ形式SMBはをサポートします `mixed` および `ntfs` セキュリティ形式	NFSのデフォルトはです `unix` 。SMBのデフォルトはです `ntfs`。

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'
```

「 ONTAP-NAS' 」と「 ONTAP-NAS-flexgroups' では、 Astra Trident は新しい計算を使用して、 FlexVol がスナップショット予約の割合と PVC で正しくサイズ設定されるようにします。ユーザが PVC を要求すると、 Astra Trident は、新しい計算を使用して、より多くのスペースを持つ元の FlexVol を作成します。この計算により、ユーザは要求された PVC 内の書き込み可能なスペースを受信し、要求されたスペースよりも少ないスペースを確保できます。v21.07 より前のバージョンでは、ユーザが PVC を要求すると（ 5GiB など）、 snapshotReserve が 50% に設定されている場合、書き込み可能なスペースは 2.5GiB のみになります。これは、ユーザが要求したボリューム全体が「 SnapshotReserve 」であるためです。Trident 21.07 では、ユーザが要求するのは書き込み可能なスペースであり、 Astra Trident は「 napshotReserve 」の値をボリューム全体の割合で定義します。これは「 ONTAP-NAS-エコノミー 」には適用されません。この機能の仕組みについては、次の例を参照してください。

計算は次のとおりです。

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

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

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

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

最小限の設定例

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

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

ONTAP NASエコノミーの例

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

ONTAP NAS FlexGroupの例

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

MetroClusterの例

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

シームレスなスイッチオーバーとスイッチバックを実現するには、 `managementLIF` を省略します。 `dataLIF` および `svm` パラメータ例：

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

SMBボリュームの例

```
---
version: 1
backendName: ExampleBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
nasType: smb
securityStyle: ntfs
unixPermissions: ""
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
```

証明書ベースの認証の例

これは、バックエンドの最小限の設定例です。 `clientCertificate`、 `clientPrivateKey`および`trustedCACertificate` （信頼されたCAを使用している場合はオプション）がに入力されます `backend.json` およびは、クライアント証明書、秘密鍵、信頼されたCA証明書のbase64エンコード値をそれぞれ取得します。

```
---
version: 1
backendName: DefaultNASBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.15
svm: nfs_svm
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
storagePrefix: myPrefix_
```

自動エクスポートポリシーの例

この例は、動的なエクスポートポリシーを使用してエクスポートポリシーを自動的に作成および管理するように Astra Trident に指示する方法を示しています。これは、でも同様に機能します `ontap-nas-economy` および `ontap-nas-flexgroup` ドライバ。

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

IPv6アドレスの例

この例は、を示しています `managementLIF` IPv6アドレスを使用している。

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas_ipv6_backend
managementLIF: "[5c5d:5edf:8f:7657:bef8:109b:1b41:d491]"
labels:
  k8scluster: test-cluster-east-1a
  backend: test1-ontap-ipv6
svm: nas_ipv6_svm
username: vsadmin
password: password
```

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

。 `smbShare` SMBボリュームを使用するFSx for ONTAPの場合、パラメータは必須です。

```
---
version: 1
backendName: SMBBackend
storageDriverName: ontap-nas
managementLIF: example.mgmt.fqdn.aws.com
nasType: smb
dataLIF: 10.0.0.15
svm: nfs_svm
smbShare: smb-share
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
storagePrefix: myPrefix_
```

仮想プールを使用するバックエンドの例

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

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

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

ONTAP NASの例

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

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

ONTAP NAS FlexGroupの例

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

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

ONTAP NASエコノミーの例

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

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

バックエンドを **StorageClasses** にマッピングします

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

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

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

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

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

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

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

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

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

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

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: 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。

更新 `dataLIF` 初期設定後

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

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

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

NetApp ONTAP 対応の Amazon FSX

Amazon FSX for NetApp ONTAP で Astra Trident を使用

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

概要

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

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

考慮事項

- SMBボリューム：
  - SMBボリュームは、を使用してサポートされます `ontap-nas` ドライバーのみ。
  - SMBボリュームは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
- 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以降でこの問題を無効にするには、 `fsxFilesystemID`、AWS `apiRegion`、AWS `apikey`および`AWS `secretKey` AWS FSx for ONTAPのバックエンド構成ファイルに保存されます。

> Astra TridentにIAMロールを指定する場合は、 `apiRegion`、 `apiKey`および`secretKey` フィールドをAstra Tridentに明示的に追加詳細については、を参照してください "FSX（ONTAP の構成オプションと例）"。

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

ドライバーの詳細については、を参照してください。 "NASドライバ" および "SANドライバ"。

認証

Astra Tridentは、2種類の認証モードを提供します。

- 証明書ベース： Astra Trident は、 SVM にインストールされている証明書を使用して、 FSX ファイルシステムの SVM と通信します。
- 認証情報ベース：ファイルシステムには「 fsxadmin 」ユーザを、 SVM には「 vsadmin 」ユーザを使用できます。

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

証明書ベースの方法と証明書ベースの方法を切り替えるために、バックエンドを更新できます。ただし、*クレデンシャルと*証明書を入力しようとすると、バックエンドの作成に失敗します。別の認証方式に切り替えるには、バックエンド設定から既存の方式を削除する必要があります。

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

- "ONTAP NAS認証"
- "ONTAP SAN認証"

EKSのクラウドID

Cloud Identityを使用すると、Kubernetesポッドは、明示的なAWSクレデンシャルを指定するのではなく、AWS IAMロールとして認証することでAWSリソースにアクセスできます。

AWSでクラウドIDを利用するには、以下が必要です。

- EKSを使用して導入されるKubernetesクラスタ

- Astra Tridentをインストール（以下を含む） `cloudProvider` シテイ `"AWS"` および `cloudIdentity` AWS IAMロールの指定

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、`tridentorchestrator_cr.yaml` をクリックして設定します `cloudProvider` 終了： `"AWS"` をクリックして設定します `cloudIdentity` をAWS IAMロールに割り当てます。

例：

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

Helm

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::123456:role/astratrident-role'"
```

次の例では、Astra Tridentとセットをインストールします。 `cloudProvider` 終了： `AWS` 環境変数の使用 `$CP` 環境変数を使用して'cloudIdentity'を設定します `$CI`：

```
helm install trident trident-operator-100.2402.0.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="AWS"
export CI="'eks.amazonaws.com/role-arn:
arn:aws:iam::123456:role/astratrident-role'"
```

次の例では、Astra Tridentをインストールして `cloud-provider` フラグの対象 `$CP`` および `cloud-identity` 終了： `$CI`：

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

詳細については、こちらをご覧ください

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

NetApp ONTAP 向け**Amazon FSX**を統合します

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

要件

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

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

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

- Astra Tridentは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみをサポートを参照してください SMBボリュームをプロビジョニングする準備をします を参照してください。

ONTAP SANと**NAS**ドライバの統合

> SMBボリュームについて設定する場合は、を参照してください SMBボリュームをプロビジョニングする準備をします バックエンドを作成する前に。

手順

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

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

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

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

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

YAML

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

JSON

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

また、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）を使用してバックエンドファイルを作成することもできます。

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
  managementLIF:
  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、次のリンクを参照してください。

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

SMBボリュームをプロビジョニングする準備をします

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

作業を開始する前に

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

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

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

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

手順

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

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

   b. 指定したSVMに関連付けられているSMB共有を作成します。

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

   c. 共有が作成されたことを確認します。

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

を参照してください "SMB 共有を作成" 詳細については、

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

パラメータ	説明	例
`smbShare`	次のいずれかを指定できます。Microsoft管理コンソールまたはONTAP CLIを使用して作成されたSMB共有の名前、またはAstra TridentでSMB共有を作成できるようにする名前。 このパラメータは、Amazon FSx for ONTAPバックエンドに必要です。	`smb-share`
`nasType`	をに設定する必要があります **`smb`**. nullの場合、デフォルトはです `nfs`。	`smb`
'ecurityStyle'	新しいボリュームのセキュリティ形式。をに設定する必要があります **`ntfs`** または **`mixed` SMB**ボリューム	`ntfs` または `mixed` SMBボリュームの場合
「 unixPermissions 」	新しいボリュームのモード。* SMBボリュームは空にしておく必要があります。*	""

FSX（ONTAP の構成オプションと例）

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

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

パラメータ	説明	例
「管理 LIF 」	クラスタ管理 LIF または SVM 管理 LIF の IP アドレス Fully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定できます。 IPv6フラグを使用して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。	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」
「重複排除	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 」を参照してください	エクスポートポリシーの自動作成と更新を有効にします[ブーリアン]。を使用する `autoExportPolicy` および `autoExportCIDRs` ネットアップのAstra Tridentなら、エクスポートポリシーを自動的に管理できます。	「偽」
「 autoExportCI` 」	KubernetesのノードIPをフィルタリングするCIDRのリスト `autoExportPolicy` が有効になります。 を使用する `autoExportPolicy` および `autoExportCIDRs` ネットアップのAstra Tridentなら、エクスポートポリシーを自動的に管理できます。	「[0.0.0.0/0]、「::/0」」」
「ラベル」	ボリュームに適用する任意の JSON 形式のラベルのセット	""
「 clientCertificate 」をクリックします	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""
「 clientPrivateKey 」	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""
「 trustedCacertifate 」	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます。	""
「ユーザ名」	クラスタまたはSVMに接続するためのユーザ名。クレデンシャルベースの認証に使用されます。たとえば、vsadminのように指定します。	
「 password 」と入力します	クラスタまたはSVMに接続するためのパスワード。クレデンシャルベースの認証に使用されます。	
'VM'	使用する Storage Virtual Machine	SVM管理LIFが指定されている場合に生成されます。
'toragePrefix'	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。作成後に変更することはできません。このパラメータを更新するには、新しいバックエンドを作成する必要があります。	`trident`

パラメータ	説明	例
「 AggreglimitateUsage 」と入力します	* NetApp ONTAP にはAmazon FSXを指定しないでください。*提供されている `fsxadmin` および `vsadmin` アグリゲートの使用状況を取得し、Astra Tridentを使用して制限するために必要な権限が含まれていない。	使用しないでください。
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。また、qtreeおよびLUN用に管理するボリュームの最大サイズも制限します `qtreesPerFlexvol` オプションを使用すると、FlexVol あたりの最大qtree数をカスタマイズできます。	“”（デフォルトでは適用されません）
'lunsPerFlexvol	FlexVol あたりの最大LUN数。有効な範囲は50、200です。SANのみ。	`100`
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例：{"API"：false、"method"：true}は使用されません `debugTraceFlags` トラブルシューティングを実行していて、詳細なログダンプが必要な場合を除きます。	null
「 nfsvMountOptions 」のように入力します	NFSマウントオプションをカンマで区切ったリスト。Kubernetes永続ボリュームのマウントオプションは通常はストレージクラスで指定されますが、ストレージクラスでマウントオプションが指定されていない場合、Astra Tridentはストレージバックエンドの構成ファイルで指定されているマウントオプションを使用します。ストレージクラスや構成ファイルにマウントオプションが指定されていない場合、Astra Tridentは関連付けられた永続的ボリュームにマウントオプションを設定しません。	""
`nasType`	NFSボリュームまたはSMBボリュームの作成を設定オプションはです `nfs`、 `smb`、またはnull。*をに設定する必要があります `smb` SMBボリューム。*をnullに設定すると、デフォルトでNFSボリュームが使用されます。	`nfs`

パラメータ	説明	例
qtreesPerFlexvol`	FlexVol あたりの最大 qtree 数。有効な範囲は [50 、 300] です。	`200`
`smbShare`	次のいずれかを指定できます。Microsoft管理コンソールまたはONTAP CLIを使用して作成されたSMB共有の名前、またはAstra TridentでSMB共有を作成できるようにする名前。 このパラメータは、Amazon FSx for ONTAPバックエンドに必要です。	`smb-share`
「 useREST` 」	ONTAP REST API を使用するためのブーリアンパラメータ。* テクニカルプレビュー * `useREST` は、テクニカルプレビューとして提供されています。テスト環境では、本番環境のワークロードでは推奨されません。に設定すると `true`Astra Trident` は、`ONTAP REST API`を使用してバックエンドと通信します。この機能には`ONTAP 9.11.1`以降が必要です。また、使用する`ONTAP` ログインロールにはへのアクセス権が必要です `ontap` アプリケーション：これは事前定義されたによって満たされます `vsadmin` および `cluster-admin` ロール。	「偽」
`aws`	AWS FSx for ONTAPの構成ファイルでは、次の項目を指定できます。 - `fsxFilesystemID`：AWS FSxファイルシステムのIDを指定します。 - `apiRegion`：AWS APIリージョン名。 - `apikey`：AWS APIキー。 - `secretKey`：AWSシークレットキー。	"" "" ""

パラメータ	説明	例
`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 Resource Name（ARN）。SVMのクレデンシャルが含まれています。 - `type`:に設定 `awsarn`。 を参照してください "AWS Secrets Managerシークレットの作成" を参照してください。	

更新 `dataLIF` 初期設定後

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

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

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

ボリュームのプロビジョニング用のバックエンド構成オプション

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

パラメータ	説明	デフォルト
「平和の配分」	space-allocation for LUN のコマンドを指定します	「真」
「平和のための準備」を参照してください	スペースリザベーションモード：「 none 」（シン）または「 volume 」（シック）	「 NONE 」
「ナプショットポリシー」	使用する Snapshot ポリシー	「 NONE 」

パラメータ	説明	デフォルト
「 QOSPolicy 」	作成したボリュームに割り当てるQoS ポリシーグループ。ストレージプールまたはバックエンドごとに、QOSPolicyまたはadaptiveQosPolicyのいずれかを選択します。Trident 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 」を参照してください	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプールまたはバックエンドごとに、QOSPolicyまたはadaptiveQosPolicyのいずれかを選択します。経済性に影響するONTAP - NAS ではサポートされません。	「」
「スナップショット予約」	スナップショット "0" 用に予約されたボリュームの割合	状況 `snapshotPolicy` はです `none`、 `else` 「」
'plitOnClone	作成時にクローンを親からスプリットします	「偽」
「暗号化」	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは「false」です。このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームがに有効になります。詳細については、以下を参照してください。 "Astra TridentとNVEおよびNAEの相互運用性"。	「偽」
`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。	""
階層ポリシー	使用する階層化ポリシー `none`	`snapshot-only` ONTAP 9.5より前のSVM-DR構成の場合

パラメータ	説明	デフォルト
「 unixPermissions 」	新しいボリュームのモード。* SMBボリュームは空にしておきます。*	「」
'ecurityStyle'	新しいボリュームのセキュリティ形式。NFSのサポート `mixed` および `unix` セキュリティ形式SMBはをサポートします `mixed` および `ntfs` セキュリティ形式	NFSのデフォルトはです `unix` 。SMBのデフォルトはです `ntfs`。

構成例

SMBホリユウムノストレエシクラスノセツテイ

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

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

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
```

EKSクラスタでの**Astra Trident EKS**アドオンバージョン**23.10**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。

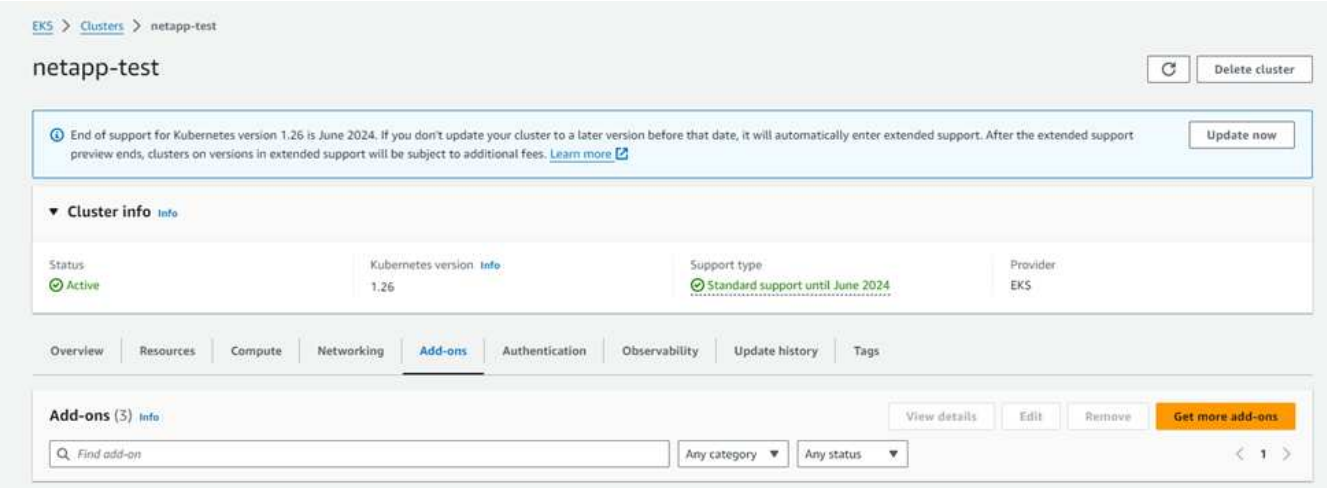
前提条件

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。

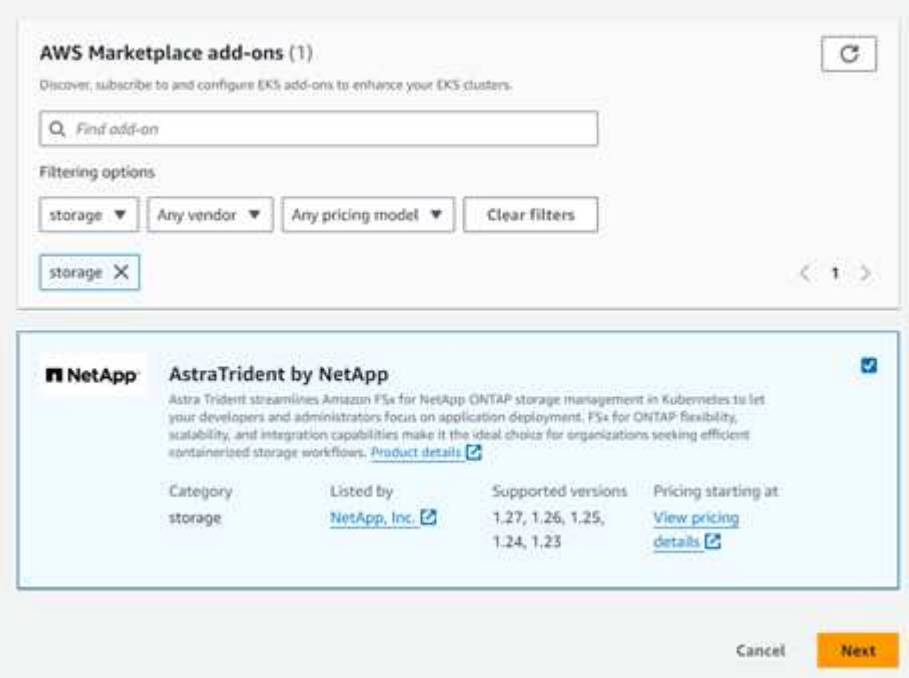
- アドオンサブスクリプションがあるAmazon EKSクラスタアカウント
- AWS MarketplaceへのAWS権限：
  `"aws-marketplace:ViewSubscriptions",`
  `"aws-marketplace:Subscribe",`
  `"aws-marketplace:Unsubscribe`
- AMIタイプ：Amazon Linux 2（AL2_x86_64）またはAmazon Linux 2 Arm（AL2_ARM_64）
- ノードタイプ：AMDまたはARM
- 既存のAmazon FSx for NetApp ONTAPファイルシステム

手順

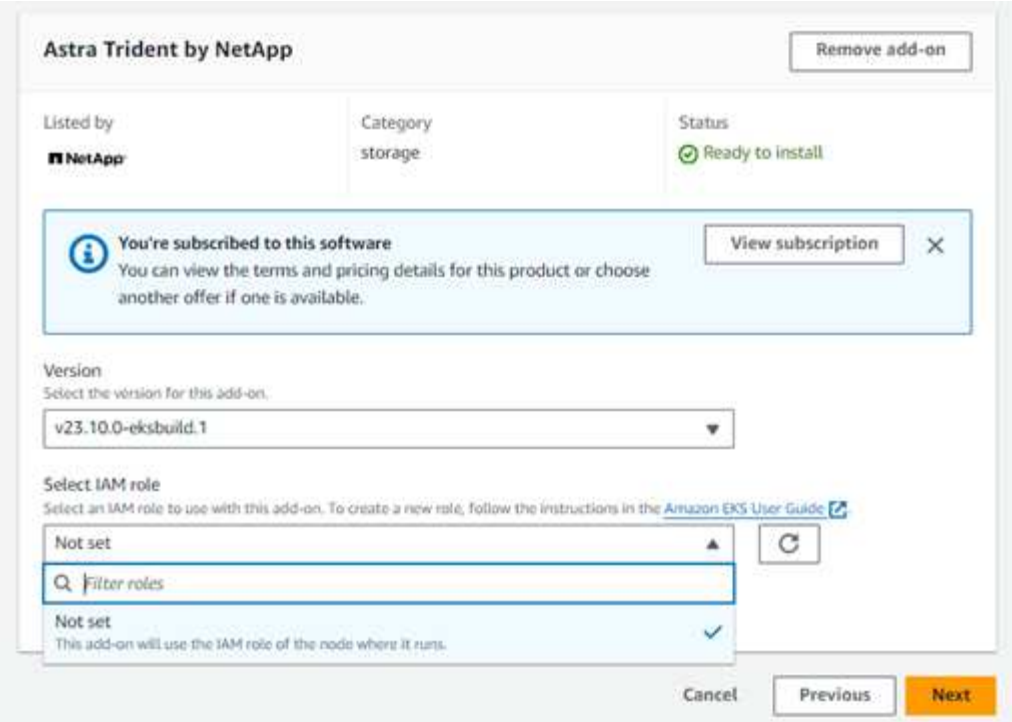
1. EKS Kubernetesクラスタで、*アドオン*タブに移動します。



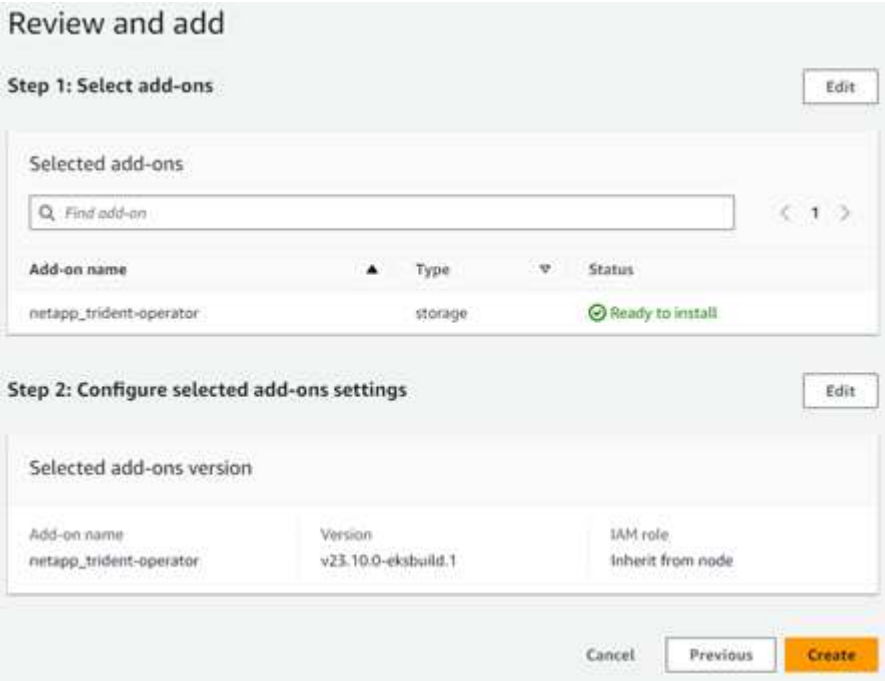
2. [AWS Marketplace add-ons]*にアクセスし、_storage_categoryを選択します。



3. [AstraTrident by NetApp *]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. 必要に応じてオプションの設定を行い、* Next *を選択します。



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。

```
eksctl create addon --cluster K8s-arm --name netapp_trident-operator --version
v23.10.0-eksbuild.
eksctl create addon --cluster K8s-arm --name netapp_trident-operator --version
v23.10.0-eksbuild.1 （専用バージョンを使用）
```

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 カスタムリソース定義（ CRD ）を使用すると、 Trident バックエンドを Kubernetes インターフェイスから直接作成および管理できます。これを行うには 'kubectl' または Kubernetes ディストリビューションに相当する CLI ツールを使用します

`TridentBackendConfig`

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

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

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

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

TridentBackend CRS は Astra Trident によって自動的に作成されます。これらは * 変更しないでください。バックエンドを更新する場合は、「 TridentBackendConfig 」オブジェクトを変更します。

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

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

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

。 `spec` バックエンド固有の設定パラメータを使用します。この例では、バックエンドはを使用します `ontap-san` storage driverおよびでは、に示す構成パラメータを使用します。ご使用のストレージドライバの設定オプションのリストについては、 "ストレージドライバのバックエンド設定情報"。

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

- `credentials` ：このパラメータは必須フィールドで、ストレージシステム / サービスとの認証に使用されるクレデンシャルが含まれています。ユーザが作成した Kubernetes Secret に設定されます。クレデンシャルをプレーンテキストで渡すことはできないため、エラーになります。
- `DeleetionPolicy`: 「 TridentBackendConfig 」が削除されたときに何が起こるかを定義します。次の 2 つの値のいずれかを指定できます。
  - 「削除」：これにより、「 TridentBackendConfig 」 CR とそれに関連付けられたバックエンドの両方が削除されます。これがデフォルト値です。
  - 「管理」：「 TridentBackendConfig 」 CR を削除しても、バックエンド定義は引き続き表示され、「 tridentctl 」で管理できます。削除ポリシーを「 retain 」に設定すると、ユーザは以前のリリース (21.04 より前) にダウングレードし、作成されたバックエンドを保持できます。このフィールドの値は、「 TridentBackendConfig 」が作成された後で更新できます。

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

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

手順の概要

kubectl' を使用して新しいバックエンドを作成するには ' 次の手順を実行する必要があります

1. を作成します "Kubernetes Secret"。シークレットには、ストレージクラスタ / サービスと通信するために Trident から必要なクレデンシャルが含まれています。
2. 「 TridentBackendConfig 」オブジェクトを作成します。ストレージクラスタ / サービスの詳細を指定し、前の手順で作成したシークレットを参照します。

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

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

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

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

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

ストレージプラットフォームのシークレットフィールド概要	秘密	**Field** 概要の略
Azure NetApp Files の特長	ClientID	アプリケーション登録からのクライアント ID
Cloud Volumes Service for GCP	private_key_id です	秘密鍵の ID 。CVS 管理者ロールを持つ GCP サービスアカウントの API キーの一部

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
Cloud Volumes Service for GCP	private_key を使用します	秘密鍵CVS 管理者ロールを持つ GCP サービスアカウントの API キーの一部
Element （ NetApp HCI / SolidFire ）	エンドポイント	テナントのクレデンシャルを使用する SolidFire クラスタの MVIP
ONTAP	ユーザ名	クラスタ / SVM に接続するためのユーザ名。クレデンシャルベースの認証に使用されます
ONTAP	パスワード	クラスタ / SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます
ONTAP	clientPrivateKey	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます
ONTAP	chapUsername のコマンド	インバウンドユーザ名。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合
ONTAP	chapInitiatorSecret	CHAP イニシエータシークレット。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合
ONTAP	chapTargetUsername のコマンド	ターゲットユーザ名。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合
ONTAP	chapTargetInitiatorSecret	CHAP ターゲットイニシエータシークレット。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合

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

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

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

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

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

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

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

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

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

フェーズには次のいずれかの値を指定できます。

- `Bound`: `TridentBackendConfig` CRはバックエンドに関連付けられており、そのバックエンドにはが含まれています `configRef` をに設定します `TridentBackendConfig` crのuid
- Unbound ： "" を使用して表現されています「 TridentBackendConfig 」オブジェクトはバックエンドにバインドされません。新しく作成されたすべての TridentBackendConfig ' CRS は、デフォルトでこのフェーズに入ります。フェーズが変更された後、再度 Unbound に戻すことはできません。
- `Deleting`: `TridentBackendConfig` CR `deletionPolicy` が削除対象に設定されました。をクリックします `TridentBackendConfig` CRが削除され、削除状態に移行します。
  - バックエンドに永続ボリューム要求（ PVC ）が存在しない場合、「 TridentBackendConfig 」を削除すると、 Astra Trident はバックエンドと「 TridentBackendConfig 」 CR を削除します。
  - バックエンドに 1 つ以上の PVC が存在する場合は、削除状態になります。次に 'TridentBackendConfig'CR が削除フェーズに入りますバックエンドおよび TridentBackendConfig は、すべての PVC が削除された後にのみ削除されます。

- `lost` ：「 TridentBackendConfig 」 CR に関連付けられているバックエンドが誤って削除されたか、意図的に削除されました。「 TridentBackendConfig 」 CR には削除されたバックエンドへの参照があります。「 TridentBackendConfig 」 CR は、「 $eleetionPolicy 」の値に関係なく削除できます。
- `Unknown`：Astra Tridentは、に関連付けられているバックエンドの状態または存在を特定できません `TridentBackendConfig` CR。たとえば、APIサーバが応答していない場合や、が応答していない場合などです `tridentbackends.trident.netapp.io` CRDがありません。これには介入が必要な場合があります

この段階では、バックエンドが正常に作成されます。など、いくつかの操作を追加で処理することができます "バックエンドの更新とバックエンドの削除"。

（オプション）手順 **4** ：詳細を確認します

バックエンドに関する詳細情報を確認するには、次のコマンドを実行します。

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

```
NAME                      BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8   Bound   Success   ontap-san        delete
```

さらに、「 TridentBackendConfig 」の YAML ／ JSON ダンプを取得することもできます。

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

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

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

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

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

バックエンドの管理

kubectl を使用してバックエンド管理を実行します

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

バックエンドを削除します

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

次のコマンドを実行します。

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

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

既存のバックエンドを表示します

次のコマンドを実行します。

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

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

バックエンドを更新します

バックエンドを更新する理由はいくつかあります。

- ストレージシステムのクレデンシャルが変更されている。クレデンシャルを更新するには、「 TridentBackendConfig 」オブジェクトで使用される Kubernetes Secret を更新する必要があります。Astra Trident が、提供された最新のクレデンシャルでバックエンドを自動的に更新次のコマンドを実行して、Kubernetes Secret を更新します。

  ```
  kubectl apply -f <updated-secret-file.yaml> -n trident
  ```

- パラメータ（使用する ONTAP SVM の名前など）を更新する必要があります。
  - 更新できます `TridentBackendConfig` 次のコマンドを使用して、Kubernetesから直接オブジェクトを作成します。

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

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

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

> - バックエンドの更新に失敗した場合、バックエンドは最後の既知の設定のまま残ります。ログを表示して原因を確認するには、「 kubectl get tbc <tbc-name> -o yaml -n trident` または「 kubectl describe tbc <tbc-name> -n trident` 」を実行します。
> - 構成ファイルで問題を特定して修正したら、 update コマンドを再実行できます。

tridentctl を使用してバックエンド管理を実行します

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

バックエンドを作成します

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

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

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

```
tridentctl logs -n trident
```

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

バックエンドを削除します

Astra Trident からバックエンドを削除するには、次の手順を実行します。

1. バックエンド名を取得します。

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

2. バックエンドを削除します。

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

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

既存のバックエンドを表示します

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

- 概要を取得するには、次のコマンドを実行します。

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

- すべての詳細を確認するには、次のコマンドを実行します。

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

バックエンドを更新します

新しいバックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs -n trident
```

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

バックエンドを使用するストレージクラスを特定します

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

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

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

バックエンド管理オプション間を移動します

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

バックエンドを管理するためのオプション

を導入しました `TridentBackendConfig`管理者は現在、バックエンドを2つの方法で管理できるようになっています。これには、次のような質問があります。

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

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

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

これは、次のシナリオに該当します。

- 既存のバックエンドには `TridentBackendConfig` を使用して作成されたためです `tridentctl`。
- 「 tridentctl 」で作成された新しいバックエンドと、その他の「 TridentBackendConfig 」オブジェクトが存在します。

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

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

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

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

手順 **0** ：バックエンドを特定します

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

```
tridentctl get backend ontap-nas-backend -n trident
+---------------------+----------------
+--------------------------------------+--------+---------+
```

```
|          NAME       | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+---------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+----------------
+--------------------------------------+--------+---------+

cat ontap-nas-backend.json

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

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

```
    ]
}
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

手順 **0** ：バックエンドを特定します

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ストレージクラスの作成と管理

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

。 "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を特定し、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: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

を参照してください "Kubernetes オブジェクトと Trident オブジェクト" ストレージクラスとの連携の詳細については、を参照してください。 `PersistentVolumeClaim` とパラメータを使用して、Astra Tridentでボリュームをプロビジョニングする方法を制御します。

ストレージクラスを作成する。

StorageClassオブジェクトを作成したら、ストレージクラスを作成できます。 [ストレージクラスノサンプル] に、使用または変更できる基本的なサンプルを示します。

手順

1. これはKubernetesオブジェクトなので、 `kubectl` をクリックしてKubernetesで作成します。

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

2. Kubernetes と 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、 `sample-input/storage-class-csi.yaml.templ` インストーラに付属しており、 *`BACKEND_TYPE`* ストレージドライバの名前を指定します。

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

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

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

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の「 torageclass.Kubernetes .io/is-default-class 」を true に設定して、デフォルトのストレージクラスを定義します。仕様に応じて、それ以外の値やアノテーションがない場合は false と解釈されます。
- 次のコマンドを使用して、既存のストレージクラスをデフォルトのストレージクラスとして設定できます。

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

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

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

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

> クラスタ内のデフォルトのストレージクラスは一度に1つだけにしてください。Kubernetes では、技術的に複数のストレージを使用することはできますが、デフォルトのストレージクラスがまったくない場合と同様に動作します。

ストレージクラスのバックエンドを特定します

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

```
tridentctl get storageclass -o json | jq  '[.items[] | {storageClass:
.Config.name, backends: [.storage]|unique}]'
```

ストレージクラスを削除する

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

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

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

このストレージクラスで作成された永続ボリュームには変更はなく、 Astra Trident によって引き続き管理されます。

> Astra Tridentでは空白が強制される `fsType` を作成します。iSCSIバックエンドの場合は、適用することを推奨します `parameters.fsType` ストレージクラス。既存のストレージクラスを削除して、で再作成する必要があります `parameters.fsType` 指定された。

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

A "永続ボリューム_" （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。

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**

次の例は、という名前のStorageClassに関連付けられた、RWOアクセスが設定された基本的なPVCを示しています。 `basic-csi`。

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

NVMe / TCP対応**PVC**

この例は、という名前のStorageClassに関連付けられたNVMe/TCPの基本的なPVCとRWOアクセスを示しています。 `protection-gold`。

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

PODマニフェストのサンプル

次の例は、PVCをポッドに接続するための基本的な設定を示しています。

キホンセツテイ

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: 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を作成します。

```
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 を作成します。

```
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
```

> 進捗状況は次を使用して監視できます。 `kubectl get pod --watch`。

6. ボリュームがマウントされていることを確認します。 `/my/mount/path`。

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

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
```

を参照してください "Kubernetes オブジェクトと Trident オブジェクト" ストレージクラスとの連携の詳細については、を参照してください。 `PersistentVolumeClaim` とパラメータを使用して、Astra Tridentでボリュームをプロビジョニングする方法を制御します。

ボリュームを展開します

Astra Trident により、 Kubernetes ユーザは作成後にボリュームを拡張できます。ここでは、 iSCSI ボリュームと NFS ボリュームの拡張に必要な設定について説明します。

iSCSI ボリュームを展開します

CSI プロビジョニングを使用して、 iSCSI Persistent Volume （ PV ）を拡張できます。

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

手順 **1** ：ボリュームの拡張をサポートするようにストレージクラスを設定する

StorageClass定義を編集して `allowVolumeExpansion` フィールドからに移動します `true`。

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

既存の StorageClass の場合は 'allowVolumeExpansion パラメータを含めるように編集します

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

PVC定義を編集し、 `spec.resources.requests.storage` 新たに必要となったサイズを反映するには、元のサイズよりも大きくする必要があります。

```
cat pvc-ontapsan.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

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、ポッドが作成され、「 1-pvc 」が使用されます。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

ステップ **4** ： **PV** を展開します

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

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

手順 **5** ：拡張を検証する

PVC 、 PV 、 Astra Trident のボリュームのサイズを確認することで、拡張が正しく機能しているかどうかを検証できます。

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
tridentctl get volumes -n trident
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
```

NFS ボリュームを拡張します

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

手順 **1** ：ボリュームの拡張をサポートするようにストレージクラスを設定する

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

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

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

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

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

Astra Trident が、この PVC に対して 20MiB の NFS PV を作成する必要があります。

```
kubectl get pvc
NAME           STATUS   VOLUME
CAPACITY     ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi
RWO            ontapnas        9s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi       RWO
Delete           Bound    default/ontapnas20mb   ontapnas
2m42s
```

ステップ **3** ： **PV** を展開します

新しく作成した20MiBのPVのサイズを1GiBに変更するには、そのPVCを編集してを設定します `spec.resources.requests.storage` 1GiBへ：

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

手順 **4** ：拡張を検証する

PVC 、 PV 、 Astra Trident のボリュームのサイズを確認することで、サイズ変更が正しく機能しているかどうかを検証できます。

```
kubectl get pvc ontapnas20mb
NAME           STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi
RWO            ontapnas        4m44s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi        RWO
Delete           Bound    default/ontapnas20mb   ontapnas
5m35s

tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n trident
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
```

ボリュームをインポート

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

概要と考慮事項

Astra Tridentにボリュームをインポートすると、次のことが可能になります。

- アプリケーションをコンテナ化し、既存のデータセットを再利用する
- 一時的なアプリケーションにはデータセットのクローンを使用
- 障害が発生したKubernetesクラスタを再構築します
- ディザスタリカバリ時にアプリケーションデータを移行

考慮事項

ボリュームをインポートする前に、次の考慮事項を確認してください。

- Astra TridentでインポートできるのはRW（読み取り/書き込み）タイプのONTAPボリュームのみです。DP（データ保護）タイプのボリュームはSnapMirrorデスティネーションボリュームです。ボリュームをAstra

Tridentにインポートする前に、ミラー関係を解除する必要があります。

- アクティブな接続がないボリュームをインポートすることを推奨します。アクティブに使用されているボリュームをインポートするには、ボリュームのクローンを作成してからインポートを実行します。

> Kubernetesは以前の接続を認識せず、アクティブなボリュームをポッドに簡単に接続できるため、これはブロックボリュームで特に重要です。その結果、データが破損する可能性があります。

- でもね `StorageClass` PVCに対して指定する必要があります。Astra Tridentはインポート時にこのパラメータを使用しません。ストレージクラスは、ボリュームの作成時に、ストレージ特性に基づいて使用可能なプールから選択するために使用されます。ボリュームはすでに存在するため、インポート時にプールを選択する必要はありません。そのため、PVCで指定されたストレージクラスと一致しないバックエンドまたはプールにボリュームが存在してもインポートは失敗しません。
- 既存のボリュームサイズはPVCで決定され、設定されます。ストレージドライバによってボリュームがインポートされると、 PV は ClaimRef を使用して PVC に作成されます。
  - 再利用ポリシーは、最初にににに設定されています `retain` PVにあります。Kubernetes が PVC と PV を正常にバインドすると、再利用ポリシーがストレージクラスの再利用ポリシーに合わせて更新されます。
  - ストレージクラスの再利用ポリシーがの場合 `delete`にすると、PVが削除されるとストレージボリュームが削除されます。
- デフォルトでは、Astra TridentがPVCを管理し、バックエンドでFlexVolとLUNの名前を変更します。を渡すことができます `--no-manage` 管理対象外のボリュームをインポートするフラグ。を使用する場合 `--no-manage`Astra Tridentは、オブジェクトのライフサイクルを通じてPVCやPVに対して追加の処理を実行することはありません。PVが削除されてもストレージボリュームは削除されず、ボリュームのクローンやボリュームのサイズ変更などのその他の処理も無視されます。

> このオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、Kubernetes 以外でストレージボリュームのライフサイクルを管理する場合に便利です。

- PVC と PV にアノテーションが追加されます。このアノテーションは、ボリュームがインポートされたこと、および PVC と PV が管理されていることを示す二重の目的を果たします。このアノテーションは変更または削除しないでください。

ボリュームをインポートします

を使用できます `tridentctl import` をクリックしてボリュームをインポートします。

手順

1. Persistent Volume Claim（PVC；永続的ボリューム要求）ファイルを作成します（例： `pvc.yaml`）をクリックします。PVCファイルには、が含まれている必要があります `name`、 `namespace`、 `accessModes`および `storageClassName`。必要に応じて、を指定できます `unixPermissions` 定義されています。

   最小仕様の例を次に示します。

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

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. を使用します `tridentctl import` コマンドを使用して、ボリュームを含むAstra Tridentバックエンドの名前と、ストレージ上のボリュームを一意に識別する名前（ONTAP FlexVol、Elementボリューム、Cloud Volumes Serviceパスなど）を指定します。。 `-f` PVCファイルへのパスを指定するには、引数が必要です。

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

例

サポートされているドライバについて、次のボリュームインポートの例を確認してください。

ONTAP NASおよびONTAP NAS FlexGroup

Astra Tridentでは、を使用したボリュームインポートがサポートされます `ontap-nas` および `ontap-nas-flexgroup` ドライバ。

- 。 `ontap-nas-economy` ドライバでqtreeをインポートおよび管理できない。
- 。 `ontap-nas` および `ontap-nas-flexgroup` ドライバでボリューム名の重複が許可されていません。

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

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、という名前のボリュームをインポートします `managed_volume` という名前のバックエンドで `ontap_nas`：

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

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

管理対象外のボリューム

を使用する場合 `--no-manage` 引数に指定します。Astra Tridentはボリュームの名前を変更しません。

次に、をインポートする例を示します `unmanaged_volume` をクリックします `ontap_nas` バックエンド：

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

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

ONTAP SAN

Astra Tridentでは、を使用したボリュームインポートがサポートされます `ontap-san` ドライバ。を使用したボリュームインポートはサポートされていません `ontap-san-economy` ドライバ。

Astra Tridentでは、単一のLUNを含むONTAP SAN FlexVolをインポートできます。これはと同じです `ontap-`

`san` ドライバ。FlexVol 内の各PVCおよびLUNにFlexVol を作成します。Astra TridentがFlexVolをインポートし、PVCの定義に関連付けます。

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の名前をに変更します `pvc-<uuid>` およびFlexVol内のLUNをからにフォーマットします `lun0`。

次の例は、をインポートします `ontap-san-managed` にあるFlexVol `ontap_san_default` バックエンド：

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

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

管理対象外のボリューム

次に、をインポートする例を示します `unmanaged_example_volume` をクリックします `ontap_san` バックエンド：

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

次の例に示すように、KubernetesノードのIQNとIQNを共有するigroupにLUNをマッピングすると、エラーが表示されます。 `LUN already mapped to initiator(s) in this group`。ボリュームをインポートするには、イニシエータを削除するか、LUNのマッピングを解除する必要があります。

```
Vserver   Igroup       Protocol OS Type  Initiators
--------- ------------ -------- -------- ------------------------------------
svm0      k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3
                       iscsi    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
svm0      unmanaged-example-igroup
                       mixed    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
```

要素（ **Element** ）

Astra Tridentでは、を使用したNetApp ElementソフトウェアとNetApp HCIボリュームのインポートがサポートされます `solidfire-san` ドライバ。

> Element ドライバではボリューム名の重複がサポートされます。ただし、ボリューム名が重複している場合はAstra Tridentからエラーが返されます。回避策としてボリュームをクローニングし、一意のボリューム名を指定して、クローンボリュームをインポートします。

要素の例

次に、をインポートする例を示します `element-managed` バックエンドのボリューム `element_default`。

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

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

Google Cloud Platform の **1** つです

Astra Tridentでは、を使用したボリュームインポートがサポートされます `gcp-cvs` ドライバ。

NetApp Cloud Volumes Serviceから作成されたボリュームをGoogle Cloud Platformにインポートするには、ボリュームパスでボリュームを特定します。ボリュームパスは、ボリュームのエクスポートパスののに続く部分です `:/`。たとえば、エクスポートパスがの場合などです `10.0.0.1:/adroit-jolly-swift`、ボリュームのパスはです `adroit-jolly-swift`。

Google Cloud Platformの例

次に、をインポートする例を示します `gcp-cvs` バックエンドのボリューム `gcpcvs_YEppr` を指定します `adroit-jolly-swift`。

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

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

Azure NetApp Files の特長

Astra Tridentでは、を使用したボリュームインポートがサポートされます `azure-netapp-files` ドライバ。

Azure NetApp Filesボリュームをインポートするには、ボリュームパスでボリュームを特定します。ボリュームパスは、ボリュームのエクスポートパスののに続く部分です `:/`。たとえば、マウントパスがの場合などです `10.0.0.2:/importvol1`、ボリュームのパスはです `importvol1`。

Azure NetApp Filesの例

次に、をインポートする例を示します `azure-netapp-files` バックエンドのボリューム `azurenetappfiles_40517` を指定します `importvol1`。

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

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

ネームスペース間で**NFS**ボリュームを共有します

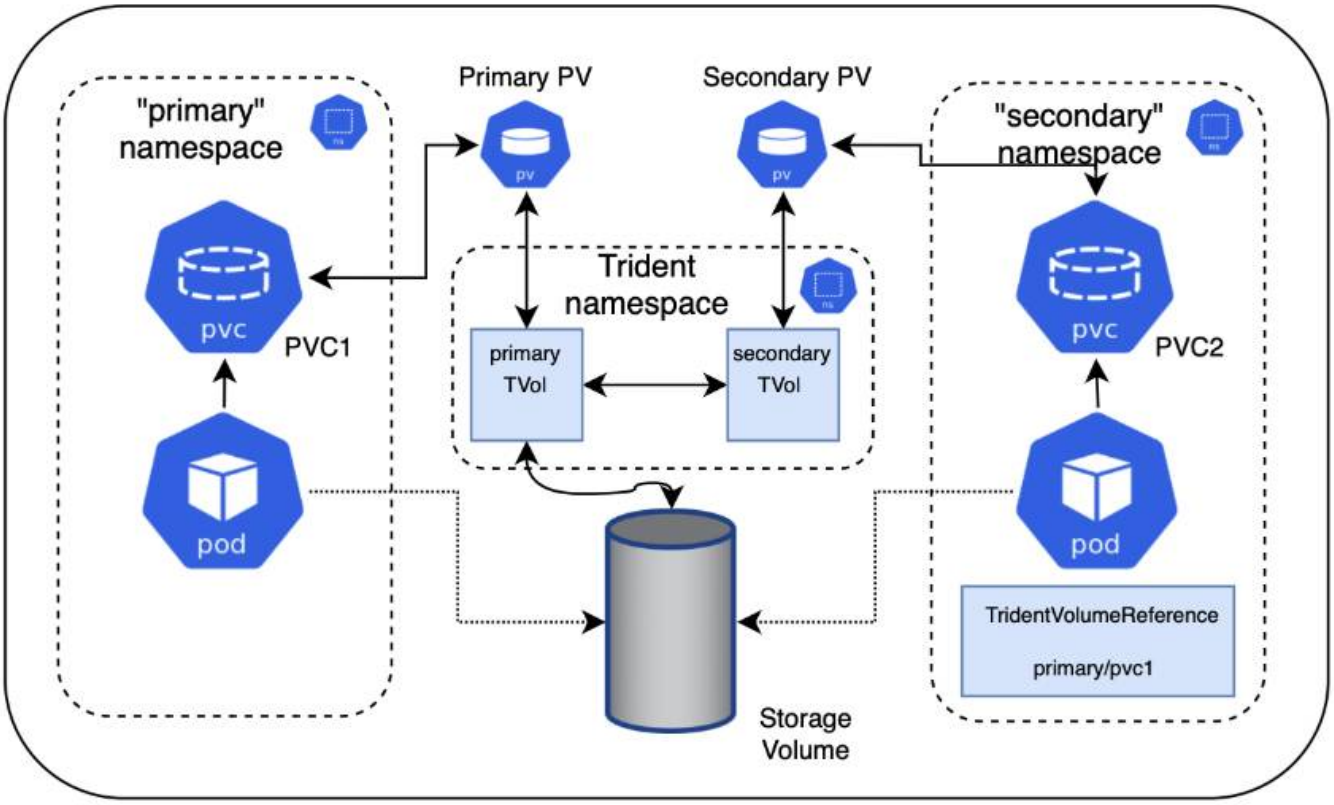
Tridentを使用すると、プライマリネームスペースにボリュームを作成し、1つ以上のセカンダリネームスペースで共有できます。

の機能

Astra TridentVolumeReference CRを使用すると、1つ以上のKubernetesネームスペース間でReadWriteMany（RWX）NFSボリュームをセキュアに共有できます。このKubernetesネイティブ解決策 には、次のようなメリットがあります。

- セキュリティを確保するために、複数のレベルのアクセス制御が可能です
- すべてのTrident NFSボリュームドライバで動作
- tridentctlやその他の非ネイティブのKubernetes機能に依存しません

この図は、2つのKubernetesネームスペース間でのNFSボリュームの共有を示しています。



クイックスタート

NFSボリューム共有はいくつかの手順で設定できます。

1 ボリュームを共有するようにソース**PVC**を設定します

ソースネームスペースの所有者は、ソースPVCのデータにアクセスする権限を付与します。

2 デスティネーションネームスペースに**CR**を作成する権限を付与します

クラスタ管理者が、デスティネーションネームスペースの所有者にTridentVolumeReference CRを作成する権限を付与します。

3 デスティネーションネームスペースに**TridentVolumeReference**を作成します

宛先名前空間の所有者は、送信元PVCを参照するためにTridentVolumeReference CRを作成します。

4 宛先名前空間に下位**PVC**を作成します

宛先名前空間の所有者は、送信元PVCからのデータソースを使用する下位PVCを作成します。

ソースネームスペースとデスティネーションネームスペースを設定します

セキュリティを確保するために、ネームスペース間共有では、ソースネームスペースの所有者、クラスタ管理

者、および宛先ネームスペースの所有者によるコラボレーションとアクションが必要です。ユーザロールは各手順で指定します。

手順

1. ソース名前空間の所有者： PVCを作成します (`pvc1`）をソースネームスペースに追加し、デスティネーションネームスペースとの共有権限を付与します (`namespace2`)を使用します `shareToNamespace` アノテーション

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

   Astra TridentがPVとバックエンドのNFSストレージボリュームを作成

   - カンマ区切りリストを使用して、複数の名前空間にPVCを共有できます。例： `trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`。
   - を使用して、すべてのネームスペースに共有できます `*`。例： `trident.netapp.io/shareToNamespace: *`
   - PVCを更新してを含めることができます `shareToNamespace` アノテーションはいつでも作成できます。

2. *クラスタ管理者：*カスタムロールとkubeconfigを作成して、デスティネーションネームスペースの所有者にTridentVolumeReference CRを作成する権限を付与します。
3. *デスティネーションネームスペース所有者：*ソースネームスペースを参照するデスティネーションネームスペースにTridentVolumeReference CRを作成します `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 宛先名前空間の所有者： PVCを作成します (`pvc2`）をデスティネーションネームスペースに展開します (`namespace2`)を使用します `shareFromPVC` 送信元PVCを指定する注釈。

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

宛先PVCのサイズは、送信元PVCのサイズ以下である必要があります。

結果

Astra Tridentがを読み取り `shareFromPVC` デスティネーションPVCにアノテーションを設定し、ソースPVを参照するストレージリソースを持たない下位のボリュームとしてデスティネーションPVを作成し、ソースPVストレージリソースを共有します。宛先PVCとPVは、通常どおりバインドされているように見えます。

共有ボリュームを削除

複数のネームスペースで共有されているボリュームは削除できます。Tridentが、ソースネームスペースのボリュームへのアクセスを削除し、ボリュームを共有する他のネームスペースへのアクセスを維持します。ボリュームを参照するすべてのネームスペースが削除されると、Astra Tridentによってボリュームが削除されます。

使用 `tridentctl get` 下位のボリュームを照会する

を使用する[`tridentctl` ユーティリティを使用すると、を実行できます `get` コマンドを使用して下位のボリュームを取得します。詳細については、リンク:./trident-reference/tridentctl.htmlを参照してください[`tridentctl` コマンドとオプション]。

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

フラグ：

- `-h, --help：ボリュームのヘルプ。
- `--parentOfSubordinate string`：クエリを下位のソースボリュームに制限します。
- `--subordinateOf string`:クエリをボリュームの下位に制限します。

制限

- Astra Tridentでは、デスティネーションネームスペースが共有ボリュームに書き込まれるのを防ぐことはできません。共有ボリュームのデータの上書きを防止するには、ファイルロックなどのプロセスを使用する必要があります。
- を削除しても、送信元PVCへのアクセスを取り消すことはできません `shareToNamespace` または `shareFromNamespace` 注釈またはを削除します `TridentVolumeReference` CR。アクセスを取り消すには、下位PVCを削除する必要があります。
- Snapshot、クローン、およびミラーリングは下位のボリュームでは実行できません。

を参照してください。

ネームスペース間のボリュームアクセスの詳細については、次の資料を参照してください。

- にアクセスします "ネームスペース間でのボリュームの共有：ネームスペース間のボリュームアクセスを許可する場合は「Hello」と入力します"。
- のデモをご覧ください "ネットアップTV"。

CSI トポロジを使用します

Astra Trident では、を使用して、 Kubernetes クラスタ内にあるノードにボリュームを選択的に作成して接続できます "CSI トポロジ機能"。

概要

CSI トポロジ機能を使用すると、領域およびアベイラビリティゾーンに基づいて、ボリュームへのアクセスをノードのサブセットに制限できます。現在、クラウドプロバイダは、 Kubernetes 管理者がゾーンベースのノードを生成できるようになっています。ノードは、リージョンによって異なるアベイラビリティゾーンに配置することも、リージョンによって配置することもできます。マルチゾーンアーキテクチャでワークロード用のボリュームをプロビジョニングするために、 Astra Trident は CSI トポロジを使用します。

> CSI トポロジ機能の詳細については、を参照してください "こちらをご覧ください"。

Kubernetes には、 2 つの固有のボリュームバインドモードがあります。

- を使用 `VolumeBindingMode` をに設定します `Immediate`トポロジを認識することなくボリュームを作成できます。ボリュームバインディングと動的プロビジョニングは、 `PVC` が作成されるときに処理され

ます。これがデフォルトです `VolumeBindingMode` また、トポロジの制約を適用しないクラスタにも適しています。永続ボリュームは、要求元ポッドのスケジュール要件に依存することなく作成されます。

- VolumeBindingMode を「 WaitForFirstConsumer 」に設定すると、 PVC の永続ボリュームの作成とバインドは、 PVC を使用するポッドがスケジュールされて作成されるまで遅延されます。これにより、トポロジの要件に応じたスケジュールの制約を満たすようにボリュームが作成されます。

> 「 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 ' トポロジー認識を導入するラベルが必要です（ topology .Kubernetes .io/region および topology .Kubernetes .io/zone ）このラベル * は、 Astra Trident をトポロジ対応としてインストールする前に、クラスタ内のノードに存在する必要があります。

```
kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},
{.metadata.labels}]{"\n"}{end}' | grep --color "topology.kubernetes.io"
[node1,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node1","kubernetes.io/
os":"linux","node-
role.kubernetes.io/master":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-a"}]
[node2,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node2","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-b"}]
[node3,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-c"}]
```

手順 **1** ：トポロジ対応バックエンドを作成する

Astra Trident ストレージバックエンドは、アベイラビリティゾーンに基づいてボリュームを選択的にプロビジョニングするように設計できます。各バックエンドは、サポートする必要があるゾーンおよびリージョンのリストを表すオプションの「 upportedTopologies 」ブロックを伝送できます。ストレージクラスがそのようなバックエンドを使用する場合、ボリュームは、サポートされているリージョン / ゾーンでスケジュールされているアプリケーションから要求された場合にのみ作成されます。

バックエンド定義の例を次に示します。

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"}
]
}
```

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

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

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-backend-us-central1
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
- topology.kubernetes.io/region: us-central1
  topology.kubernetes.io/zone: us-central1-a
- topology.kubernetes.io/region: us-central1
  topology.kubernetes.io/zone: us-central1-b
storage:
- labels:
    workload: production
  region: Iowa-DC
  zone: Iowa-DC-A
  supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-a
- labels:
    workload: dev
  region: Iowa-DC
  zone: Iowa-DC-B
  supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-b
```

この例では、「 re Gion 」および「 zone 」ラベルはストレージプールの場所を表しています。「 topology.efix/region' 」と「 topology.uns.io/zone 」は、ストレージプールの消費元を決定します。

手順 **2** ：トポロジを認識するストレージクラスを定義する

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

次の例を参照してください。

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

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

ステップ **3** ： **PVC** を作成して使用する

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

以下の例「 PEC 」を参照してください。

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

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

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

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

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

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

次の出力を参照してください。

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

バックエンドを更新して追加 `supportedTopologies`

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

詳細については、こちらをご覧ください

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

スナップショットを操作します

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、でサポートされます `ontap-nas`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`、 `gcp-cvs`および `azure-netapp-files` ドライバ。

作業を開始する前に

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. を作成します `VolumeSnapshotClass`。詳細については、を参照してください "ボリュームSnapshotクラス"。
   - 。 `driver` Astra Trident CSIドライバを指します。
   - `deletionPolicy` は、です `Delete` または `Retain`。に設定すると `Retain`を使用すると、ストレージクラスタの基盤となる物理`Snapshot`が、の場合でも保持されます `VolumeSnapshot` オブジェクトが削除された。

     例

     ```
     cat snap-sc.yaml
     apiVersion: snapshot.storage.k8s.io/v1
     kind: VolumeSnapshotClass
     metadata:
       name: csi-snapclass
     driver: csi.trident.netapp.io
     deletionPolicy: Delete
     ```

2. 既存のPVCのスナップショットを作成します。

   例

   - 次に、既存のPVCのスナップショットを作成する例を示します。

     ```
     cat snap.yaml
     apiVersion: snapshot.storage.k8s.io/v1
     kind: VolumeSnapshot
     metadata:
       name: pvc1-snap
     spec:
       volumeSnapshotClassName: csi-snapclass
       source:
         persistentVolumeClaimName: pvc1
     ```

   - 次の例は、という名前のPVCのボリュームSnapshotオブジェクトを作成します。 `pvc1` Snapshotの名前はに設定されます `pvc1-snap`。ボリュームSnapshotはPVCに似ており、に関連付けられています `VolumeSnapshotContent` 実際のスナップショットを表すオブジェクト。

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

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

- 次の情報を確認できます。 `VolumeSnapshotContent` のオブジェクト `pvc1-snap` ボリュームSnapshot。ボリュームSnapshotの詳細を定義します。。 `Snapshot Content Name` このSnapshotを提供するVolumeSnapshotContentオブジェクトを特定します。。 `Ready To Use` パラメータは、スナップショットを使用して新しいPVCを作成できることを示します。

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

ボリューム**Snapshot**から**PVC**を作成

を使用できます `dataSource` という名前のVolumeSnapshotを使用してPVCを作成するには `<pvc-name>` データのソースとして。作成された PVC は、ポッドに接続して、他の PVC と同様に使用できます。

> PVCはソースボリュームと同じバックエンドに作成されます。を参照してください "KB：Trident PVCスナップショットからPVCを作成することは代替バックエンドではできない"。

次に、を使用してPVCを作成する例を示します。 `pvc1-snap` をデータソースとして使用します。

```
cat pvc-from-snap.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

ボリューム**Snapshot**のインポート

Astra Tridentは以下をサポートします。 "Kubernetesの事前プロビジョニングされたSnapshotプロセス" クラスタ管理者が `VolumeSnapshotContent` Astra Tridentの外部で作成されたオブジェクトとSnapshotをインポート

作業を開始する前に

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： `VolumeSnapshotContent` バックエンドスナップショットを参照するオブジェクト。これにより、Astra TridentでSnapshotワークフローが開始されます。

   - バックエンドスナップショットの名前を `annotations` として `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`。
   - を指定します `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>` インチ `snapshotHandle`。Astra Tridentに提供される唯一の情報は、 `ListSnapshots` 電話だ

     > 。 `<volumeSnapshotContentName>` CRの命名規則のため、バックエンドスナップショット名が常に一致するとは限りません。

     例

     次の例では、 `VolumeSnapshotContent` バックエンドスナップショットを参照するオブジェクト `snap-01`。

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

2. クラスタ管理者： `VolumeSnapshot` を参照するCR `VolumeSnapshotContent` オブジェクト。これにより、 `VolumeSnapshot` 指定された名前空間内。

   例

   次の例では、 `VolumeSnapshot` CR名 `import-snap` を参照しています。 `VolumeSnapshotContent` 名前付き `import-snap-content`。

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

3. *内部処理（アクション不要）：*外部スナップショットは、新しく作成されたスナップショットを認識します。 `VolumeSnapshotContent` を実行します。 `ListSnapshots` 電話だAstra Tridentが `TridentSnapshot`。
   - 外部スナップショットは、 `VolumeSnapshotContent` 終了： `readyToUse` および `VolumeSnapshot` 終了： `true`。
   - Tridentのリターン `readyToUse=true`。
4. 任意のユーザー： `PersistentVolumeClaim` 新しい `VolumeSnapshot`を参照してください `spec.dataSource` （または `spec.dataSourceRef`）nameは `VolumeSnapshot` 名前。

   例

   次に、を参照するPVCを作成する例を示します。 `VolumeSnapshot` 名前付き `import-snap`。

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

Snapshotを使用してボリュームデータをリカバリします

snapshotディレクトリは、を使用してプロビジョニングされるボリュームの互換性を最大限に高めるため、デフォルトでは非表示になっています `ontap-nas` および `ontap-nas-economy` ドライバ。を有効にします `.snapshot` スナップショットからデータを直接リカバリするディレクトリ。

ボリュームを以前のSnapshotに記録されている状態にリストアするには、ボリュームSnapshotリストアONTAP CLIを使用します。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

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、を開きます `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` およびを更新します `namespace` に移動します。

関連リンク

- "ボリューム Snapshot"
- "ボリュームSnapshotクラス"

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

Astra Trident は四半期ごとにリリースサイクルを実施し、毎年 4 つのメジャーリリースをリリースしています。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。
- Cloud Volumes Service for Google Cloudを "CVS サービスタイプ"を使用するには、バックエンド構成を更新する必要があります。 `standardsw` または `zoneredundantstandardsw` Astra Trident 23.01からアップグレードする場合のサービスレベル。の更新に失敗しました `serviceLevel` バックエンドでは、原因ボリュームで障害が発生する可能性があります。を参照してください "CVSサービスタイプのサンプル" を参照してください。
- アップグレードするときは、この作業を行うことが重要です `parameter.fsType` インチ `StorageClasses` Astra Tridentが使用。削除して再作成することができます `StorageClasses` 実行前のボリュームの中断はなし。
  - これは、強制の 要件 です "セキュリティコンテキスト" SAN ボリュームの場合。
  - sample inputディレクトリには、https://github.com/NetApp/trident/blob/master/trident-installer/sample-input/storage-class-samples/storage-class-basic.yaml.templなどの例が含まれています[`storage-class-basic.yaml.templ`）とリンク：https://github.com/NetApp/trident/blob/master/trident-installer/sample-input/storage-class-samples/storage-class-bronze-default.yaml[`storage-class-bronze-default.yaml`^]をクリックします。
  - 詳細については、を参照してください "既知の問題"。

ステップ**1**：バージョンを選択します

Astra Tridentバージョンは日付ベースです `YY.MM` 命名規則。「YY」は年の最後の2桁、「MM」は月です。ドットリリースは、の後に続きます `YY.MM.X` 条約。ここで、「X」はパッチレベルです。アップグレード前のバージョンに基づいて、アップグレード後のバージョンを選択します。

- インストールされているバージョンの4リリースウィンドウ内にある任意のターゲットリリースに直接アップグレードできます。たとえば、23.01（または任意の23.01 DOTリリース）から24.02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。のアップグレード手順を使用します。 "以前のバージョン" から、4つのリリースウィンドウに適合する最新のリリースにアップグレードします。たとえば、22.01を実行していて、24.02にアップグレードする場合は、次の手順を実行します。
  a. 22.01から23.01への最初のアップグレード。
  b. その後、23.01から24.02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、 "GitHub の問題の詳細"。

ステップ**2:**元のインストール方法を決定します

Astra Tridentの最初のインストールに使用したバージョンを確認するには、次の手順を実行します。

1. 使用 `kubectl get pods -n trident` ポッドを検査するために。
   - オペレータポッドがない場合は、を使用してAstra Tridentがインストールされています `tridentctl`。
   - オペレータポッドがある場合、Astra Tridentは手動またはHelmを使用してインストールされています。
2. オペレータポッドがある場合は、を使用します `kubectl describe torc` をクリックし、Helmを使用してAstra Tridentがインストールされたかどうかを確認します。
   - Helmラベルがある場合は、Helmを使用してAstra Tridentがインストールされています。
   - Helmラベルがない場合は、Astra TridentをTridentオペレータを使用して手動でインストールしています。

ステップ**3**：アップグレード方法を選択します

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"

> レビューすることをお勧めします "オペレータのアップグレードワークフローについて理解する" オペレータでアップグレードする前に。

*

オペレータにアップグレードしてください

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

多数のうちの1つ "Tridentオペレータを使用するメリット" Astra Tridentのインストールとアップグレードは、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します。これにより、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、 "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オペレータ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環境から別のTridentオペレータのインストール環境へのアップグレード、または `tridentctl` Tridentオペレータバージョンへのインストールレビュー "アップグレード方法を選択します" Tridentオペレータのインストールをアップグレードする前に

手動インストールのアップグレード

クラスタを対象とした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、次のアップグレード手順を使用してください："インストールされているバージョン" 実績があります。

このタスクについて

Tridentにはバンドルファイルが用意されています。このファイルを使用して、オペレータをインストールしたり、Kubernetesバージョンに対応する関連オブジェクトを作成したりできます。

- クラスタでKubernetes 1.24以前を実行している場合は、を使用します "Bundle_pre_1_25.yaml"。
- クラスタでKubernetes 1.25以降を実行している場合は、を使用します "bundle_post_1_25.yaml"。

作業を開始する前に

を実行しているKubernetesクラスタを使用していることを確認します "サポートされるKubernetesバージョン"。

手順

1. Astra Tridentのバージョンを確認します。

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

2. 現在の Astra Trident インスタンスのインストールに使用した Trident オペレータを削除たとえば、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. を使用して初期インストールをカスタマイズした場合 `TridentOrchestrator` 属性を編集できます `TridentOrchestrator` インストールパラメータを変更するオブジェクト。これには、ミラーリングされたTridentおよびCSIイメージレジストリをオフラインモードに指定したり、デバッグログを有効にしたり、イメージプルシークレットを指定したりするための変更が含まれます。
4. 環境に応じた適切なバンドルYAMLファイルを使用してAstra Tridentをインストールします（_YAML <bundle.yaml>_は `bundle_pre_1_25.yaml` または `bundle_post_1_25.yaml` 使用しているKubernetesのバージョンに基づきます。たとえば、Astra Trident 24.02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 24.02.0/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ているkubernetesクラスタを1.24から1.25以降にアップグレードする場合は、value.yamlを更新して設定する必要があります `excludePodSecurityPolicy` 終了： `true` または、を追加します `--set excludePodSecurityPolicy=true` に移動します `helm upgrade` コマンドを実行してからクラスタをアップグレードしてください。

手順

1. あなたの場合 "Helmを使用したAstra Tridentのインストール"を使用できます。 `helm upgrade trident netapp-trident/trident-operator --version 100.2402.0` 1つの手順でアップグレードできます。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. 次のサイトからAstra Tridentの最新リリースをダウンロードしてください： "GitHubの_Assets_sectionを参照してください"。

   b. を使用します `helm upgrade` コマンドを入力します `trident-operator-24.02.0.tgz` アップグレード後のバージョンが反映されます。

   ```
   helm upgrade <name> trident-operator-24.02.0.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イメージとCSIイメージのプライベートなミラーレジストリの指定など）は、 `helm upgrade` コマンド `--set` これらのオプションがupgradeコマンドに含まれるようにするため、それらのオプションの値をdefaultにリセットします。

2. を実行します `helm list` グラフとアプリのバージョンが両方ともアップグレードされていることを確認します。を実行します `tridentctl logs` デバッグメッセージを確認します。

からのアップグレード `tridentctl` Tridentオペレータへのインストール

からTridentの最新リリースにアップグレードできます `tridentctl` インストール：既存のバックエンドとPVCは自動的に使用可能になります。

> インストール方法を切り替える前に、 "インストール方法を切り替える"。

手順

1. 最新の Astra Trident リリースをダウンロード

```
# Download the release required [24.020.0]
mkdir 24.02.0
cd 24.02.0
wget
https://github.com/NetApp/trident/releases/download/v24.02.0/trident-
installer-24.02.0.tar.gz
tar -xf trident-installer-24.02.0.tar.gz
cd trident-installer
```

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

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

3. クラスタを対象としたオペレータを同じネームスペースに導入します。

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

serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created

#Examine the pods in the Trident namespace
NAME                                  READY   STATUS    RESTARTS   AGE
trident-controller-79df798bdc-m79dc   6/6     Running   0          150d
trident-node-linux-xrst8              2/2     Running   0          150d
trident-operator-5574dbbc68-nthjv     1/1     Running   0          1m30s
```

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

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

kubectl create -f deploy/crds/tridentorchestrator_cr.yaml

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

5. Tridentが目的のバージョンにアップグレードされたことを確認

```
kubectl describe torc trident | grep Message -A 3

Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v24.02.0
```

tridentctl を使用してアップグレードします

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

このタスクについて

Astra Trident のアンインストールと再インストールはアップグレードとして機能します。Trident をアンインストールしても、 Astra Trident 環境で使用されている Persistent Volume Claim （ PVC ；永続的ボリューム要求）と Persistent Volume （ PV ；永続的ボリューム）は削除されません。Astra Trident がオフラインの間は、すでにプロビジョニング済みの PVS を引き続き使用でき、 Astra Trident は、オンラインに戻った時点で作成された PVC に対してボリュームをプロビジョニングします。

作業を開始する前に

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

手順

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

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

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

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

Tridentctlを使用したAstra Tridentの管理

。 "Trident インストーラバンドル" には、 `tridentctl` Astra Tridentへのシンプルなアクセスを提供するコマンドラインユーティリティ。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

走れ `tridentctl help` 使用可能なコマンドのリストを取得するには `tridentctl` または、 `--help` 特定のコマンドのオプションとフラグのリストを取得するには、任意のコマンドにフラグを付けます。

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

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

コマンド

`create`
: 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

「バージョン」
: 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
ヘルプ `tridentctl`。

-k、 --kubeconfig string
を指定します。 `KUBECONFIG` コマンドをローカルまたはKubernetesクラスタ間で実行するパス。

> または、 `KUBECONFIG` 特定のKubernetesクラスタと問題をポイントする変数 `tridentctl` そのクラスタにコマンドを送信します。

-n、 --namespace string
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

を使用します `create` Astra Tridentにリソースを追加するコマンド。

```
tridentctl create [option]
```

オプション（ **Options** ）

`backend`：Astra Tridentにバックエンドを追加

削除

を使用します `delete` コマンドを使用して、Astra Tridentから1つ以上のリソースを削除します。

```
tridentctl delete [option]
```

オプション（ **Options** ）

`backend`：Tridentから1つ以上のストレージバックエンドを削除
`snapshot`：Astra Tridentから1つ以上のボリュームSnapshotを削除

`storageclass`：Astra Tridentから1つ以上のストレージクラスを削除
`volume`：Astra Tridentから1つ以上のストレージボリュームを削除

取得

を使用します `get` Astra Tridentから1つ以上のリソースを取得するためのコマンドです。

```
tridentctl get [option]
```

オプション（ **Options** ）

`backend`：Tridentから1つ以上のストレージバックエンドを取得
`snapshot`：Astra Tridentから1つ以上のスナップショットを取得
`storageclass`：Astra Tridentから1つ以上のストレージクラスを取得
`volume`：Astra Tridentから1つ以上のボリュームを取得

フラグ

`-h`、 `--help`：ボリュームのヘルプ。
`--parentOfSubordinate string`：クエリを下位のソースボリュームに制限します。
`--subordinateOf string`:クエリをボリュームの下位に制限します。

イメージ

使用 `images` Astra Tridentが必要とするコンテナイメージの表を出力するためのフラグ。

```
tridentctl images [flags]
```

フラグ

`-h`、 `--help`:画像のヘルプ。
`-v`、 `--k8s-version string`: Kubernetesクラスタ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

を使用します `import volume` コマンドを使用して、既存のボリュームをAstra Tridentにインポートします。

```
tridentctl import volume <backendName> <volumeName> [flags]
```

エイリアス

`volume`、 `v`

フラグ

`-f`、 `--filename string`: YAMLまたはJSON PVCファイルへのパス。
`-h`、 `--help`：ボリュームのヘルプ。
`--no-manage`：PV/PVCのみを作成します。ボリュームのライフサイクル管理を想定しないでください。

をインストールします

を使用します `install` Astra Tridentのインストールにフラグを付けます。

```
tridentctl install [flags]
```

フラグ

`--autosupport-image string`：AutoSupportテレメトリ用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プロキシのアドレス/ポート。
`--enable-node-prep`：ノードに必要なパッケージをインストールします。
`--generate-custom-yaml`:インストールを行わずにYAMLファイルを生成します。
`-h`、 `--help`:インストールのヘルプ。
`--http-request-timeout`：TridentコントローラのREST APIのHTTP要求タイムアウトを上書きします（デフォルトは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3分0）。
`--kubelet-dir string`: kubeletの内部状態のホストの場所(デフォルトは/var/lib/kubelet)
`--log-format string`: Astra Tridentのログ形式(テキスト、JSON](デフォルトは「text」)。
`--pv string`: Astra Tridentが使用するレガシーPVの名前は、存在しないことを確認します(デフォルトは"trident")。
`--pvc string`：Astra Tridentで使用されている従来のPVCの名前。このPVCが存在しないことを確認します（デフォルトは「trident」）。
`--silence-autosupport`：AutoSupport バンドルを自動的にネットアップに送信しない（デフォルトはtrue）。
`--silent`:インストール中は、ほとんどの出力を無効にします。
`--trident-image string`：インストールするAstra Tridentのイメージ
`--use-custom-yaml`: setupディレクトリに存在する既存のYAMLファイルを使用します。
`--use-ipv6`：Astra Tridentの通信にIPv6を使用

ログ

使用 `logs` Astra Tridentからログを印刷するためのフラグ。

```
tridentctl logs [flags]
```

フラグ

`-a`、 `--archive`：特に指定がないかぎり、すべてのログを含むサポートアーカイブを作成します。
`-h`、 `--help`:ログのヘルプ。
`-l`、 `--log string`：Astra Tridentのログが表示されます。trident | auto | trident-operator | all （デフォルトは「 auto 」）のいずれかです。
`--node string`：ノードポッドログの収集元のKubernetesノード名。
`-p`、 `--previous`:以前のコンテナインスタンスのログが存在する場合は、それを取得します。
`--sidecars`:サイドカーコンテナのログを取得します。

送信

を使用します `send` Astra Tridentからリソースを送信するコマンド。

```
tridentctl send [option]
```

オプション（ **Options** ）

`autosupport`：ネットアップにAutoSupport アーカイブを送信します。

をアンインストールします

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

```
tridentctl uninstall [flags]
```

フラグ

`-h, --help`:アンインストールのヘルプ。
`--silent`:アンインストール中のほとんどの出力を無効にします。

更新

を使用します `update` Astra Tridentでリソースを変更するコマンド。

```
tridentctl update [option]
```

オプション（ **Options** ）

`backend`：Astra Tridentのバックエンドを更新。

バックエンドの状態を更新

を使用します `update backend state` バックエンド処理を一時停止または再開するコマンド。

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

フラグ

`-h`、 `--help`:バックエンド状態のヘルプ。
`--user-state`:に設定 `suspended` バックエンド処理を一時停止します。をに設定します `normal` バックエンド処理を再開します。に設定すると `suspended`：

- `AddVolume`、 `CloneVolume`、 `Import Volume`、 `ResizeVolume` は一時停止しています。
- `PublishVolume`、 `UnPublishVolume`、 `CreateSnapshot`、 `GetSnapshot`、 `RestoreSnapshot`、 `DeleteSnapshot`、 `RemoveVolume`、 `GetVolumeExternal`、 `ReconcileNodeAccess` 引き続き使用できます。

バージョン

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

```
tridentctl version [flags]
```

フラグ

`--client`:クライアントバージョンのみ(サーバは不要)。
`-h, --help`:バージョンのヘルプ。

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

必要なもの

- Astra Trident がインストールされた Kubernetes クラスタ
- Prometheus インスタンス。これは a である場合もある "コンテナ化された Prometheus 環境" または、Prometheus をとして実行することもできます "ネイティブアプリケーション"。

手順 1 ： Prometheus ターゲットを定義する

Prometheus ターゲットを定義して指標を収集し、 Astra Trident が管理するバックエンド、作成するボリュームなどの情報を取得する必要があります。これ "ブログ" Prometheus と Grafana を Astra Trident とともに使用して指標を取得する方法について説明します。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 のメトリックを使用するには、「 trident-csi 」サービスを監視し、「 m etrics 」ポートを監視する Prometheus ServiceMonitor を作成する必要があります。ServiceMonitor のサンプルは次のようになります。

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

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

Astra Tridentから直接取得できる指標に加えて、kubeletは多くの指標を公開しています `kubelet_volume_*` 独自の指標エンドポイントを使用した指標。Kubelet では、接続されているボリュームに関する情報、およびポッドと、それが処理するその他の内部処理を確認できます。を参照してください "こちらをご覧ください"。

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

PromptQL 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、 PromptQL クエリーのいくつかを示します。

Trident の健常性情報を取得

- **Astra Trident** からの **HTTP 2XX** 応答の割合

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

- **Astra Trident** からのステータスコードによる **REST** 応答の割合

```
(sum (trident_rest_ops_seconds_total_count) by (status_code)  / scalar
(sum (trident_rest_ops_seconds_total_count))) * 100
```

- **Astra Trident** によって実行された処理の平均時間（ミリ秒）

```
sum by (operation)
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by
(operation)
(trident_operation_duration_milliseconds_count{success="true"})
```

Astra Trident の使用状況に関する情報を入手

- 平均体積サイズ

```
trident_volume_allocated_bytes/trident_volume_count
```

- 各バックエンドによってプロビジョニングされた合計ボリューム容量

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

個々のボリュームの使用状況を取得する

これは、 kubelet 指標も収集された場合にのみ有効になります。

- 各ボリュームの使用済みスペースの割合

```
kubelet_volume_stats_used_bytes / kubelet_volume_stats_capacity_bytes *
100
```

Astra Trident AutoSupport の計測データ

デフォルトでは、 Astra Trident は Prometheus 指標と基本バックエンド情報を毎日定期的にネットアップに送信します。

- Astra Trident が Prometheus 指標と基本バックエンド情報をネットアップに送信しないようにするには、Astra Trident のインストール時に「 --silence -autosupport 」フラグを渡します。
- Astra Trident は 'tridentctl send AutoSupport を介してコンテナ・ログをオンデマンドでネットアップ・サポートに送信することもできますAstra Trident をトリガーしてログをアップロードする必要があります。ログを送信する前に、ネットアップのに同意する必要がありますhttps://www.netapp.com/company/legal/privacy-policy/["プライバシーポリシー"^]。
- 指定しないと、 Astra Trident は過去 24 時間からログを取得します。
- ログの保持期間は、で指定できます `--since` フラグ。例： `tridentctl send autosupport --since=1h`。この情報は、を介して収集および送信されます `trident-autosupport` TridentがAstraと一緒にインストールされるコンテナ。コンテナイメージは、で取得できます "Trident AutoSupport の略"。
- Trident AutoSupport は、個人情報（ PII ）や個人情報を収集または送信しません。それにはが付いていま

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

Astra Trident から送信されるペイロードの例を次に示します。

```
---
items:
- backendUUID: ff3852e1-18a5-4df4-b2d3-f59f829627ed
  protocol: file
  config:
    version: 1
    storageDriverName: ontap-nas
    debug: false
    debugTraceFlags:
    disableDelete: false
    serialNumbers:
    - nwkvzfanek_SN
    limitVolumeSize: ''
  state: online
  online: true
```

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

Astra Trident の指標を無効化

- メトリックがレポートされないようにするには '--generate-custom-yaml' フラグを使用してカスタム YAML を生成し ' これらを編集して 'trident-main' コンテナに対して --metrics フラグが呼び出されないようにします

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。

このタスクについて

- アップグレード、依存関係の問題、アップグレードの失敗や不完全な完了後に観察され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、 "Astra TridentとCRDを完全に削除"。

作業を開始する前に

Kubernetesクラスタの運用を停止する場合は、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. 使用 `kubectl get pods -n trident` ポッドを検査するために。
   - オペレータポッドがない場合は、を使用してAstra Tridentがインストールされています `tridentctl`。
   - オペレータポッドがある場合、Astra Tridentは手動またはHelmを使用してインストールされています。
2. オペレータポッドがある場合は、を使用します `kubectl describe tproc trident` をクリックし、Helmを使用してAstra Tridentがインストールされたかどうかを確認します。
   - Helmラベルがある場合は、Helmを使用してAstra Tridentがインストールされています。
   - Helmラベルがない場合は、Astra TridentをTridentオペレータを使用して手動でインストールしています。

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. 編集 **`TridentOrchestrator` CR**を実行し、アンインストールフラグを設定します：

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

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

2. 削除 **`TridentOrchestrator`**： `TridentOrchestrator` Astra Tridentの導入に使用したCRでは、Tridentをアンインストールするようオペレータに指示します。オペレータがの削除を処理します `TridentOrchestrator` さらに、Astra Tridentの導入とデプロイを削除し、インストールの一部として作成したTridentポッドを削除します。

   ```
   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 'helm uninstall を使用してアンインストールできます

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

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

のアンインストール `tridentctl` インストール

を使用します `uninstall` のコマンド `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、次の手順を実行します。

```
./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

- の導入がすべてを満たしていることを確認します "要件"。
- サポートされているバージョンの Docker がインストールされていることを確認します。Docker のバージョンが最新でない場合は、 "インストールまたは更新します"。

```
docker --version
```

- プロトコルの前提条件がホストにインストールおよび設定されていることを確認します。

NFSツール

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

RHEL 8以降

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

Ubuntu

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

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
```

> 「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。

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

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

6. 'iSCSI' を有効にして開始します

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

Ubuntu

1. 次のシステムパッケージをインストールします。

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

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

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

3. スキャンを手動に設定：

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

4. マルチパスを有効化：

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

> 「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。

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

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

NVMeツール

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

> - 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）

> 作業を開始する前に
>
> 従来のデーモン方式で Astra Trident 以前の Docker 1.13 / 17.03 を使用していた場合は、マネージドプラグイン方式を使用する前に Astra Trident プロセスを停止し、 Docker デーモンを再起動してください。

1. 実行中のインスタンスをすべて停止します。

```
pkill /usr/local/bin/netappdvp
pkill /usr/local/bin/trident
```

2. Docker を再起動します。

```
systemctl restart docker
```

3. Docker Engine 17.03 （新しい 1.13 ）以降がインストールされていることを確認します。

```
docker --version
```

バージョンが最新でない場合は、 "インストール環境をインストールまたは更新します"。

手順

1. 構成ファイルを作成し、次のオプションを指定します。
   - `config`: デフォルトのファイル名は 'config.json ですが ' ファイル名に config オプションを指定することで ' 選択した任意の名前を使用できます構成ファイルは ' ホスト・システムの /etc/netappdvp ディレクトリに格納されている必要があります
   - 「 log-level 」：ログレベルを指定します（「 debug` 」、「 info 」、「 warn 」、「 error 」、「 fatal 」）。デフォルトは「 info 」です。
   - `ebug`: デバッグログを有効にするかどうかを指定します。デフォルトは false です。true の場合、ログレベルを上書きします。
     i. 構成ファイルの場所を作成します。

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

     ii. 構成ファイルを作成します

     ```
     cat << EOF > /etc/netappdvp/config.json
     {
         "version": 1,
         "storageDriverName": "ontap-nas",
         "managementLIF": "10.0.0.1",
         "dataLIF": "10.0.0.2",
         "svm": "svm_nfs",
         "username": "vsadmin",
         "password": "password",
         "aggregate": "aggr1"
     }
     EOF
     ```

2. マネージドプラグインシステムを使用して Astra Trident を起動交換してください `<version>` 使用しているプラグインのバージョン（xxx.xxx.xxx.xxx）を使用している必要があります。

```
docker plugin install --grant-all-permissions --alias netapp
netapp/trident-plugin:<version> config=myConfigFile.json
```

3. Astra Trident を使用して、構成したシステムのストレージを使用しましょう。

a. 「 firstVolume 」という名前のボリュームを作成します。

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

b. コンテナの開始時にデフォルトのボリュームを作成します。

```
docker run --rm -it --volume-driver netapp --volume
secondVolume:/my_vol alpine ash
```

c. ボリューム「 firstVolume 」を削除します。

```
docker volume rm firstVolume
```

従来の方法（バージョン **1.12** 以前）

作業を開始する前に

1. バージョン 1.10 以降の Docker がインストールされていることを確認します。

```
docker --version
```

使用しているバージョンが最新でない場合は、インストールを更新します。

```
curl -fsSL https://get.docker.com/ | sh
```

または "ご使用のディストリビューションの指示に従ってください"。

2. NFS または iSCSI がシステムに対して設定されていることを確認します。

手順

1. NetApp Docker Volume Plugin をインストールして設定します。

a. アプリケーションをダウンロードして開梱します。

```
wget
https://github.com/NetApp/trident/releases/download/v24.10.0/trident-
installer-24.02.0.tar.gz
tar zxf trident-installer-24.02.0.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

システムベースのシステムのサンプルユニットファイルは、から入手できます `contrib/trident.service.example` Gitリポジトリで実行します。RHELでファイルを使用するには、次の手順を実行します。

1. ファイルを正しい場所にコピーします。

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

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

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

```
systemctl daemon-reload
```

4. サービスを有効にします。

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

```
systemctl enable trident
```

5. サービスを開始します。

```
systemctl start trident
```

6. ステータスを確認します。

```
systemctl status trident
```

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

Astra Trident をアップグレードまたはアンインストールする

使用中のボリュームに影響を与えることなく、 Astra Trident for Docker を安全にアップグレードできます。アップグレード処理中に、プラグインで指示された「 Occker volume 」コマンドが正常に実行されず、プラグインが再度実行されるまでアプリケーションはボリュームをマウントできません。ほとんどの場合、これは秒の問題です。

アップグレード

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

手順

1. 既存のボリュームを表示します。

```
docker volume ls
DRIVER              VOLUME NAME
netapp:latest       my_volume
```

2. プラグインを無効にします。

```
docker plugin disable -f netapp:latest
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
```

3. プラグインをアップグレードします。

```
docker plugin upgrade --skip-remote-check --grant-all-permissions
netapp:latest netapp/trident-plugin:21.07
```

nDVP に代わる、 Astra Trident の 18.01 リリース。NetApp/ndvp-plugin イメージから NetApp/trident-plugin` イメージに直接アップグレードする必要があります。

4. プラグインを有効にします。

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

5. プラグインが有効になっていることを確認します。

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

6. ボリュームが表示されることを確認します。

```
docker volume ls
DRIVER              VOLUME NAME
netapp:latest       my_volume
```

> 古いバージョンの Astra Trident （ 20.10 より前）から Astra Trident 20.10 以降にアップグレードすると、エラーが発生する場合があります。詳細については、を参照してください "既知の問題"。このエラーが発生した場合は、まずプラグインを無効にしてからプラグインを削除し、次に追加のconfigパラメータを渡して、必要なAstra Tridentバージョンをインストールします。
> `docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant-all-permissions config=config.json`

をアンインストールします

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

手順

1. プラグインで作成されたボリュームをすべて削除します。
2. プラグインを無効にします。

```
docker plugin disable netapp:latest
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
```

3. プラグインを削除します。

```
docker plugin rm netapp:latest
```

ボリュームを操作します

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

ボリュームを作成します

- デフォルトの名前を使用して、ドライバでボリュームを作成します。

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

- 特定の Astra Trident インスタンスを使用してボリュームを作成します。

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

何も指定しない場合 "オプション（ Options ）"、ドライバのデフォルトが使用されます。

- デフォルトのボリュームサイズを上書きします。次の例を参照して、ドライバで 20GiB ボリュームを作成してください。

```
docker volume create -d netapp --name my_vol --opt size=20G
```

ボリュームサイズは、オプションの単位（ 10G 、 20GB 、 3TiB など）を含む整数値で指定します。単位を指定しない場合、デフォルトは g ですサイズの単位は、 2 の累乗（ B 、 KiB 、 MiB 、 GiB 、 TiB ）または 10 の累乗（ B 、 KB 、 MB 、 GB 、 TB ）のいずれかです。略記単位では、 2 の累乗が使用されます（ G=GiB 、 T=TiB 、…）。

ボリュームを削除します

- 他の Docker ボリュームと同様にボリュームを削除します。

```
docker volume rm firstVolume
```

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

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

ボリュームのクローンを作成します

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

- ボリュームを調べて Snapshot を列挙します。

```
docker volume inspect <volume_name>
```

- 既存のボリュームから新しいボリュームを作成します。その結果、新しい Snapshot が作成されます。

```
docker volume create -d <driver_name> --name <new_name> -o
from=<source_docker_volume>
```

- ボリューム上の既存の Snapshot から新しいボリュームを作成します。新しい Snapshot は作成されません。

```
docker volume create -d <driver_name> --name <new_name> -o
from=<source_docker_volume> -o fromSnapshot=<source_snap_name>
```

例

```
docker volume inspect firstVolume

[
    {
        "Driver": "ontap-nas",
        "Labels": null,
        "Mountpoint": "/var/lib/docker-volumes/ontap-
nas/netappdvp_firstVolume",
        "Name": "firstVolume",
        "Options": {},
        "Scope": "global",
        "Status": {
            "Snapshots": [
                {
                    "Created": "2017-02-10T19:05:00Z",
                    "Name": "hourly.2017-02-10_1505"
                }
            ]
        }
    }
]

docker volume create -d ontap-nas --name clonedVolume -o from=firstVolume
clonedVolume

docker volume rm clonedVolume
docker volume create -d ontap-nas --name volFromSnap -o from=firstVolume
-o fromSnapshot=hourly.2017-02-10_1505
volFromSnap

docker volume rm volFromSnap
```

外部で作成されたボリュームにアクセス

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

ドライバ固有のボリュームオプション

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

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

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

ONTAP ボリュームのオプション

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

オプション	説明
「 size 」	ボリュームのサイズ。デフォルトは 1GiB です。
「平和のための準備」を参照してください	ボリュームをシンプロビジョニングまたはシックプロビジョニングします。デフォルトはシンです。有効な値は 'none(シン・プロビジョニング) と 'volume(シック・プロビジョニング) です
「ナプショットポリシー」	Snapshot ポリシーが目的の値に設定されます。デフォルトは「 none 」です。つまり、ボリュームに対してスナップショットが自動的に作成されることはありません。ストレージ管理者によって変更されていない限り、「 default 」という名前のポリシーがすべての ONTAP システムに存在し、 6 個の時間単位 Snapshot 、 2 個の日単位 Snapshot 、および 2 個の週単位 Snapshot を作成して保持します。スナップショットに保存されているデータは ' ボリューム内の任意のディレクトリ内の .snapshot ディレクトリに移動してリカバリできます
「スナップショット予約」	これにより、 Snapshot リザーブの割合が希望する値に設定されます。デフォルト値は no で、 Snapshot ポリシーを選択した場合は ONTAP によって snapshotReserve が選択されます（通常は 5% ）。 Snapshot ポリシーがない場合は 0% が選択されます。構成ファイルのすべての ONTAP バックエンドに対して snapshotReserve のデフォルト値を設定できます。また、この値は、 ONTAP-NAS-エコノミー を除くすべての ONTAP バックエンドでボリューム作成オプションとして使用できます。
'plitOnClone	ボリュームをクローニングすると、そのクローンが原因 ONTAP によって親から即座にスプリットされます。デフォルトはです `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
「暗号化」	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは「false」です。このオプションを使用するには、クラスタでNVE のライセンスが設定され、有効になっている必要があります。 NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームが NAEに有効になります。 詳細については、以下を参照してください。 "Astra TridentとNVEおよびNAEの相互運用性"。
階層ポリシー	ボリュームに使用する階層化ポリシーを設定します。これにより、アクセス頻度の低いコールドデータをクラウド階層に移動するかどうかが決まります。

以下は、 NFS * のみ * 用の追加オプションです。

オプション	説明
「 unixPermissions 」	これにより、ボリューム自体の権限セットを制御できます。デフォルトでは ' アクセス権は '--rwxr-xr-x' または数値表記 0755 に設定され 'root' は所有者になりますテキスト形式または数値形式のどちらかを使用できます。
「スナップショット方向」	これをに設定します `true` がを作成します `.snapshot` ボリュームにアクセスしているクライアントから認識できるディレクトリ。デフォルト値はです `false`の可視性を意味します `.snapshot` ディレクトリはデフォルトで無効になっています。一部のイメージ（公式のMySQLイメージなど）が、`.snapshot` ディレクトリが表示されます。
「 exportPolicy 」と入力します	ボリュームで使用するエクスポートポリシーを設定します。デフォルトは「デフォルト」です。
'ecurityStyle'	ボリュームへのアクセスに使用するセキュリティ形式を設定します。デフォルトは「 unix 」です。有効な値は「 unix 」と「 immimixed 」です。

以下の追加オプションは、 iSCSI * のみ * 用です。

オプション	説明
「 filesystemtype 」です	iSCSI ボリュームのフォーマットに使用するファイルシステムを設定します。デフォルトは「 ext4 」です。有効な値は「 ext3 」、「 ext4 」、「 xfs 」です。
「平和の配分」	これをに設定します `false` LUNのスペース割り当て機能を無効にします。デフォルト値はです `true`つまり、ボリュームのスペースが不足し、ボリューム内のLUNに書き込みを受け付けられなくなったときに、ONTAP からホストに通知されます。また、このオプションで ONTAP 、ホストでデータが削除された時点での自動スペース再生も有効になります。

例

以下の例を参照してください。

- 10GiB ボリュームを作成します。

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

- Snapshot を使用して 100GiB のボリュームを作成します。

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

- setuid ビットが有効になっているボリュームを作成します。

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

最小ボリュームサイズは 20MiB です。

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

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

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

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

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

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

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

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

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

Element ソフトウェアのボリュームオプション

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

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

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

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

例

QoS 定義を含む次のサンプル構成ファイルを参照してください。

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

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

- 10GiB の Gold ボリュームを作成します。

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

- 100GiB Bronze ボリュームを作成します。

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

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

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

   標準的なロギングレベルでは、ほとんどの問題を診断できます。十分でない場合は、デバッグロギングを有効にすることができます。

2. デバッグロギングをイネーブルにするには、デバッグロギングをイネーブルにしてプラグインをインストールします。

```
docker plugin install netapp/trident-plugin:<version> --alias <alias>
debug=true
```

   または、プラグインがすでにインストールされている場合にデバッグログを有効にします。

```
docker plugin disable <plugin>
docker plugin set <plugin> debug=true
docker plugin enable <plugin>
```

3. ホストでバイナリ自体を実行している場合、ログはホストの `/var/log/netappdvp` ディレクトリ。デバッグロギングを有効にするには、を指定します `-debug` プラグインを実行すると、

一般的なトラブルシューティングのヒント

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

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

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

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

複数の **Astra Trident** インスタンスを管理

複数のストレージ構成を同時に使用する必要がある場合は、 Trident の複数のインスタンスが必要です。複数のインスタンスの鍵は、コンテナ化されたプラグインでは「 --alias 」オプション、ホストで Trident をインスタンス化する場合は「 --volume-driver 」オプションを使用して、それぞれ異なる名前を指定することです。

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

1. エイリアスと構成ファイルを指定して、最初のインスタンスを起動します。

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

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

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

3. ドライバ名としてエイリアスを指定するボリュームを作成します。

   たとえば、 gold ボリュームの場合：

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

   たとえば、 Silver ボリュームの場合：

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

従来の（バージョン **1.12** 以前）の場合の手順

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

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

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

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

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

   たとえば、 NFS の場合：

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

   たとえば、 iSCSI の場合：

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

ストレージ構成オプション

Astra Trident 構成で使用できる設定オプションを確認してください。

グローバル構成オプション

以下の設定オプションは、使用するストレージプラットフォームに関係なく、すべての Astra Trident 構成に適用されます。

オプション	説明	例
「バージョン」	構成ファイルのバージョン番号	`1`
'torageDriverName'	ストレージドライバの名前	`ontap-nas`、 `ontap-san`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `solidfire-san`
'toragePrefix'	ボリューム名のオプションのプレフィックス。デフォルト： `netappdvp_`。	`staging_`

オプション	説明	例
「 limitVolumeSize 」と入力します	ボリュームサイズに関するオプションの制限。デフォルト：""（強制なし）	`10g`

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

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

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

ONTAP の設定

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

オプション	説明	例
「管理 LIF 」	ONTAP 管理 LIF の IP アドレス。Fully Qualified Domain Name （ FQDN ；完全修飾ドメイン名）を指定できます。	`10.0.0.1`

オプション	説明	例
「重複排除	プロトコル LIF の IP アドレス。 • ONTAP NASドライバ*:を指定することをお勧めします `dataLIF`。指定しない場合は、Astra TridentがSVMからデータLIFを取得します。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。 • ONTAP SANドライバ*: iSCSIには指定しないでくださいAstra Tridentが使用 "ONTAPの選択的LUNマップ" iSCSI LIFを検出するには、マルチパスセッションを確立する必要があります。の場合は警告が生成されます `dataLIF` は明示的に定義されます。	`10.0.0.2`
'VM'	使用する Storage Virtual Machine（管理 LIF がクラスタ LIF である場合は必須）	`svm_nfs`
「ユーザ名」	ストレージデバイスに接続するユーザ名	`vsadmin`
「 password 」と入力します	ストレージ・デバイスに接続するためのパスワード	`secret`
「集約」	プロビジョニング用のアグリゲート（オプション。設定する場合は SVM に割り当てる必要があります）。「 ONTAP-NAS-flexgroup 」ドライバの場合、このオプションは無視されます。SVM に割り当てられたすべてのアグリゲートを使用して FlexGroup ボリュームがプロビジョニングされます。	`aggr1`
「 AggreglimitateUsage 」と入力します	オプション。使用率がこの割合を超えている場合は、プロビジョニングを失敗させます	`75%`

オプション	説明	例
「 nfsvMountOptions 」のように入力します	NFS マウントオプションのきめ細かな制御。デフォルトは「 -o nfsvers=3 」です。* 「 ONTAP-NAS' 」および「 ONTAP-NAS-エコノミー 」ドライバ専用です。 "ここでは、 NFS ホストの設定情報を参照してください"。	`-o nfsvers=4`
「 igroupName 」と入力します	Astra Tridentはノード単位で作成、管理します `igroups` として `netappdvp`。 この値は変更したり省略したりすることはできません。 でのみ使用できます **`ontap-san`** ドライバ。	`netappdvp`
「 limitVolumeSize 」と入力します	最大要求可能ボリュームサイズと qtree 親ボリュームサイズ。* 「 ONTAP-NAS-エコノミー 」ドライバの場合、このオプションにより、作成する FlexVol のサイズも制限されます。 *	`300g`
qtreesPerFlexvol`	FlexVol あたりの最大 qtree 数は [50 、 300] の範囲で指定する必要があります。デフォルトは 200 です。 *のため `ontap-nas-economy` ドライバ。このオプションを使用すると、FlexVol あたりの最大qtree数をカスタマイズできます。	`300`
`sanType`	サポート対象 **`ontap-san`** ドライバーのみ。 を使用して選択 `iscsi` iSCSIの場合または `nvme` （NVMe/TCPの場合）。	`iscsi` 空白の場合

作成するすべてのボリュームでデフォルトのオプションを指定しなくても済むようになっています。

オプション	説明	例
「平和のための準備」を参照してください	スペースリザベーションモード `none` （シンプロビジョニング）または `volume` （シック）	「 NONE 」

オプション	説明	例
「ナプショットポリシー」	使用するSnapshotポリシー。デフォルトはです `none`	「 NONE 」
「スナップショット予約」	Snapshotリザーブの割合。デフォルトはONTAP のデフォルトをそのまま使用する場合はです	`10`
'plitOnClone	作成時に親からクローンをスプリットします。デフォルトはです `false`	「偽」
「暗号化」	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは「false」です。このオプションを使用するには、クラスタでNVE のライセンスが設定され、有効になっている必要があります。 NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームが NAEに有効になります。 詳細については、以下を参照してください。 "Astra TridentとNVEおよびNAEの相互運用性"。	正しいです
「 unixPermissions 」	プロビジョニングされたNFSボリュームのNASオプション。デフォルトはです `777`	`777`
「スナップショット方向」	にアクセスするためのNASオプション `.snapshot` ディレクトリ。デフォルトはです `false`	「真」
「 exportPolicy 」と入力します	NFSエクスポートポリシーで使用するNASオプション。デフォルトはです `default`	`default`
'ecurityStyle'	プロビジョニングされたNFSボリュームにアクセスするためのNASオプション。 NFSのサポート `mixed` および `unix` セキュリティ形式デフォルトはです `unix`。	`unix`
「 filesystemtype 」です	ファイルシステムタイプを選択するためのSANオプション。デフォルトはです `ext4`	`xfs`
階層ポリシー	使用する階層化ポリシー。デフォルトはです `none`; `snapshot-only` ONTAP 9.5より前のSVM-DR構成の場合	「 NONE 」

スケーリングオプション

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

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

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

> Swarm は複数のノード間でのボリューム作成のオーケストレーションを行わないため 'ONTAP-NAS-エコノミー のドライバは現在 Docker Swarm ではサポートされていません

「 ONTAP と SAN の経済性」のドライバは、自動的に管理される FlexVol の共有プール内で、 ONTAP LUN として Docker ボリュームを作成します。この方法により、各 FlexVol が 1 つの LUN に制限されることはなく、 SAN ワークロードのスケーラビリティが向上します。ストレージアレイに応じて、 ONTAP はクラスタあたり最大 16384 個の LUN をサポートします。このドライバは、ボリュームが下位の LUN であるため、 Docker ボリューム単位の Snapshot とクローニングをサポートします。

「 ONTAP-NAS-flexgroup 」ドライバを選択して、数十億個のファイルを含むペタバイト規模に拡張可能な 1 つのボリュームに並列処理を増やすことができます。FlexGroup のユースケースとしては、 AI / ML / DL 、ビッグデータと分析、ソフトウェアのビルド、ストリーミング、ファイルリポジトリなどが考えられます。Trident は、 FlexGroup ボリュームのプロビジョニング時に SVM に割り当てられたすべてのアグリゲートを使用します。Trident での FlexGroup のサポートでは、次の点も考慮する必要があります。

- ONTAP バージョン 9.2 以降が必要です。
- 本ドキュメントの執筆時点では、 FlexGroup は NFS v3 のみをサポートしています。
- SVM で 64 ビットの NFSv3 ID を有効にすることを推奨します。
- 推奨される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の詳細については、を参照してください "『 NetApp FlexGroup Volume Best Practices and Implementation Guide 』にある、ボリュームへの移行に関するセクション"。

同じ環境で高度な機能と大規模な拡張性を実現するために 'ONTAP-NAS' を使用して Docker Volume Plugin の複数のインスタンスを実行し ' もう 1 つは「 ONTAP-NAS-エコノミー 」を使用して実行できます

ONTAP 構成ファイルの例

<code>ontap-nas</code> ドライバの**NFS**の例

```
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1",
    "defaults": {
      "size": "10G",
      "spaceReserve": "none",
      "exportPolicy": "default"
    }
}
```

<code>ontap-nas-flexgroup</code> ドライバの**NFS**の例

```
{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "password",
    "defaults": {
      "size": "100G",
      "spaceReserve": "none",
      "exportPolicy": "default"
    }
}
```

<code>ontap-nas-economy</code> ドライバの**NFS**の例

```
{
    "version": 1,
    "storageDriverName": "ontap-nas-economy",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1"
}
```

<code>ontap-san</code> ドライバの**iSCSI**の例

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1",
    "igroupName": "netappdvp"
}
```

<code>ontap-san-economy</code> ドライバの**NFS**の例

```
{
    "version": 1,
    "storageDriverName": "ontap-san-economy",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi_eco",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1",
    "igroupName": "netappdvp"
}
```

<code>ontap-san</code> ドライバの**NVMe/TCP**の例

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

Element ソフトウェアの設定

Element ソフトウェア（ NetApp HCI / SolidFire ）を使用する場合は、グローバルな設定値のほかに、以下のオプションも使用できます。

オプション	説明	例
「エンドポイント」	\ <a href="https://<login>:<password>@<mvip>/json-rpc/<element-version>" class="bare">https://<login>:<password>@<mvip>/json-rpc/<element-version>;	\ https://admin:admin@192.168.160.3/json-rpc/8.0
「 VIP 」	iSCSI の IP アドレスとポート	10.0.0.7 ： 3260
「 tenantname 」	使用する SolidFire テナント（見つからない場合に作成）	`docker`
「 InitiatorIFCace 」	iSCSI トラフィックをデフォルト以外のインターフェイスに制限する場合は、インターフェイスを指定します	`default`
「タイプ」	QoS の仕様	以下の例を参照してください

オプション	説明	例
「 LegacyNamePrefix 」のように入力します	アップグレードされた Trident インストールのプレフィックス。1.3.2 より前のバージョンのTridentを使用していて、既存のボリュームでアップグレードを実行した場合は、volume-nameメソッドでマッピングされた古いボリュームにアクセスするためにこの値を設定する必要があります。	`netappdvp-`

「 olidfire -san 」ドライバは Docker Swarm をサポートしていません。

Element ソフトウェア構成ファイルの例

```
{
    "version": 1,
    "storageDriverName": "solidfire-san",
    "Endpoint": "https://admin:admin@192.168.160.3/json-rpc/8.0",
    "SVIP": "10.0.0.7:3260",
    "TenantName": "docker",
    "InitiatorIFace": "default",
    "Types": [
        {
            "Type": "Bronze",
            "Qos": {
                "minIOPS": 1000,
                "maxIOPS": 2000,
                "burstIOPS": 4000
            }
        },
        {
            "Type": "Silver",
            "Qos": {
                "minIOPS": 4000,
                "maxIOPS": 6000,
                "burstIOPS": 8000
            }
        },
        {
            "Type": "Gold",
            "Qos": {
                "minIOPS": 6000,
                "maxIOPS": 8000,
                "burstIOPS": 10000
            }
        }
    ]
}
```

既知の問題および制限事項

Astra Trident と Docker を使用する際の既知の問題と制限事項について説明しています。

Trident Docker Volume Plugin を旧バージョンから **20.10** 以降にアップグレードすると、該当するファイルエラーまたはディレクトリエラーなしでアップグレードが失敗します。

回避策

1. プラグインを無効にします。

```
docker plugin disable -f netapp:latest
```

2. プラグインを削除します。

```
docker plugin rm -f netapp:latest
```

3. 追加の 'config' パラメータを指定して ' プラグインを再インストールします

```
docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant
-all-permissions config=config.json
```

ボリューム名は 2 文字以上にする必要があります。

これは Docker クライアントの制限事項です。クライアントは、 1 文字の名前を Windows パスと解釈します。 "バグ 25773 を参照"。

Docker Swarm には、 Astra Trident がストレージやドライバのあらゆる組み合わせでサポートしないようにする一定の動作があります。

- Docker Swarm は現在、ボリューム ID ではなくボリューム名を一意のボリューム識別子として使用します。
- ボリューム要求は、 Swarm クラスタ内の各ノードに同時に送信されます。
- ボリュームプラグイン（ Astra Trident を含む）は、 Swarm クラスタ内の各ノードで個別に実行する必要があります。ONTAP の仕組みと 'ONTAP-NAS' および 'ONTAP-SAN' ドライバの機能の仕組みにより ' これらの制限の範囲内で動作できるのはこれらのドライバだけです

その他のドライバには、競合状態などの問題があります。このような問題が発生すると、ボリュームを同じ名前で異なる ID にする機能が Element に備わっているため、「勝者」を明確にせずに 1 回の要求で大量のボリュームを作成できるようになります。

ネットアップは Docker チームにフィードバックを提供しましたが、今後の変更の兆候はありません。

FlexGroup をプロビジョニングする場合、プロビジョニングする FlexGroup と共通のアグリゲートが 2 つ目の FlexGroup に 1 つ以上あると、 ONTAP は 2 つ目の FlexGroup をプロビジョニングしません。

ベストプラクティスと推奨事項

導入

Astra Trident の導入時には、ここに示す推奨事項を使用してください。

専用のネームスペースに導入します

"ネームスペース" 異なるアプリケーション間で管理を分離できるため、リソース共有の障壁となります。たとえば、あるネームスペースの PVC を別のネームスペースから使用することはできません。Astra Trident は、Kubernetes クラスタ内のすべてのネームスペースに PV リソースを提供するため、権限が昇格されたサービスアカウントを利用します。

また、 Trident ポッドにアクセスすると、ユーザがストレージシステムのクレデンシャルやその他の機密情報にアクセスできるようになります。アプリケーションユーザと管理アプリケーションが Trident オブジェクト定義またはポッド自体にアクセスできないようにすることが重要です。

クォータと範囲制限を使用してストレージ消費を制御します

Kubernetes には、 2 つの機能があります。これらの機能を組み合わせることで、アプリケーションによるリソース消費を制限する強力なメカニズムが提供されます。。 "ストレージクォータメカニズム" 管理者は、グローバルおよびストレージクラス固有の、容量とオブジェクト数の使用制限をネームスペース単位で実装できます。さらに、を使用します "範囲制限" 要求がプロビジョニングツールに転送される前に、 PVC 要求が最小値と最大値の両方の範囲内にあることを確認します。

これらの値はネームスペース単位で定義されます。つまり、各ネームスペースに、リソースの要件に応じた値を定義する必要があります。の詳細については、こちらを参照してください "クォータの活用方法"。

ストレージ構成

ネットアップポートフォリオの各ストレージプラットフォームには、コンテナ化されたアプリケーションやそうでないアプリケーションに役立つ独自の機能があります。

プラットフォームの概要

Trident は ONTAP や Element と連携1 つのプラットフォームが他のプラットフォームよりもすべてのアプリケーションとシナリオに適しているわけではありませんが、プラットフォームを選択する際には、アプリケーションのニーズとデバイスを管理するチームを考慮する必要があります。

使用するプロトコルに対応したホストオペレーティングシステムのベースラインベストプラクティスに従う必要があります。必要に応じて、アプリケーションのベストプラクティスを適用する際に、バックエンド、ストレージクラス、 PVC の設定を利用して、特定のアプリケーションのストレージを最適化することもできます。

ONTAP と **Cloud Volumes ONTAP** のベストプラクティス

Trident 向けに ONTAP と Cloud Volumes ONTAP を設定するためのベストプラクティスをご確認ください。

次に示す推奨事項は、 Trident によって動的にプロビジョニングされたボリュームを消費するコンテナ化されたワークロード用に ONTAP を設定する際のガイドラインです。それぞれの要件を考慮し、環境内で適切かどうかを評価する必要があります。

Trident 専用の **SVM** を使用

Storage Virtual Machine （ SVM ）を使用すると、 ONTAP システムのテナントを分離し、管理者が分離できます。SVM をアプリケーション専用にしておくと、権限の委譲が可能になり、リソース消費を制限するためのベストプラクティスを適用できます。

SVM の管理には、いくつかのオプションを使用できます。

- バックエンド構成でクラスタ管理インターフェイスを適切なクレデンシャルとともに指定し、 SVM 名を指定します。
- ONTAP System Manager または CLI を使用して、 SVM 専用の管理インターフェイスを作成します。
- NFS データインターフェイスで管理ロールを共有します。

いずれの場合も、インターフェイスは DNS にあり、 Trident の設定時には DNS 名を使用する必要があります。これにより、ネットワーク ID を保持しなくても SVM-DR などの一部の DR シナリオが簡単になります。

専用の管理 LIF または共有の管理 LIF を SVM に使用する方法は推奨されませんが、ネットワークセキュリティポリシーを選択した方法と一致させる必要があります。最大の柔軟性を確保するには、どのような場合でも DNS 経由で管理 LIF にアクセスできるようにします "SVM-DR" Trident と組み合わせて使用できます。

最大ボリューム数を制限します

ONTAP ストレージシステムの最大ボリューム数は、ソフトウェアのバージョンとハードウェアプラットフォームによって異なります。を参照してください "NetApp Hardware Universe の略" 具体的な制限については、使用しているプラットフォームと ONTAP のバージョンに対応しています。ボリューム数を使い果たした場合、 Trident のプロビジョニング処理だけでなく、すべてのストレージ要求に対してプロビジョニング処理が失敗します。

Trident の「 ONTAP - NAS 」と「 ONTAP - SAN 」ドライバは、作成された Kubernetes 永続ボリューム（ PV ）ごとに FlexVol をプロビジョニングします。「 ONTAP-NAS-エコノミー 」ドライバは、 PVS 200 個につき約 1 つの FlexVol を作成します（ 50 〜 300 の範囲で構成可能）。「 ONTAP-SAN-エコノミー 」ドライバは、 PVS 100 個につき約 1 つの FlexVol を作成します（ 50 〜 200 の範囲で設定可能）。Trident がストレージシステム上の使用可能なボリュームをすべて消費しないようにするには、 SVM に制限を設定する必要があります。コマンドラインから実行できます。

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

「 mAX-VOLUMES 」の値は、環境に固有のいくつかの条件によって異なります。

- ONTAP クラスタ内の既存のボリュームの数
- 他のアプリケーション用に Trident 外部でプロビジョニングするボリュームの数
- Kubernetes アプリケーションで消費されると予想される永続ボリュームの数

「 mAX-volumes 」の値は、 ONTAP クラスタ内のすべてのノードでプロビジョニングされたボリュームの合計数であり、個々の ONTAP ノードではプロビジョニングされません。その結果、 ONTAP クラスタノードの

Trident でプロビジョニングされたボリュームの数が、別のノードよりもはるかに多い、または少ない場合があります。

たとえば、 2 ノードの ONTAP クラスタでは、最大 2 、 000 個の FlexVol をホストできます。最大ボリューム数を 1250 に設定していると、非常に妥当な結果が得られます。ただし、のみの場合 "アグリゲート" あるノードから SVM に割り当てられている場合や、あるノードから割り当てられたアグリゲートをプロビジョニングできない場合（容量など）は、他のノードが Trident でプロビジョニングされたすべてのボリュームのターゲットになります。これは、「 mAX-VOLUMES 」の値に達する前にそのノードのボリューム数の上限に達する可能性があることを意味し、 Trident とそのノードを使用する他のボリューム処理の両方に影響します。* クラスタ内の各ノードのアグリゲートを、 Trident が使用する SVM に同じ番号で確実に割り当てることで、この状況を回避できます。 *

Trident で作成できるボリュームの最大サイズを制限

Trident で作成できるボリュームの最大サイズを設定するには、「 backend.json 」の定義で「 limitVolumeSize 」パラメータを使用します。

ストレージアレイでボリュームサイズを制御するだけでなく、 Kubernetes の機能も利用する必要があります。

双方向 **CHAP** を使用するように **Trident** を設定します

バックエンド定義で CHAP イニシエータとターゲットのユーザ名とパスワードを指定し、 Trident を使用して SVM で CHAP を有効にすることができます。を使用する `useCHAP` バックエンド構成のパラメータであるTridentは、CHAPを使用してONTAP バックエンドのiSCSI接続を認証します。

SVM QoS ポリシーを作成して使用します

SVM に適用された ONTAP QoS ポリシーを使用すると、 Trident でプロビジョニングされたボリュームが使用できる IOPS の数が制限されます。これはに役立ちます "Bully を防止します" Trident SVM 外のワークロードに影響を及ぼす、制御不能なコンテナ。

SVM の QoS ポリシーはいくつかの手順で作成します。正確な情報については、ご使用の ONTAP バージョンのマニュアルを参照してください。次の例は、 SVM で使用可能な合計 IOPS を 5000 に制限する QoS ポリシーを作成します。

```
# create the policy group for the SVM
qos policy-group create -policy-group <policy_name> -vserver <svm_name>
-max-throughput 5000iops

# assign the policy group to the SVM, note this will not work
# if volumes or files in the SVM have existing QoS policies
vserver modify -vserver <svm_name> -qos-policy-group <policy_name>
```

また、使用しているバージョンの ONTAP でサポートされている場合は、最小 QoS を使用してコンテナ化されたワークロードへのスループットを保証することもできます。アダプティブ QoS は SVM レベルのポリシーには対応していません。

コンテナ化されたワークロード専用の IOPS は、さまざまな要素によって異なります。その中には、次のようなものがあります。

- ストレージアレイを使用するその他のワークロード。Kubernetes 環境とは関係なく、ストレージリソースを利用するほかのワークロードがある場合は、それらのワークロードが誤って影響を受けないように注意する必要があります。
- 想定されるワークロードはコンテナで実行されます。IOPS 要件が高いワークロードをコンテナで実行する場合は、 QoS ポリシーの値が低いとエクスペリエンスが低下します。

SVM レベルで割り当てた QoS ポリシーを使用すると、 SVM にプロビジョニングされたすべてのボリュームで同じ IOPS プールが共有されることに注意してください。コンテナ化されたアプリケーションの 1 つまたは少数のに高い IOPS が必要な場合、コンテナ化された他のワークロードに対する Bully になる可能性があります。その場合は、外部の自動化を使用したボリュームごとの QoS ポリシーの割り当てを検討してください。

> ONTAP バージョン 9.8 より前の場合は、 QoS ポリシーグループを SVM * only * に割り当ててください。

Trident の **QoS** ポリシーグループを作成

Quality of Service （ QoS ；サービス品質）は、競合するワークロードによって重要なワークロードのパフォーマンスが低下しないようにします。ONTAP の QoS ポリシーグループには、ボリュームに対する QoS オプションが用意されており、ユーザは 1 つ以上のワークロードに対するスループットの上限を定義できます。QoSの詳細については、を参照してください。 "QoS によるスループットの保証"。
QoS ポリシーグループはバックエンドまたはストレージプールに指定でき、そのプールまたはバックエンドに作成された各ボリュームに適用されます。

ONTAP には、従来型とアダプティブ型の 2 種類の QoS ポリシーグループがあります。従来のポリシーグループは、最大スループット（以降のバージョンでは最小スループット）がフラットに表示されます。アダプティブ QoS では、ワークロードのサイズの変更に合わせてスループットが自動的に調整され、 TB または GB あたりの IOPS が一定に維持されます。これにより、何百何千という数のワークロードを管理する大規模な環境では大きなメリットが得られます。

QoS ポリシーグループを作成するときは、次の点に注意してください。

- バックエンド構成の「金庫」ブロックに「 QOSPolicy 」キーを設定する必要があります。次のバックエンド設定例を参照してください。

```
  ---
version: 1
storageDriverName: ontap-nas
managementLIF: 0.0.0.0
dataLIF: 0.0.0.0
svm: svm0
username: user
password: pass
defaults:
  qosPolicy: standard-pg
storage:
- labels:
    performance: extreme
  defaults:
    adaptiveQosPolicy: extremely-adaptive-pg
- labels:
    performance: premium
  defaults:
    qosPolicy: premium-pg
```

- ボリュームごとにポリシーグループを適用して、各ボリュームがポリシーグループの指定に従ってスループット全体を取得するようにします。共有ポリシーグループはサポートされません。

QoSポリシーグループの詳細については、を参照してください。 "ONTAP 9.8 QoS コマンド"。

ストレージリソースへのアクセスを **Kubernetes** クラスタメンバーに制限する

Trident によって作成される NFS ボリュームと iSCSI LUN へのアクセスを制限することは、 Kubernetes 環境のセキュリティ体制に欠かせない要素です。これにより、 Kubernetes クラスタに属していないホストがボリュームにアクセスしたり、データが予期せず変更されたりすることを防止できます。

ネームスペースは Kubernetes のリソースの論理的な境界であることを理解することが重要です。ただし、同じネームスペース内のリソースは共有可能であることが前提です。重要なのは、ネームスペース間に機能がないことです。つまり、 PVS はグローバルオブジェクトですが、 PVC にバインドされている場合は、同じネームスペース内のポッドからのみアクセス可能です。* 適切な場合は、名前空間を使用して分離することが重要です。 *

Kubernetes 環境でデータセキュリティを使用する場合、ほとんどの組織で最も懸念されるのは、コンテナ内のプロセスがホストにマウントされたストレージにアクセスできることですが、コンテナ用ではないためです。 "ネームスペース" この種の妥協を防ぐように設計されています。ただし、特権コンテナという例外が 1 つあります。

権限付きコンテナは、通常よりもホストレベルの権限で実行されるコンテナです。デフォルトでは拒否されないため、を使用してこの機能を無効にしてください "ポッドセキュリティポリシー"。

Kubernetes と外部ホストの両方からアクセスが必要なボリュームでは、 Trident ではなく管理者が導入した PV で、ストレージを従来の方法で管理する必要があります。これにより、 Kubernetes と外部ホストの両方が切断され、ボリュームを使用していない場合にのみ、ストレージボリュームが破棄されます。また、カスタムエクスポートポリシーを適用して、 Kubernetes クラスタノードおよび Kubernetes クラスタの外部にある

ターゲットサーバからのアクセスを可能にすることもできます。

専用のインフラノード（OpenShiftなど）や、ユーザアプリケーションをスケジュールできない他のノードを導入する場合は、ストレージリソースへのアクセスをさらに制限するために別 々 のエクスポートポリシーを使用する必要があります。これには、これらのインフラノードに導入されているサービス（ OpenShift Metrics サービスや Logging サービスなど）のエクスポートポリシーの作成と、非インフラノードに導入されている標準アプリケーションの作成が含まれます。

専用のエクスポートポリシーを使用します

Kubernetes クラスタ内のノードへのアクセスのみを許可するエクスポートポリシーが各バックエンドに存在することを確認する必要があります。Tridentはエクスポートポリシーを自動的に作成、管理できます。これにより、 Trident はプロビジョニング対象のボリュームへのアクセスを Kubernetes クラスタ内のノードに制限し、ノードの追加や削除を簡易化します。

また、エクスポートポリシーを手動で作成し、各ノードのアクセス要求を処理する 1 つ以上のエクスポートルールを設定することもできます。

- 「 vserver export-policy create 」 ONTAP CLI コマンドを使用して、エクスポートポリシーを作成します。
- 「 vserver export-policy rule create 」 ONTAP CLI コマンドを使用して、エクスポートポリシーにルールを追加します。

これらのコマンドを実行すると、データにアクセスできる Kubernetes ノードを制限できます。

無効にします `showmount` アプリケーション**SVM**用

「 SVM 」機能を使用すると、 NFS クライアントが SVM に照会して使用可能な NFS エクスポートのリストを表示できます。Kubernetes クラスタに導入されたポッドは、データ LIF に対する「 howmount-e 」コマンドを問題に送信し、アクセス権がないマウントも含め、使用可能なマウントのリストを受信できます。これだけではセキュリティ上の妥協ではありませんが、権限のないユーザが NFS エクスポートに接続するのを阻止する可能性のある不要な情報が提供されます。

SVM レベルの ONTAP CLI コマンドを使用して、 SVM の howmount を無効にする必要があります。

```
vserver nfs modify -vserver <svm_name> -showmount disabled
```

SolidFire のベストプラクティス

Trident に SolidFire ストレージを設定するためのベストプラクティスをご確認ください。

SolidFire アカウントを作成します

各 SolidFire アカウントは固有のボリューム所有者で、 Challenge Handshake Authentication Protocol （ CHAP ；チャレンジハンドシェイク認証プロトコル）クレデンシャルのセットを受け取ります。アカウントに割り当てられたボリュームには、アカウント名とその CHAP クレデンシャルを使用してアクセスするか、ボリュームアクセスグループを通じてアクセスできます。アカウントには最大 2 、 000 個のボリュームを関連付けることができますが、 1 つのボリュームが属することのできるアカウントは 1 つだけです。

QoS ポリシーを作成する

標準的なサービス品質設定を作成して保存し、複数のボリュームに適用する場合は、 SolidFire のサービス品質（ QoS ）ポリシーを使用します。

QoS パラメータはボリューム単位で設定できます。QoS を定義する 3 つの設定可能なパラメータである Min IOPS 、 Max IOPS 、 Burst IOPS を設定することで、各ボリュームのパフォーマンスが保証されます。

4KB のブロックサイズの最小 IOPS 、最大 IOPS 、バースト IOPS の値を次に示します。

IOPS パラメータ	定義（ **Definition** ）	最小価値	デフォルト値	最大値（ **4KB** ）
最小 IOPS	ボリュームに対して保証されたレベルのパフォーマンス。	50	50	15000
最大 IOPS	パフォーマンスはこの制限を超えません。	50	15000	200,000
バースト IOPS	短時間のバースト時に許容される最大 IOPS 。	50	15000	200,000

> Max IOPS と Burst IOPS は最大 200 、 000 に設定できますが、実際のボリュームの最大パフォーマンスは、クラスタの使用量とノードごとのパフォーマンスによって制限されます。

ブロックサイズと帯域幅は、 IOPS に直接影響します。ブロックサイズが大きくなると、システムはそのブロックサイズを処理するために必要なレベルまで帯域幅を増やします。帯域幅が増えると、システムが処理可能な IOPS は減少します。を参照してください "SolidFire のサービス品質" QoS およびパフォーマンスの詳細については、を参照してください。

SolidFire 認証

Element では、認証方法として CHAP とボリュームアクセスグループ（ VAG ）の 2 つがサポートされています。CHAP は CHAP プロトコルを使用して、バックエンドへのホストの認証を行います。ボリュームアクセスグループは、プロビジョニングするボリュームへのアクセスを制御します。CHAP はシンプルで拡張性に制限がないため、認証に使用することを推奨します。

> Trident と強化された CSI プロビジョニングツールは、 CHAP 認証の使用をサポートします。VAG は、従来の CSI 以外の動作モードでのみ使用する必要があります。

CHAP 認証（イニシエータが対象のボリュームユーザであることの確認）は、アカウントベースのアクセス制御でのみサポートされます。認証に CHAP を使用している場合は、単方向 CHAP と双方向 CHAP の 2 つのオプションがあります。単方向 CHAP は、 SolidFire アカウント名とイニシエータシークレットを使用してボリュームアクセスを認証します。双方向の CHAP オプションを使用すると、ボリュームがアカウント名とイニシエータシークレットを使用してホストを認証し、ホストがアカウント名とターゲットシークレットを使用してボリュームを認証するため、ボリュームを最も安全に認証できます。

ただし、 CHAP を有効にできず VAG が必要な場合は、アクセスグループを作成し、ホストのイニシエータとボリュームをアクセスグループに追加します。アクセスグループに追加した各 IQN は、 CHAP 認証の有無に

関係なく、グループ内の各ボリュームにアクセスできます。iSCSI イニシエータが CHAP 認証を使用するように設定されている場合は、アカウントベースのアクセス制御が使用されます。iSCSI イニシエータが CHAP 認証を使用するように設定されていない場合は、ボリュームアクセスグループのアクセス制御が使用されます。

詳細情報の入手方法

ベストプラクティスのドキュメントの一部を以下に示します。を検索します "NetApp ライブラリ" 最新バージョンの場合。

- ONTAP *
- "『 NFS Best Practice and Implementation Guide 』を参照してください"
- "『 SAN アドミニストレーションガイド』" （ iSCSI の場合）
- "RHEL 向けの iSCSI のクイック構成"
- Element ソフトウェア *
- "SolidFire for Linux を設定しています"
- NetApp HCI *
- "NetApp HCI 導入の前提条件"
- "NetApp Deployment Engine にアクセスします"
- アプリケーションのベストプラクティス情報 *
- "ONTAP での MySQL に関するベストプラクティスです"
- "SolidFire での MySQL に関するベストプラクティスです"
- "NetApp SolidFire および Cassandra"
- "SolidFire での Oracle のベストプラクティス"
- "SolidFire での PostgreSQL のベストプラクティスです"

すべてのアプリケーションに具体的なガイドラインがあるわけではありません。そのためには、ネットアップのチームと協力し、を使用することが重要です "NetApp ライブラリ" 最新のドキュメントを検索できます。

Astra Trident を統合

Astra Tridentを統合するには、設計とアーキテクチャに関する次の要素を統合する必要があります。ドライバの選択と導入、ストレージクラスの設計、仮想プールの設計、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、フル ONTAP FlexVol です。
- 「 ONTAP-NAS-エコノミー 」：プロビジョニングされた各 PV は qtree で、 FlexVol あたりの qtree 数を設定できます（デフォルトは 200 ）。
- 「 ONTAP-NAS-flexgroup 」：フル ONTAP FlexGroup としてプロビジョニングされた各 PV と、 SVM に割り当てられたすべてのアグリゲートが使用されます。
- 「 ONTAP - SAN 」：プロビジョニングされた各 PV は、固有の FlexVol 内の LUN です。
- 「 ONTAP-SAN-エコノミー 」：各 PV がプロビジョニングされた LUN で、 FlexVol あたりの LUN 数を設定できます（デフォルトは 100 ）。

3 つの NAS ドライバの間で選択すると、アプリケーションで使用できる機能にいくつかの影響があります。

次の表では、 Astra Trident からすべての機能が提供されるわけではありません。一部の機能は、プロビジョニング後にストレージ管理者が適用する必要があります。上付き文字の脚注は、機能やドライバごとに機能を区別します。

ONTAP NAS ドライバ	**Snapshot**	クローン	動的なエクスポートポリシー	マルチアタッチ	**QoS**	サイズ変更	レプリケーション
「 ONTAP - NAS 」	はい。	はい。	○脚注：5[]	はい。	○脚注：1[]	はい。	○脚注：1[]
「 ONTAP - NAS - エコノミー」	○脚注：3[]	○脚注：3[]	○脚注：5[]	はい。	○脚注：3[]	はい。	○脚注：3[]
「 ONTAP-NAS-flexgroup 」	○脚注：1[]	いいえ	○脚注：5[]	はい。	○脚注：1[]	はい。	○脚注：1[]

Astra Trident は、 ONTAP 向けに 2 つの SAN ドライバを提供しています。このドライバの機能は次のとおりです。

ONTAP SAN ドライバ	**Snapshot**	クローン	マルチアタッチ	双方向 **CHAP**	**QoS**	サイズ変更	レプリケーション
「 ontap - san 」	はい。	はい。	○脚注：4[]	はい。	○脚注：1[]	はい。	○脚注：1[]
「 ONTAP - SAN - エコノミー」	はい。	はい。	○脚注：4[]	はい。	○脚注：3[]	はい。	○脚注：3[]

> 上記の表の脚注：
>
> Yes [1]：Astra Tridentで管理されない
>
> Yesfootnote: 2[]：Astra Tridentが管理しますが、PV Granularは管理しません
>
> Yesfootnote: 3[]：Astra Tridentで管理されず、PV Granularでは管理されない
>
> Yes [4]:raw-blockボリュームでサポート
>
> Yesfootnote: 5[]：Astra Tridentによるサポート

PV に細分化されていない機能は FlexVol 全体に適用され、 PVS （共有 FlexVol 内の qtree または LUN ）にはすべて共通のスケジュールが適用されます。

上の表に示すように 'ONTAP-NAS' と「 ONTAP-NAS-エコノミー 」の機能の多くは同じですしかし 'ONTAP-NAS-エコノミー のドライバは ' スケジュールを PV 単位で制御する機能を制限するため ' これは特に災害復旧やバックアップ計画に影響を与える可能性がありますONTAP ストレージ上で PVC クローン機能を活用したい開発チームの場合 ' これは 'ONTAP-NAS' ONTAP -SAN' または ONTAP -SAN' のいずれかのドライバを使用する場合にのみ可能です

> 「 olidfire -san 」ドライバは PVC のクローン作成にも対応しています。

Cloud Volumes ONTAP バックエンドドライバ

Cloud Volumes ONTAP は、ファイル共有や NAS および SAN プロトコル（ NFS 、 SMB / CIFS 、 iSCSI ）を提供するブロックレベルストレージなど、さまざまなユースケースでデータ制御とエンタープライズクラスのストレージ機能を提供します。Cloud Volume ONTAP の互換性のあるドライバは、「 ONTAP-NAS' 」、「 ONTAP-NAS-エコノミー 」、「 ONTAP-SAN' 」、「 ONTAP-SAN-エコノミー 」です。Cloud Volume ONTAP for Azure と Cloud Volume ONTAP for GCP に該当します。

ONTAP バックエンドドライバ用の**Amazon FSX**

Amazon FSx for NetApp ONTAPを使用すると、AWSにデータを格納する際のシンプルさ、即応性、セキュリティ、拡張性を活用しながら、使い慣れたNetAppの機能、パフォーマンス、管理機能を活用できます。FSx for ONTAPは、多くのONTAPファイルシステム機能と管理APIをサポートしています。Cloud Volume ONTAP の互換性のあるドライバはです `ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `ontap-san` および `ontap-san-economy`。

NetApp HCI / SolidFireバックエンドドライバ

NetApp HCI / SolidFire プラットフォームで使用される「 olidfire -SAN」 ドライバは、管理者が QoS 制限に基づいて Trident の Element バックエンドを構成するのに役立ちます。Trident によってプロビジョニングされるボリュームに特定の QoS 制限を設定するためにバックエンドを設計する場合は、バックエンドファイルの「 type 」パラメータを使用します。管理者は 'limitVolumeSize' パラメータを使用して ' ストレージ上に作成できるボリューム・サイズを制限することもできます現在、ボリュームのサイズ変更やボリュームのレプリケーションなどの Element ストレージ機能は、 'olidfire-san' ドライバではサポートされていません。これらの処理は、 Element ソフトウェアの Web UI から手動で実行する必要があります。

SolidFire ドライバ	Snapshot	クローン	マルチアタッチ	CHAP	QoS	サイズ変更	レプリケーション
「 olidfire -san 」	はい。	はい。	○脚注： 2[]	はい。	はい。	はい。	○脚注： 1[]

> 脚注：はい脚注： 1[] ： 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、「 azure-NetApp-files 」ドライバを使用してを管理します "Azure NetApp Files の特長" サービス

このドライバの詳細と設定方法については、を参照してください "Azure NetApp Files 向けの Trident バックエンド構成"。

Azure NetApp Files ドライバ	Snapshot	クローン	マルチアタッチ	QoS	を展開します	レプリケーション
「 azure-NetApp-files 」と入力します	はい。	はい。	はい。	はい。	はい。	○脚注： 1[]

> 脚注：はい脚注： 1[] ： Astra Trident で管理されていません

Google Cloudバックエンドドライバ上の**Cloud Volumes Service**

Astra Tridentがを使用 `gcp-cvs` Google CloudのCloud Volumes Service にリンクするドライバ。

。 `gcp-cvs` ドライバは仮想プールを使用してバックエンドを抽象化し、Astra Tridentでボリュームの配置を判断できるようにします。管理者が、で仮想プールを定義します `backend.json` ファイル。ストレージクラスには、ラベルで仮想プールを識別するセレクタが使用されます。

- バックエンドに仮想プールが定義されている場合、Astra Tridentは、その仮想プールが制限されているGoogle Cloudストレージプール内にボリュームを作成しようとします。
- バックエンドに仮想プールが定義されていない場合、Astra Tridentは、リージョン内の使用可能なストレージプールからGoogle Cloudストレージプールを選択します。

Astra TridentでGoogle Cloudバックエンドを設定するには、と指定する必要があります `projectNumber`、`apiRegion`および `apiKey` バックエンドファイル内。プロジェクト番号はGoogle Cloudコンソールで確認できます。APIキーは、Google CloudでCloud Volumes Service のAPIアクセスを設定するときに作成したサービスアカウントの秘密鍵ファイルから取得されます。

Google CloudでのCloud Volumes Serviceのサービスタイプとサービスレベルの詳細については、を参照してください。 "CVS for GCPのAstra Tridentサポートについてご確認ください"。

Cloud Volumes Service for Google Cloudドライバ	Snapshot	クローン	マルチアタッチ	QoS	を展開します	レプリケーション
「 gcp-cvs 」	はい。	はい。	はい。	はい。	はい。	CVS -パフォーマンスサービスタイプでのみ利用できます。

レプリケーションに関する注意事項

- レプリケーションはAstra Tridentで管理されていません。
- クローンは、ソースボリュームと同じストレージプールに作成されます。

ストレージクラスの設計

Kubernetes ストレージクラスオブジェクトを作成するには、個々のストレージクラスを設定して適用する必要があります。このセクションでは、アプリケーション用のストレージクラスの設計方法について説明します。

特定のバックエンド使用率

フィルタリングは、特定のストレージクラスオブジェクト内で使用でき、そのストレージクラスで使用するストレージプールまたはプールのセットを決定します。ストレージクラスでは'`toragePools’additionalStoragePools’excludeStoragePools" の 3 セットのフィルタを設定できます

'toragePools' パラメータは ' 指定した属性に一致するプールのセットにストレージを制限するのに役立ちます「 additionalStoragePools 」パラメータは、 Astra Trident がプロビジョニングに使用する一連のプールと、属性と「 toragePools 」パラメータで選択した一連のプールを拡張するために使用されます。どちらか一方のパラメータを単独で使用することも、両方を使用して、適切なストレージプールセットが選択されていることを確認することもできます。

excludeStoragePools' パラメータを使用して ' 属性に一致するプールの一覧を除外します

QoSポリシーをエミュレートします

ストレージクラスを設計して Quality of Service ポリシーをエミュレートする場合は ' 「メディア」属性を「 hdd 」または「 sd 」として ' ストレージクラスを作成しますストレージクラスで言及されている「メディア」属性に基づいて、 Trident は「 hdd 」アグリゲートまたは「 sd 」アグリゲートにメディア属性と一致させる適切なバックエンドを選択し、ボリュームのプロビジョニングを特定のアグリゲートに誘導します。したがって、「メディア」属性が「 SD 」に設定されているストレージクラス Premium を作成して、プレミアム QoS ポリシーに分類できます。メディア属性を「 hdd 」に設定し、標準の QoS ポリシーとして分類できる、別のストレージクラス標準を作成できます。また、ストレージクラスの「 IOPS 」属性を使用して、 QoS ポリシーとして定義できる Element アプライアンスにプロビジョニングをリダイレクトすることもできます。

特定の機能に基づいてバックエンドを利用する

ストレージクラスは、シンプロビジョニングとシックプロビジョニング、 Snapshot 、クローン、暗号化などの機能が有効になっている特定のバックエンドでボリュームを直接プロビジョニングするように設計できます。使用するストレージを指定するには、必要な機能を有効にしてバックエンドに適したストレージクラスを作成します。

仮想プール

仮想プールはすべての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。設定 `servicelevel` 適切なパフォーマンスレベルを考慮し、各ラベルの下にその他の必要な側面を追加します。次に、異なる仮想プールにマッピングするさまざまなKubernetesストレージクラスを作成します。を使用する `parameters.selector` 各StorageClassは、ボリュームのホストに使用できる仮想プールを呼び出します。

特定の一連の側面を割り当てます

特定の側面を持つ複数の仮想プールは、単一のストレージバックエンドから設計できます。そのためには、バックエンドに複数のラベルを設定し、各ラベルに必要な側面を設定します。を使用して、さまざまなKubernetesストレージクラスを作成します `parameters.selector` 異なる仮想プールにマッピングされるフィールド。バックエンドでプロビジョニングされるボリュームには、選択した仮想プールに定義された設定が適用されます。

ストレージプロビジョニングに影響する **PVC** 特性

要求されたストレージクラスを超えたパラメータの中には、PVCを作成する際にAstra Tridentプロビジョニングの判断プロセスに影響するものがあります。

アクセスモード

PVC 経由でストレージを要求する場合、必須フィールドの 1 つがアクセスモードです。必要なモードは、ストレージ要求をホストするために選択されたバックエンドに影響を与える可能性があります。

Astra Trident は、次のマトリックスで指定されたアクセス方法で使用されているストレージプロトコルと一致するかどうかを試みます。これは、基盤となるストレージプラットフォームに依存しません。

	ReadWriteOnce コマンドを使用します	**ReadOnlyMany**	**ReadWriteMany**
iSCSI	はい。	はい。	○（ Raw ブロック）
NFS	はい。	はい。	はい。

NFS バックエンドが設定されていない Trident 環境に送信された ReadWriteMany PVC が要求された場合、ボリュームはプロビジョニングされません。このため、リクエスタは、アプリケーションに適したアクセスモードを使用する必要があります。

ボリューム操作

永続ボリュームの変更

永続ボリュームとは、 Kubernetes で変更不可のオブジェクトを 2 つだけ除いてです。再利用ポリシーとサイズは、いったん作成されると変更できます。ただし、これにより、ボリュームの一部の要素が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。これを実現するには、 PersistentVolumeClaim を作成し、ボリュームを作成する必要のある Snapshot として「ソース」を指定します。Astra Trident がこの PVC を処理するには、 Snapshot にデータが存在するボリュームを作成します。この機能を使用すると、複数のリージョン間でデータを複製したり、テスト環境を作成したり、破損した本番ボリューム全体を交換したり、特定のファイルとディレクトリを取得して別の接続ボリュームに転送したりできます。

クラスタ内でボリュームを移動します

ストレージ管理者は、 ONTAP クラスタ内のアグリゲート間およびコントローラ間で、ストレージ利用者への無停止でボリュームを移動できます。この処理は、デスティネーションアグリゲートが Trident が使用している SVM からアクセス可能なアグリゲートであるかぎり、 Astra Trident または Kubernetes クラスタには影響しません。この点が重要なのは、アグリゲートが SVM に新たに追加された場合、 Astra Trident に再追加してバックエンドを更新する必要があることです。これにより、 Astra Trident が SVM のインベントリを再作成し、新しいアグリゲートが認識されるようになります。

ただし、バックエンド間でのボリュームの移動は Astra Trident では自動ではサポートされていません。これには、同じクラスタ内の SVM 間、クラスタ間、または別のストレージプラットフォーム上の SVM 間が含まれます（たとえストレージシステムが Trident から Astra に接続されている場合でも）。

ボリュームが別の場所にコピーされた場合、ボリュームインポート機能を使用して現在のボリュームを Astra Trident にインポートできます。

ボリュームを展開します

Astra Trident は、 NFS と iSCSI PVS のサイズ変更をサポートしています。これにより、ユーザは Kubernetes レイヤを介してボリュームのサイズを直接変更できます。ボリュームを拡張できるのは、 ONTAP 、 SolidFire / NetApp HCI 、 Cloud Volumes Service バックエンドなど、主要なすべてのネットアップストレージプラットフォームです。後で拡張できるようにするには ' ボリュームに関連づけられたストレージ・クラスで 'allowVolumeExpansion を true に設定します永続ボリュームのサイズを変更する必要がある場合は、 Persistent Volume Claim の「 PEC.resources.request.storage 」注釈を必要なボリュームサイズに編集します。Trident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」 、「 ONTAP-NAS-flexgroup 」、「 solidfire-san-」 、「 azure-netapp-files 」、「 gcp-cvs` ドライバ」でサポートされています。この機能は、既存のアプリケーションを Kubernetes に移植する場合や、ディザスタリカバリシナリオで使用する場合に便利です。

ONTAP ドライバと 'olidfire-san`drivers を使用する場合は 'tridentctl import volume <backend-name><volume-name><f/path/pvc.yaml コマンドを使用して 'Astra Trident で管理する既存のボリュームを Kubernetes にインポートしますimport volume コマンドで使用した PVC YAML または JSON ファイルは、 Astra Trident をプロビジョニングツールとして識別するストレージクラスを指定します。NetApp HCI / SolidFire バックエンドを使用する場合は、ボリューム名が一意であることを確認してください。ボリューム名が重複している場合は、ボリュームインポート機能で区別できるように、ボリュームを一意の名前にクローニングします。

「 azure-NetApp-file' 」または「 gcp-cvs` ドライバが使用されている場合は、「 tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml 」コマンドを使用して、 Astra Trident で管理される Kubernetes にボリュームをインポートします。これにより、ボリューム参照が一意になります。

上記のコマンドを実行すると、 Astra Trident がバックエンド上にボリュームを検出し、サイズを確認します。設定されたPVCのボリュームサイズを自動的に追加（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、に記載されています "netapp.io のコマンドです" を参照してください "ブログ"。

ロギングサービス

他の OpenShift サービスと同様に、ログ記録サービスは、 Ansible と、インベントリファイル（別名）で提供される構成パラメータを使用して導入されますホスト。プレイブックに含まれています。ここでは、 OpenShift の初期インストール時にロギングを導入し、 OpenShift のインストール後にロギングを導入するという、 2 つのインストール方法について説明します。

> Red Hat OpenShift バージョン 3.9 以降、データ破損に関する懸念があるため、記録サービスに NFS を使用しないことを公式のドキュメントで推奨しています。これは、 Red Hat 製品のテストに基づいています。ONTAP NFSサーバにはこのような問題がないため、ロギング環境を簡単にバックアップできます。ロギングサービスには最終的にどちらかのプロトコルを選択する必要がありますが、両方のプロトコルがネットアッププラットフォームを使用する場合に適していることと、 NFS を使用する理由がないことを確認してください。

ロギング・サービスで NFS を使用する場合は、インストーラが失敗しないように、 Ansible 変数「 OpenShift 」の「 OpenShift 」 enable_unsupported _configurations 」を「 true 」に設定する必要があります。

はじめに

ロギングサービスは、必要に応じて、両方のアプリケーションに導入することも、 OpenShift クラスタ自体のコア動作に導入することもできます。オペレーション・ログを配置する場合 ' 変数 OpenShift の logging_use_ops を true として指定すると ' サービスの 2 つのインスタンスが作成されます操作のロギングインスタンスを制御する変数には「 ops 」が含まれ、アプリケーションのインスタンスには含まれません。

基盤となるサービスで正しいストレージが使用されるようにするには、導入方法に応じてAnsible変数を設定することが重要です。それぞれの導入方法のオプションを見てみましょう。

> 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。その他のオプションは、で確認できます "Red Hat OpenShift のロギングに関するドキュメント" 導入環境に応じて、確認、設定、使用する必要があります。

次の表の変数では、入力した詳細を使用してロギングサービスの PV と PVC を作成する Ansible プレイブックが作成されます。この方法は、 OpenShift インストール後にコンポーネントインストールプレイブックを使用するよりもはるかに柔軟性に劣るが、既存のボリュームがある場合はオプションとなります。

変数（ **Variable** ）	詳細
「 OpenShift 」ロギング・ストレージ・タイプ	インストーラがログサービス用の NFS PV を作成するように 'NFS' に設定します
「 OpenShift 」ロギング・ストレージ・ホスト	NFS ホストのホスト名または IP アドレス。仮想マシンのデータ LIF に設定してください。
「 OpenShift 」ロギング・ストレージ・NFS_DIRECT'	NFS エクスポートのマウントパス。たとえば、ボリュームが「 /OpenShift _logging 」としてジャンクションされている場合、この変数にそのパスを使用します。
「 OpenShift 」ロギング・ストレージ・ボリューム名	作成する PV の名前（「 pv_ose_logs 」など）。
「 OpenShift 」ロギング・ストレージ・ボリューム・サイズ	NFS エクスポートのサイズ（例： 100Gi ）

OpenShift クラスタがすでに実行中で、そのため Trident を導入して設定した場合、インストーラは動的プロビジョニングを使用してボリュームを作成できます。次の変数を設定する必要があります。

変数（ **Variable** ）	詳細
'OpenShift の logging_es_vpc_dynamic	動的にプロビジョニングされたボリュームを使用する場合は true に設定します。
「 OpenShift logging _es_vpc_storage_class_name 」	PVC で使用されるストレージクラスの名前。
「 OpenShift logging _es_vpc_size 」を参照してください	PVC で要求されたボリュームのサイズ。
「 OpenShift logging _es_vpc_prefix 」を参照してください	ロギングサービスで使用される PVC のプレフィックス。
'OpenShift の logging_es_ops_pvc_dynamic	動的にプロビジョニングされたボリュームを ops ロギングインスタンスに使用するには、「 true 」に設定します。
「 OpenShift logging _es_ops_pvc_storage_class_name 」を参照してください	処理ロギングインスタンスのストレージクラスの名前。
'OpenShift logging _es_ops_pvc_size	処理インスタンスのボリューム要求のサイズ。
「 OpenShift logging _es_ops_pvc_prefix 」を参照してください	ops インスタンス PVC のプレフィックス。

ロギングスタックを導入します

初期の OpenShift インストールプロセスの一部としてロギングを導入する場合、標準の導入プロセスに従うだけで済みます。Ansible は、必要なサービスと OpenShift オブジェクトを構成および導入して、 Ansible が完了したらすぐにサービスを利用できるようにします。

ただし、最初のインストール後に導入する場合は、コンポーネントプレイブックを Ansible で使用する必要があります。このプロセスは、 OpenShift のバージョンが異なるためわずかに変更される場合があるので、必ず読んで従うようにしてください "Red Hat OpenShift Container Platform 3.11 のドキュメント" 使用しているバージョンに対応した

指標サービス

この指標サービスは、 OpenShift クラスタのステータス、リソース利用率、可用性に関する重要な情報を管理者に提供します。ポッドの自動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（ **Variable** ）	詳細
「 OpenShift _ metrics _ storage _kind 」	インストーラがログサービス用の NFS PV を作成するように 'NFS' に設定します
「 OpenShift _ metrics _storage_host 」というようになります	NFS ホストのホスト名または IP アドレス。これは SVM のデータ LIF に設定されている必要があります。
「 OpenShift _ metrics _storage_nfs_directory 」というエラーが表示されます	NFS エクスポートのマウントパス。たとえば、ボリュームが「 /OpenShift メトリック」としてジャンクションされている場合は、この変数にそのパスを使用します。
「 OpenShift _ metrics _storage_volume_name 」という形式で指定します	作成する PV の名前（「 pv_ose_metrics 」など）。
「 OpenShift _ metrics _storage_volume_size 」というようになります	NFS エクスポートのサイズ（例： 100Gi ）

OpenShift クラスタがすでに実行中で、そのため Trident を導入して設定した場合、インストーラは動的プロビジョニングを使用してボリュームを作成できます。次の変数を設定する必要があります。

変数（ **Variable** ）	詳細
「 OpenShift _ metrics _ cassandra_vpc_prefix 」という形式で指定します	メトリック PVC に使用するプレフィックス。
「 OpenShift _ metrics _ cassandra_vp_size' 」のようになります	要求するボリュームのサイズ。
「 OpenShift _ metrics _ cassandra_storage_type 」のようになります	指標に使用するストレージのタイプ。適切なストレージクラスを使用して PVC を作成するには、 Ansible に対してこれを dynamic に設定する必要があります。
「 OpenShift _ metrics _cassanda_pvc_storage_class_name 」という形式で指定します	使用するストレージクラスの名前。

指標サービスを導入する

ホスト / インベントリファイルに適切な Ansible 変数を定義して、 Ansible でサービスを導入します。OpenShift インストール時に導入する場合は、 PV が自動的に作成されて使用されます。コンポーネントプレイブックを使用して導入する場合は、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。必ず見直しを行ってください "RedHat OpenShift 導入ガイド" をバージョンに合わせて設定し、環境に合わせて設定します。

データ保護とディザスタリカバリ

Astra TridentとAstra Tridentを使用して作成されたボリュームの保護とリカバリのオプションについて説明します。永続性に関する要件があるアプリケーションごとに、データ保護とリカバリの戦略を用意しておく必要があります。

Astra Tridentのレプリケーションとリカバリ

災害発生時にAstra Tridentをリストアするバックアップを作成できます。

Astra Tridentのレプリケーション

Astra Tridentは、Kubernetes CRDを使用して独自の状態の格納と管理を行い、KubernetesクラスタetcdをK使用してメタデータを格納します。

手順

1. を使用してKubernetesクラスタetcdをバックアップします "Kubernetes：etcdクラスタのバックアップ"。
2. バックアップアーティファクトをFlexVolに配置します。

> FlexVolが配置されているSVMを別のSVMへのSnapMirror関係で保護することを推奨します。

Astra Tridentのリカバリ

Kubernetes CRDとKubernetesクラスタetcd Snapshotを使用して、Astra Tridentをリカバリできます。

手順

1. デスティネーションSVMから、Kubernetes etcdデータファイルと証明書が格納されているボリュームを、マスターノードとしてセットアップするホストにマウントします。
2. Kubernetesクラスタに関連する必要な証明書をすべてのにコピーします `/etc/kubernetes/pki` および の下のetcdメンバーファイル `/var/lib/etcd`。
3. を使用して、etcdバックアップからKubernetesクラスタをリストアします "Kubernetes：etcdクラスタをリストアします"。
4. を実行します `kubectl get crd` Tridentのカスタムリソースがすべて稼働していることを確認し、Tridentオブジェクトを読み出してすべてのデータが利用可能であることを確認します。

SVMのレプリケーションとリカバリ

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できます "ONTAP SnapMirrorの略" SVMをレプリケートするため。

災害が発生した場合は、 SnapMirror デスティネーション SVM をアクティブ化してデータの提供を開始できます。システムがリストアされたら、プライマリに戻すことができます。

このタスクについて

SnapMirror SVMレプリケーション機能を使用する場合は、次の点を考慮してください。

- SVM-DRを有効にしたSVMごとに、個別のバックエンドを作成する必要があります。
- SVM-DRをサポートするバックエンドにレプリケーション不要のボリュームをプロビジョニングしないように、必要な場合にのみレプリケートされたバックエンドを選択するようにストレージクラスを設定します。
- アプリケーション管理者は、レプリケーションに伴う追加コストと複雑さを理解し、このプロセスを開始する前にリカバリプランを慎重に検討する必要があります。

SVMレプリケーション

を使用できます "ONTAP：SnapMirrorレプリケーション" をクリックしてSVMレプリケーション関係を作成します。

SnapMirrorでは、レプリケートする対象を制御するオプションを設定できます。プリフォーム時に選択したオプションを知っておく必要があります Astra Tridentを使用したSVMのリカバリ。

- "-identity-preserve trueを指定します" SVMの設定全体をレプリケートします。
- "-discard-configs network" LIFと関連ネットワークの設定を除外します。
- "-identity-preserve false" ボリュームとセキュリティ設定のみをレプリケートします。

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 `-identity-preserve false` または `-discard-config network` SVMレプリケーションを設定する場合は、を更新します `managementLIF` および `dataLIF` をTridentバックエンド定義ファイルに追加します。
3. 確認します `storagePrefix` は、Tridentバックエンド定義ファイルに含まれています。このパラメータは変更できません。省略しています `storagePrefix` バックエンドの更新が失敗するように原因します。
4. 次のコマンドを使用して、必要なすべてのバックエンドを更新して新しいデスティネーションSVM名を反映します。

```
./tridentctl update backend <backend-name> -f <backend-json-file> -n
<namespace>
```

5. を指定した場合 `-identity-preserve false` または `discard-config network`、すべてのアプリケーションポッドをバウンスする必要があります。

> 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からデータの提供が開始されます。

ボリュームのレプリケーションとリカバリ

Astra TridentではSnapMirrorレプリケーション関係を設定できませんが、ストレージ管理者はを使用できます "ONTAP SnapMirrorのレプリケーションとリカバリ" Astra Tridentで作成されたボリュームをレプリケート

リカバリしたボリュームは、次のコマンドを使用してAstra Tridentにインポートできます： "tridentctlボリュームインポート"。

> ではインポートはサポートされていません `ontap-nas-economy`、 `ontap-san-economy``または `ontap-flexgroup-economy` ドライバ。

Snapshotデータの保護

次のコマンドを使用してデータを保護およびリストアできます。

- 永続ボリューム（PV）のKubernetesボリュームSnapshotを作成するための外部のSnapshotコントローラとCRD。

  "ボリューム Snapshot"

- ONTAP Snapshot：ボリュームの内容全体のリストア、または個 々 のファイルまたはLUNのリカバリに使用します。

  "ONTAPスナップショット"

Astra Control Centerアプリケーションのレプリケーション

Astra Controlを使用すると、SnapMirrorの非同期レプリケーション機能を使用して、データやアプリケーションの変更をクラスタ間でレプリケートできます。

"Astra Control：SnapMirrorテクノロジを使用してアプリケーションをリモートシステムにレプリケート"

セキュリティ

セキュリティ

ここに記載された推奨事項を参考に、Astra Tridentのインストールを安全に行ってください。

Astra Trident を独自のネームスペースで実行

アプリケーション、アプリケーション管理者、ユーザ、および管理アプリケーションが Astra Trident オブジェクト定義またはポッドにアクセスしないようにして、信頼性の高いストレージを確保し、悪意のあるアクティビティをブロックすることが重要です。

他のアプリケーションやユーザを Astra Trident から分離するには、 Astra Trident を必ず独自の Kubernetes ネームスペース（ trident` ）にインストールしてください。Astra Trident を独自の名前空間に配置することで、 Kubernetes 管理担当者のみが Astra Trident ポッドにアクセスでき、名前空間 CRD オブジェクトに格納されたアーティファクト（バックエンドや CHAP シークレット（該当する場合）にアクセスできるようになります。Astra Trident ネームスペースへのアクセスを管理者のみに許可し、 tridentctl アプリケーションへのアクセスを許可する必要があります。

ONTAP SAN バックエンドで **CHAP** 認証を使用します

Astra Trident は、 ONTAP SAN ワークロード向けの CHAP ベースの認証をサポートしています（「 ONTAP-SAN' 」ドライバと「 ONTAP-SAN-エコノミー 」ドライバを使用）。ネットアップでは、ホストとストレージバックエンドの間の認証に、双方向 CHAP と Astra Trident を使用することを推奨しています。

SANストレージドライバを使用するONTAP バックエンドの場合、Astra Tridentは双方向CHAPを設定し、を使用してCHAPユーザ名とシークレットを管理できます `tridentctl`。
を参照してください "" ONTAP バックエンドで Trident が CHAP を構成する方法をご確認ください。

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。Astra Tridentをインストールすると、CHAPシークレットが管理されて `tridentvolume` 対応するPVのCRオブジェクト。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 Volume Encryption の設定の概要"。

- NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームが NAEに対応します。
- NAEがバックエンドで有効になっていない場合、バックエンド構成でNVE暗号化フラグを「false」に設定しないかぎり、Astra Tridentでプロビジョニングされたすべてのボリュームが NVE対応になります。

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「true」に設定すると、NAE暗号化をオーバーライドし、ボリューム単位で特定の暗号化キーを使用できます。
- NAEが有効なバックエンドでNVE暗号化フラグを「false」に設定すると、NAEが有効なボリュームが作成されます。NVE暗号化フラグを「false」に設定してNAE暗号化を無効にすることはできません。

- 明示的にNVE暗号化フラグを「true」に設定すると、Astra TridentでNVEボリュームを手動で作成できます。

バックエンド構成オプションの詳細については、以下を参照してください。

- "ONTAP のSAN構成オプション"
- "ONTAP NASの構成オプション"

Linux Unified Key Setup（LUKS；統合キーセットアップ）

Linux Unified Key Setup（LUKS；ユニファイドキーセットアップ）を有効にして、Astra Trident上のONTAP SANおよびONTAP SANエコノミーボリュームを暗号化できます。Astra Tridentは、LUKS暗号化ボリュームのパスフレーズローテーションとボリューム拡張をサポートしています。

Astra Tridentでは、で推奨されるとおり、LUKSによって暗号化されたボリュームがAES-XTS -原64定型とモードを使用します "NIST"。

作業を開始する前に

- ワーカーノードにはcryptsetup 2.1以上（3.0よりも下位）がインストールされている必要があります。詳細については、を参照してください "Gitlab: cryptsetup"。
- パフォーマンス上の理由から、ワーカーノードでAdvanced Encryption Standard New Instructions（AES-NI）をサポートすることを推奨します。AES-NIサポートを確認するには、次のコマンドを実行します。

```
grep "aes" /proc/cpuinfo
```

何も返されない場合、お使いのプロセッサはAES-NIをサポートしていません。AES-NIの詳細については、以下を参照してください。 "Intel：Advanced Encryption Standard Instructions（AES-NI）"。

LUKS暗号化を有効にします

ONTAP SANおよびONTAP SANエコノミーボリュームでは、Linux Unified Key Setup（LUKS；Linux統合キーセットアップ）を使用して、ボリューム単位のホスト側暗号化を有効にできます。

手順

1. バックエンド構成でLUKS暗号化属性を定義します。ONTAP SANのバックエンド構成オプションの詳細については、を参照してください "ONTAP のSAN構成オプション"。

```
"storage": [
    {
        "labels":{"luks": "true"},
        "zone":"us_east_1a",
        "defaults": {
            "luksEncryption": "true"
        }
    },
    {
        "labels":{"luks": "false"},
        "zone":"us_east_1a",
        "defaults": {
            "luksEncryption": "false"
        }
    },
]
```

2. 使用 `parameters.selector` LUKS暗号化を使用してストレージプールを定義する方法。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: 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

LUKSで暗号化されたボリュームは、ONTAP の重複排除と圧縮を利用できません。

LUKSボリュームをインポートするためのバックエンド構成

LUKSボリュームをインポートするには、を設定する必要があります `luksEncryption` 終了：(`true` バックエンドにあります。。 `luksEncryption` optionを指定すると、ボリュームがLUKS準拠かどうかがAstra Tridentに通知されます (`true`）またはLUKS準拠ではありません (`false`）をクリックします。

```
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: trident_svm
username: admin
password: password
defaults:
  luksEncryption: 'true'
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'
```

LUKSパスフレーズをローテーションします

LUKSのパスフレーズをローテーションしてローテーションを確認できます。

> パスフレーズは、ボリューム、Snapshot、シークレットで参照されなくなることを確認するまで忘れないでください。参照されているパスフレーズが失われた場合、ボリュームをマウントできず、データが暗号化されたままアクセスできなくなることがあります。

このタスクについて

LUKSパスフレーズのローテーションは、ボリュームをマウントするポッドが、新しいLUKSパスフレーズの指定後に作成されたときに行われます。新しいポッドが作成されると、Astra TridentはボリュームのLUKSパスフレーズをシークレット内のアクティブなパスフレーズと比較します。

- ボリュームのパスフレーズがシークレットでアクティブなパスフレーズと一致しない場合、ローテーションが実行されます。
- ボリュームのパスフレーズがシークレットのアクティブなパスフレーズと一致する場合は、を参照してください `previous-luks-passphrase` パラメータは無視されます。

手順

1. を追加します `node-publish-secret-name` および `node-publish-secret-namespace` StorageClassパラメータ。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-san
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/backendType: "ontap-san"
  csi.storage.k8s.io/node-stage-secret-name: luks
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-publish-secret-name: luks
  csi.storage.k8s.io/node-publish-secret-namespace: ${pvc.namespace}
```

2. ボリュームまたはSnapshotの既存のパスフレーズを特定します。

   ボリューム

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["A"]
```

   スナップショット

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["A"]
```

3. ボリュームのLUKSシークレットを更新して、新しいパスフレーズと前のパスフレーズを指定します。確認します `previous-luke-passphrase-name` および `previous-luks-passphrase` 前のパスフレーズと同じにします。

```
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: B
  luks-passphrase: secretB
  previous-luks-passphrase-name: A
  previous-luks-passphrase: secretA
```

4. ボリュームをマウントする新しいポッドを作成します。これはローテーションを開始するために必要です。

5. パスフレーズがローテーションされたことを確認します。

ボリューム

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["B"]
```

スナップショット

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["B"]
```

結果

パスフレーズは、ボリュームとSnapshotに新しいパスフレーズのみが返されたときにローテーションされました。

> たとえば、2つのパスフレーズが返された場合などです `luksPassphraseNames: ["B", "A"]`回転が不完全です。回転を完了するために、新しいポッドをトリガできます。

ボリュームの拡張を有効にします

LUKS暗号化ボリューム上でボリューム拡張を有効にできます。

手順

1. を有効にします `CSINodeExpandSecret` 機能ゲート（ベータ1.25+）。を参照してください "Kubernetes 1.25：CSIボリュームのノードベースの拡張にシークレットを使用します" を参照してください。
2. を追加します `node-expand-secret-name` および `node-expand-secret-namespace` StorageClass パラメータ。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: 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。

Trident は、 1 月、 4 月、 7 月、 10 月の 3 カ月ごとにリリースされます。Kubernetes のリリースから 1 カ月後です。

Astra Trident は、特定のバージョンの **Kubernetes** でリリースされたすべての機能をサポートしていますか。

Astra Trident は、通常、 Kubernetes でアルファ機能をサポートしていません。Trident は、 Kubernetes ベータリリースに続く 2 つの Trident リリースでベータ機能をサポートしています。

Astra Trident には、他のネットアップ製品との依存関係はありますか。

Astra Trident は、他のネットアップソフトウェア製品に依存しないため、スタンドアロンアプリケーションとして機能します。ただし、ネットアップのバックエンドストレージデバイスが必要です。

Astra Trident の設定の詳細をすべて取得するにはどうすればよいですか。

tridentctl get コマンドを使用して 'Astra Trident の構成に関する詳細情報を取得します

Astra Trident を使用してストレージをプロビジョニングする方法に関するメトリクスを取得できますか。

はい。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。を使用することもできます "Cloud Insights の機能です" 監視と分析に使用します。

Astra Trident を **CSI** プロビジョニング担当者として使用すると、ユーザエクスペリエンスは変化しますか。

いいえユーザエクスペリエンスと機能に関する変更はありません。使用されるプロビジョニング名は csi.trident.netapp.io` です現在および将来のリリースで提供される新しい機能をすべて使用する場合は、 Astra Trident をインストールする方法を推奨します。

Kubernetes クラスタに **Astra Trident** をインストールして使用

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、クラスタへのアクセスが「 kubectl 」に設定されている任意のマシンからのリモートインストール機能がサポートされています。kubectl アクセスが確認されたら (たとえば ' リモート・マシンから kubectl get nodes コマンドを起動して確認します) ' インストール手順に従います

Astra Trident でハイアベイラビリティを構成できますか。

Astra Trident は、 1 つのインスタンスで Kubernetes Deployment （ ReplicaSet ）としてインストールされるため、 HA が組み込まれています。導入環境内のレプリカの数は増やすべきではありません。Astra Trident がインストールされているノードが失われた場合や、ポッドにアクセスできない場合は、 Kubernetes によって、クラスタ内の正常なノードにポッドが自動的に再導入されます。Astra Trident はコントロールプレーンのみであるため、 Astra Trident を再導入しても、現在マウントされているポッドには影響しません。

Astra Trident は **kube-system** ネームスペースにアクセスする必要がありますか。

Astra Trident は Kubernetes API Server からデータを読み取り、アプリケーションが新しい PVC を要求するタイミングを判断して、 kube-system へのアクセスを必要とします。

Astra Trident で使用されるロールと権限を教えてください。

Tridentインストーラによって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、「 setup/backend.json 」ファイルに一意の「 StoragePrefix 」パラメータを指定します。これは、両方のインスタンスで同じ FlexVol を使用しないためです。

ContainerLinux （旧 **CoreOS** ）に **Astra Trident** をインストールすることはできますか。

Astra Trident は Kubernetes ポッドとして機能し、 Kubernetes が実行されている場所に導入できます。

ネットアップの **Cloud Volumes ONTAP** で **Astra Trident** を使用できますか。

はい、 Astra Trident は AWS 、 Google Cloud 、 Azure でサポートされています。

Astra Trident は **Cloud Volume** サービスと連携していますか。

はい。 Astra Trident は、 Azure の Azure NetApp Files サービスと GCP の Cloud Volumes Service をサポートしています。

トラブルシューティングとサポート

ネットアップは **Astra Trident** をサポートしていますか。

Astra Trident はオープンソースであり、無償で提供されますが、ネットアップのバックエンドがサポートされていれば、完全にサポートされています。

サポートケースを作成するにはどうすればよいですか？

サポートケースを作成するには、次のいずれかを実行します。

1. サポートアカウントマネージャーに連絡して、チケットの発行に関するサポートを受けてください。
2. 連絡してサポートケースを作成します "ネットアップサポート"。

サポートログバンドルを生成するにはどうすればよいですか？

tridentctl logs-a を実行して ' サポートバンドルを作成できますバンドルでキャプチャされたログに加えて、kubelet ログをキャプチャして、 Kubernetes 側のマウントの問題を診断します。kubelet ログの取得手順は、Kubernetes のインストール方法によって異なります。

新しい機能のリクエストを発行する必要がある場合は、どうすればよいですか。

に問題を作成します "Astra Trident Github" そして、概要の件名と問題に「 * RFE * 」と明記してください。

不具合を発生させる場所

に問題を作成します "Astra Trident Github"。問題に関連する必要なすべての情報とログを記録しておいてください。

ネットアップが **Trident** の **Astra** について簡単に質問できたらどうなりますか。コミュニティやフォーラムはありますか？

ご質問、ご質問、ご要望がございましたら、ネットアップのアストラからお問い合わせください "チャネルを外します" またはGitHub。

ストレージシステムのパスワードが変更され、**Astra Trident**が機能しなくなったため、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 `tridentctl update backend myBackend -f </path/to_new_backend.json> -n trident`。交換してください `myBackend` この例では、バックエンド名にとを指定しています `/path/to_new_backend.json` と入力します `backend.json` ファイル。

Astra Trident が **Kubernetes** ノードを検出できない。この問題を解決するにはどうすればよいですか

Trident が Kubernetes ノードを検出できない場合、次の 2 つのケースが考えられます。Kubernetes または DNS 問題内のネットワーク問題が原因の場合もあります。各 Kubernetes ノードで実行される Trident ノードのデデーモンが 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確認されたバグの修正が必要な場合は、次の手順を実行してください。 "Astra Tridentをアンインストールします" そのバージョンに対応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が使用 "ONTAP の選択的LUNマップ" iSCSI LIFを検出するには、マルチパスセッションを確立する必要があります。の場合は警告が生成されます `dataLIF` は明示的に定義されます。を参照してください "ONTAP のSAN構成オプションと例" を参照してください。
- ONTAP NAS:を指定することを推奨します `dataLIF`。指定しない場合は、Astra TridentがSVMからデータLIFを取得します。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。を参照してください "ONTAP NASの設定オプションと例" を参照してください

Astra Trident が **ONTAP** バックエンドに **CHAP** を設定できるか。

はい。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。これには設定が必要です `useCHAP=true` バックエンド構成

Astra Trident を使用してエクスポートポリシーを管理するにはどうすればよいですか。

Astra Trident では、バージョン 20.04 以降からエクスポートポリシーを動的に作成、管理できます。これにより、ストレージ管理者はバックエンド構成に 1 つ以上の CIDR ブロックを指定でき、 Trident では、その範囲に含まれるノード IP を作成したエクスポートポリシーに追加できます。このようにして、 Astra Trident は特定の CIDR 内に IP アドレスが割り当てられたノードのルールの追加と削除を自動的に管理します。

管理 **LIF** とデータ **LIF** に **IPv6** アドレスを使用できますか。

Astra Tridentでは、次の機能に対してIPv6アドレスを定義できます。

- `managementLIF` および `dataLIF` ONTAP NASバックエンドの場合：
- `managementLIF` ONTAP SANバックエンドの場合：を指定することはできません `dataLIF` ONTAP SANバックエンドの場合：

フラグを使用してAstra Tridentをインストール `--use-ipv6` （ `tridentctl` インストール）、 `IPv6` （Tridentオペレータの場合）、または `tridentTPv6` （Helmインストールの場合）IPv6で機能するようにします。

バックエンドの管理 **LIF** を更新できますか。

はい。 tridentctl update backend コマンドを使用してバックエンド管理 LIF を更新できます。

バックエンドのデータ **LIF** を更新できるか。

のデータLIFを更新できます `ontap-nas` および `ontap-nas-economy` のみ。

Kubernetes 向け **Astra Trident** で複数のバックエンドを作成できますか。

Astra Trident では、同じドライバまたは別々のドライバを使用して、多数のバックエンドを同時にサポートできます。

Astra Trident はバックエンドクレデンシャルをどのように保存しますか。

Astra Trident では、バックエンドのクレデンシャルを Kubernetes のシークレットとして格納します。

Astra Trident ではどのようにして特定のバックエンドを選択しますか。

バックエンド属性を使用してクラスに適切なプールを自動的に選択できない場合は 'toragePools' パラメータと additionalStoragePools' パラメータを使用して ' 特定のプールセットを選択します

Astra Trident が特定のバックエンドからプロビジョニングされないようにするにはどうすればよいですか。

excludeStoragePools' パラメータを使用して 'Astra Trident がプロビジョニングに使用する一連のプールをフィルタリングし ' 一致するプールをすべて削除します

同じ種類のバックエンドが複数ある場合、 **Astra Trident** はどのバックエンドを使用するかをどのように選択しますか。

同じタイプのバックエンドが複数設定されている場合、 Astra Trident は、「 torageClass 」および「 PersistentVolumeClaim 」にあるパラメータに基づいて適切なバックエンドを選択します。たとえば、複数の ONTAP-NAS ドライババックエンドがある場合、 Astra Trident は「 S torageClass 」と「 PersistentVolumeClaim 」のパラメータを組み合わせて照合し、「 S torageClass 」と「 PersistentVolumeClaim 」に記載された要件を提供できるバックエンドと照合します。この要求に一致するバックエンドが複数ある場合、 Astra Trident はいずれかのバックエンドからランダムに選択します。

Astra Trident は、 **Element / SolidFire** で双方向 **CHAP** をサポートしていますか。

はい。

Trident が **ONTAP** ボリュームに **qtree** を導入する方法を教えてください。**1** つのボリュームに配置できる **qtree** の数はいくつですか。

「 ONTAP-NAS-エコノミー 」ドライバは、同一の FlexVol （ 50 ～ 300 の範囲で設定可能）で最大 200 個の qtree を作成し、クラスタ・ノードあたり 100,000 個の qtree を作成し、クラスタあたり 240 万個を作成します。エコノミー・ドライバーがサービスを提供する新しい「 PersistentVolumeClaim 」を入力すると、ドライバーは新しい qtree にサービスを提供できる FlexVol がすでに存在するかどうかを確認します。qtree を提供できる FlexVol が存在しない場合は、新しい FlexVol が作成されます。

ONTAP NAS でプロビジョニングされたボリュームに **UNIX** アクセス権を設定するにはどうすればよいですか。

Astra Trident でプロビジョニングしたボリュームに対して UNIX 権限を設定するには、バックエンド定義ファイルにパラメータを設定します。

ボリュームをプロビジョニングする際に、明示的な **ONTAP NFS** マウントオプションを設定するにはどうすればよいですか。

Trident では、デフォルトでマウントオプションが Kubernetes でどの値にも設定されていません。Kubernetes ストレージクラスでマウントオプションを指定するには、次の例を実行します "こちらをご覧ください"。

プロビジョニングしたボリュームを特定のエクスポートポリシーに設定するにはどうすればよいですか？

適切なホストにボリュームへのアクセスを許可するには、バックエンド定義ファイルに設定されている「 exportPolicy 」パラメータを使用します。

ONTAP を使用して **Astra Trident** 経由でボリューム暗号化を設定する方法を教えてください。

Trident によってプロビジョニングされたボリュームで暗号化を設定するには、バックエンド定義ファイルの暗号化パラメータを使用します。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 を実装するには、「 torageClasses 」を使用します。

Trident 経由でシンプロビジョニングやシックプロビジョニングを指定するにはどうすればよいですか。

ONTAP ドライバは、シンプロビジョニングまたはシックプロビジョニングをサポートします。ONTAP ドライバはデフォルトでシンプロビジョニングに設定されています。シックプロビジョニングが必要な場合は、バックエンド定義ファイルまたは「 torageClass 」を設定する必要があります。両方が設定されている場合は、「 torageClass 」が優先されます。ONTAP で次の項目を設定します。

1. 'S torageClass' で 'provisioningType' 属性を thick に設定します
2. バックエンド定義ファイルで 'backend spaceReserve パラメータを volume に設定して ' シックボリュームを有効にします

誤って **PVC** を削除した場合でも、使用中のボリュームが削除されないようにするにはどうすればよいですか。

Kubernetes では、バージョン 1.10 以降、 PVC 保護が自動的に有効になります。

Astra Trident によって作成された **NFS PVC** を拡張できますか。

はい。Astra Trident によって作成された PVC を拡張できます。ボリュームの自動拡張は ONTAP の機能であり、 Trident には適用されません。

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 'VolumeSnapshotDataSource フィーチャーゲートが有効になっていることを確認します

Astra Trident のボリュームスナップショットをサポートするドライバを教えてください。

現在のところ ' オンデマンドスナップショットのサポートは 'ONTAP-NAS'`ONTAP-NAS-flexgroup 'ONTAP-SAN’ONTAP-SANエコノミー "solidfire-san-SAN'"solidfire-san-""solidfire-san-'"solidfire-san-"" で利用できます「 gcp-cvs` 」と「 azure-NetApp-files 」バックエンドドライバ。

ONTAP を使用して **Astra Trident** でプロビジョニングしたボリュームの **Snapshot** バックアップを作成する方法を教えてください。

これは 'ONTAP-NAS' 'ONTAP-SAN' および 'ONTAP-NAS-flexgroup ドライバで利用できますFlexVol レベルでは「 ONTAP-SAN-エコノミー 」ドライバに「スナップショットポリシー」を指定することもできます。

これは「 ONTAP-NAS-エコノミー 」ドライバでも利用できますが、 FlexVol レベルの細分性ではなく、 qtree レベルの細分性で利用できます。Astra Trident によってプロビジョニングされたボリュームのスナップショットを作成できるようにするには、バックエンドパラメータオプション「 napshotPolicy 」を、 ONTAP バックエンドで定義されている目的のスナップショットポリシーに設定します。ストレージコントローラで作成された Snapshot は Astra Trident で認識されません。

Trident 経由でプロビジョニングしたボリュームの **Snapshot** リザーブの割合を設定できますか。

はい。バックエンド定義ファイルで「スナップショット予約」属性を設定することで、 Astra Trident を介してスナップショットコピーを保存するためのディスク領域の特定の割合を予約できます。バックエンド定義フ

ァイルで「 napshotPolicy 」と「 napshotReserve 」を設定した場合、バックエンドファイルに記載されている「 napshotReserve 」の割合に従ってスナップショット予約の割合が設定されます。「スナップショット予約」の割合の数値が指定されていない場合、 ONTAP はデフォルトでスナップショット予約の割合を 5 に設定します。「スナップショット予約」オプションが「なし」に設定されている場合、スナップショット予約の割合は 0 に設定されます。

ボリュームの **Snapshot** ディレクトリに直接アクセスしてファイルをコピーできますか。

はい。バックエンド定義ファイルで「 snapmirror directionDir 」パラメータを設定することで、 Trident によってプロビジョニングされたボリューム上のスナップショットディレクトリにアクセスできます。

Astra Trident を使用して、ボリューム用の **SnapMirror** をセットアップできますか。

現時点では、 SnapMirror は ONTAP CLI または OnCommand System Manager を使用して外部に設定する必要があります。

永続ボリュームを特定の **ONTAP Snapshot** にリストアするにはどうすればよいですか？

ボリュームを ONTAP Snapshot にリストアするには、次の手順を実行します。

1. 永続ボリュームを使用しているアプリケーションポッドを休止します。
2. ONTAP CLI または OnCommand システムマネージャを使用して、必要な Snapshot にリバートします。
3. アプリケーションポッドを再起動します。

Tridentは、負荷共有ミラーが設定されているSVM**でボリュームをプロビジョニングできますか。**

負荷共有ミラーは、NFS経由でデータを提供するSVMのルートボリューム用に作成できます。ONTAP は、Tridentによって作成されたボリュームの負荷共有ミラーを自動的に更新します。ボリュームのマウントが遅延する可能性があります。Tridentを使用して複数のボリュームを作成する場合、ボリュームをプロビジョニングする方法は、負荷共有ミラーを更新するONTAP によって異なります。

お客様 / テナントごとにストレージクラスの使用状況を分離するにはどうすればよいですか。

Kubernetes では、ネームスペース内のストレージクラスは使用できません。ただし、 Kubernetes を使用すると、ネームスペースごとにストレージリソースクォータを使用することで、ネームスペースごとに特定のストレージクラスの使用量を制限できます。特定のストレージへのネームスペースアクセスを拒否するには、そのストレージクラスのリソースクォータを 0 に設定します。

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 ポッドが正常に起動しない場合（たとえば、 Trident ポッドが 2 つ未満の「 ContainerCreating 」フェーズで停止した場合）、「 kubectl-n trident 」を実行して、展開が trident 、「 kubectl-n trident 」が pod trident- を記述します **-**** 追加のインサイトを提供できます。kubelet ログの取得 (journalctl -xeu kubelet など) も役立ちます。

- Trident ログに十分な情報がない場合は、インストールオプションに基づいてインストールパラメータに「 -d 」フラグを渡して、 Trident のデバッグモードを有効にしてみてください。

  次に './tridentctl logs -n trident` を使用して debug が設定され ' ログ内で 'level=debug msg' を検索していることを確認します

  オペレータとともにインストールされます

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

  すべての Trident ポッドが再起動されます。これには数秒かかることがあります。これを確認するには 'kubectl get pod -n trident' の出力の 'age' 列を確認します

  Astra Trident 20.07 と 20.10 では、「 Torc 」の代わりに「 tprov 」を使用します。

  Helm とともにインストールされます

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

  tridentctl を使用してインストールされます

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

- バックエンド定義に「 debugTraceFlags 」を含めると、バックエンドごとにデバッグログを取得することもできます。たとえば、 Trident ログで API 呼び出しとメソッドの逆数を取得するには、「 `debugTraceFlags: {"API":true,"method" :true,}` を指定します。既存のバックエンドには 'tridentctl backend update で構成された 'ebugTraceFlags' を設定できます
- RedHat CoreOS を使用する場合は 'iscsid がワーカー・ノードで有効になっており ' デフォルトで起動されていることを確認しますこの設定には、 OpenShift MachineConfig を使用するか、イグニッションテンプレートを変更します。
- Trident をで使用する際によく発生する問題です "Azure NetApp Files の特長" テナントとクライアントのシークレットが、必要な権限がないアプリケーションの登録から取得された場合です。Tridentの要件の一覧については、 "Azure NetApp Files の特長" 設定
- コンテナへの PV のマウントに問題がある場合は 'rpcbind' がインストールされていて実行されていることを確認してくださいホスト OS に必要なパッケージ・マネージャを使用して 'rpcbind' が実行されているかどうかを確認しますrpcbind サービスのステータスは 'systemctl status rpcbind' またはそれに相当する処理を実行することで確認できます
- Trident バックエンドが、以前に作業したことがあるにもかかわらず「 failed 」状態であると報告した場合は、バックエンドに関連付けられている SVM/admin クレデンシャルの変更が原因である可能性があります。「 tridentctl update backend 」または Trident ポッドのバウンスを使用してバックエンド情報を更新すると、この問題は修正されます。
- Docker をコンテナランタイムとして Trident をインストールするときに権限の問題が発生した場合は、「 --in cluster=false」 フラグを付けて Trident のインストールを試みてください。これはインストーラポッドを使用せず、「 trident-installer 」ユーザのために発生する許可の問題を回避します。

- 実行に失敗した後のクリーンアップには 'uninstall パラメータ <Uninstalling Trident > を使用しますデフォルトでは、スクリプトは Trident によって作成された CRD を削除しないため、実行中の導入環境でも安全にアンインストールしてインストールできます。
- 以前のバージョンのTridentにダウングレードする場合は、 `tridentctl uninstall` Tridentを削除するコマンド。必要なをダウンロードします "Trident のバージョン" を使用してをインストールします `tridentctl install` コマンドを実行します
- インストールが成功した後、 PVC が「保留中」段階で停止した場合、「 kubectl 」を実行して PVC を記述すると、 Trident がこの PVC の PV のプロビジョニングに失敗した理由を追加情報に提供できます。

オペレータを使用した**Trident**の導入に失敗

オペレータを使用して Trident を導入する場合 'TridentOrchestrator のステータスは 'Installing から Installed に変わります'Failed' ステータスが表示され ' オペレータがそれ自体で回復できない場合は ' 次のコマンドを実行してオペレータのログを確認する必要があります

```
tridentctl logs -l trident-operator
```

trident-operator コンテナのログの末尾には、問題のある場所を示すことができます。たとえば、このような問題の 1 つは、エアーギャップ環境のアップストリームレジストリから必要なコンテナイメージをプルできないことです。

Trident のインストールが失敗した理由を理解するには、「 TridentOrchestrator 」のステータスを確認する必要があります。

```
kubectl describe torc trident-2
Name:         trident-2
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  trident.netapp.io/v1
Kind:         TridentOrchestrator
...
Status:
  Current Installation Params:
    IPv6:
    Autosupport Hostname:
    Autosupport Image:
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:
    Image Pull Secrets:         <nil>
    Image Registry:
    k8sTimeout:
    Kubelet Dir:
    Log Format:
    Silence Autosupport:
    Trident Image:
  Message:                      Trident is bound to another CR 'trident'
  Namespace:                    trident-2
  Status:                       Error
  Version:
Events:
  Type     Reason  Age                From                        Message
  ----     ------  ----               ----                        -------
  Warning  Error   16s (x2 over 16s)  trident-operator.netapp.io  Trident
is bound to another CR 'trident'
```

このエラーは、 Trident のインストールに使用された「 TridentOrchestrator 」がすでに存在することを示します。各 Kubernetes クラスタは Trident のインスタンスを 1 つしか保持できないため、オペレータは任意の時点で作成可能なアクティブな TridentOrchestrator が 1 つだけ存在することを確認します。

また、 Trident ポッドのステータスを確認することで、適切でないものがあるかどうかを確認できます。

```
kubectl get pods -n trident

NAME                                READY   STATUS             RESTARTS
AGE
trident-csi-4p5kq                   1/2     ImagePullBackOff   0
5m18s
trident-csi-6f45bfd8b6-vfrkw        4/5     ImagePullBackOff   0
5m19s
trident-csi-9q5xc                   1/2     ImagePullBackOff   0
5m18s
trident-csi-9v95z                   1/2     ImagePullBackOff   0
5m18s
trident-operator-766f7b8658-ldzsv   1/1     Running            0
8m17s
```

1 つ以上のコンテナイメージがフェッチされなかったため、ポッドが完全に初期化できないことがわかります。

この問題に対処するには、「 TridentOrchestrator 」 CR を編集する必要があります。また、「 TridentOrchestrator 」を削除して、変更された正確な定義を持つ新しいものを作成することもできます。

Tridentの導入に失敗しました `tridentctl`

何が問題になったかを特定するために、インストーラをもう一度「 -d`` 」引数を使用して実行すると、デバッグモードが有効になり、問題の内容を理解するのに役立ちます。

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

問題を解決した後 ' 次のようにインストールをクリーンアップし 'tridentctl install コマンドを再度実行できます

```
./tridentctl uninstall -n trident
INFO Deleted Trident deployment.
INFO Deleted cluster role binding.
INFO Deleted cluster role.
INFO Deleted service account.
INFO Removed Trident user from security context constraint.
INFO Trident uninstallation succeeded.
```

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。

この操作は元に戻せません。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、 "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
を追加します `hostNQN` サブシステムに戻ります。
```

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
"24.02"	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
"23.07"	2024年7月	2025年7月	2026年7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
"23.04"	2024年4月	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"	—	2024年4月	2025年4月
"22.01"	—	—	2025年1月
"21.10"	—	—	2024年10月
"21.07"	—	—	2024年7月

セルフサポート

トラブルシューティング文書の包括的なリストについては、を参照してください。 "ネットアップナレッジベース（ログインが必要）"。また、 Astra に関連する問題のトラブルシューティングに関する情報も参照できます "こちらをご覧ください"。

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）を集めた活発なパブリックコミュニティがあります "チャネルを外します"。プロジェクトに関する一般的な質問をしたり、同じような気のある同僚と関連するトピックについて話し合うのには、この場所が最適です。

NetAppテクニカルサポート

Astra Trident のヘルプを参照するには 'tridentctl logs-a -n trident` を使用してサポートバンドルを作成し 'NetApp Support < 困ったときは > に送信してください

を参照してください。

- "Astra ブログ"
- "Astra Trident のブログ"
- "Kubernetes ハブ"
- "netapp.io のコマンドです"

参照

Astra Trident ポート

Tridentが通信に使用するポートの詳細をご確認ください。

Astra Trident ポート

Astra Trident は次のポート経由で通信：

ポート	目的
8443	バックチャネル HTTPS
8001	Prometheus 指標エンドポイント
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

活性/レディネスプローブポートは、を使用して設置するときに変更できます `--probe-port` フラグ。このポートがワーカーノード上の別のプロセスで使用されていないことを確認することが重要です。

Astra Trident REST API

間 "tridentctl コマンドとオプション" Trident REST APIを使用するには、RESTエンドポイントを直接使用する方法が最も簡単です。

REST APIを使用する状況

REST APIは、Kubernetes以外の環境でAstra Tridentをスタンドアロンバイナリとして使用する高度なインストールに役立ちます。

セキュリティ強化のため、Astra Tridentをぜひご利用ください `REST API` ポッド内で実行されている場合は、デフォルトでlocalhostに制限されます。この動作を変更するには、Astra Tridentを設定する必要があります `-address` 引数をポッド構成で指定します。

REST APIを使用する

これらのAPIの呼び出し方法の例については、デバッグを渡してください (`-d`) フラグ。詳細については、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** ）

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 `-k8s_api_server` をクリックしてKubernetesのサポートを有効にしこれを設定すると、 Trident はポッドの Kubernetes サービスアカウントのクレデンシャルを使用して API サーバに接続します。これは、サービスアカウントが有効になっている Kubernetes クラスタで Trident がポッドとして実行されている場合にのみ機能します。

-k8s_api_server <insecure-address:insecure-port>

このオプションまたは `-k8s_pod` をクリックしてKubernetesのサポートを有効にしTrident を指定すると、セキュアでないアドレスとポートを使用して Kubernetes API サーバに接続されます。これにより、Trident をポッドの外部に導入することができますが、サポートされるのは API サーバへのセキュアでない接続だけです。セキュアに接続するには、Tridentをポッドに搭載し、を使用して導入します `-k8s_pod` オプション

Docker です

-volume_driver <name>

Dockerプラグインの登録時に使用するドライバ名。デフォルトはです `netapp`。

-driver_port <port-number>

UNIXドメインソケットではなく、このポートでリッスンします。

-config <file>

必須。バックエンド構成ファイルへのパスを指定する必要があります。

REST

-address <ip-or-host>

TridentのRESTサーバがリスンするアドレスを指定します。デフォルトは localhost です。localhost で聞いて Kubernetes ポッド内で実行しているときに、 REST インターフェイスにポッド外から直接アクセスすることはできません。使用 `-address ""` RESTインターフェイスにポッドのIPアドレスからアクセスできるようにするため。

> Trident REST インターフェイスは、 127.0.0.1 （ IPv4 の場合）または [::1] （ IPv6 の場合）のみをリスンして処理するように設定できます。

-port <port-number>

TridentのRESTサーバがリスンするポートを指定します。デフォルトは 8000 です。

-rest

RESTインターフェイスを有効にします。デフォルトは true です。

Kubernetes オブジェクトと Trident オブジェクト

リソースオブジェクトの読み取りと書き込みを行うことで、 REST API を使用して Kubernetes や Trident を操作できます。Kubernetes と Trident 、 Trident とストレージ、 Kubernetes とストレージの関係を決定するリソースオブジェクトがいくつかあります。これらのオブジェクトの中には Kubernetes で管理されるものと Trident で管理されるものがあります。

オブジェクトは相互にどのように相互作用しますか。

おそらく、オブジェクト、その目的、操作方法を理解する最も簡単な方法は、 Kubernetes ユーザからのストレージ要求を 1 回だけ処理することです。

1. ユーザーは、「 PersistentVolumeClaim 」を作成して、特定のサイズの新しい「 PersistentVolume 」を、管理者が以前に設定した Kubernetes の「 torageClass 」から要求します。
2. Kubernetes の「 torageClass 」は、 Trident をプロビジョニングツールとして識別し、要求されたクラスのボリュームのプロビジョニング方法を Trident に指示するパラメータを含んでいます。
3. Trident は、対応する「 Backends 」と「 toragePools 」を識別する同じ名前の「 S torageClass 」を参照します。この名前は、このクラスのボリュームのプロビジョニングに使用できます。
4. Trident は、対応するバックエンドにストレージをプロビジョニングし、 2 つのオブジェクトを作成します。 Kubernetes では、「 PersistentVolume 」とは、ボリュームを検索、マウント、処理する方法を Kubernetes に伝える「 PersistentVolume 」と、「 PersistentVolume 」と実際のストレージの関係を保持する Trident 内のボリュームです。
5. Kubernetes は 'PersistentVolumeClaim を新しい 'PersistentVolume' にバインドしますPersistentVolume が実行される任意のホストに PersistentVolume をマウントする 'PersistentVolumeClaim を含むポッド。
6. ユーザーは、 Trident を指す「 VolumeSnapshotClass 」を使用して、既存の PVC の「 VolumeSnapshot 」を作成します。
7. Trident が PVC に関連付けられているボリュームを特定し、バックエンドにボリュームの Snapshot を作成します。また ' スナップショットの識別方法を Kubernetes に指示する 'VolumeSnapshotContent' も作成します
8. ユーザーは 'VolumeSnapshot' をソースとして使用して 'PersistentVolumeClaim を作成できます
9. Trident は必要なスナップショットを識別し、「 PersistentVolume 」と「 Volume 」の作成に関連する一連のステップを実行します。

Kubernetes オブジェクトの詳細については、を参照することを強く推奨します "永続ボリューム" Kubernetes のドキュメントのセクション。

Kubernetes `PersistentVolumeClaim` オブジェクト

Kubernetes の「 PersistentVolumeClaim 」オブジェクトは、 Kubernetes クラスタユーザが作成したストレージの要求です。

Trident では、標準仕様に加えて、バックエンド構成で設定したデフォルト設定を上書きする場合に、ボリューム固有の次のアノテーションを指定できます。

アノテーション	ボリュームオプション	サポートされているドライバ
trident.netapp.io/fileSystem	ファイルシステム	ONTAP-SAN、solidfire-san-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「削除」再利用ポリシーがある場合、 Trident は PV が解放されると（つまり、ユーザーが PVC を削除したときに）、 PV と元のボリュームの両方を削除します。削除操作が失敗した場合、 Trident は PV をマークします。そのような状態で操作が成功するか、 PV が手動で削除されるまで、定期的に再試行します。PV が「 + Retain + 」ポリシーを使用している場合、 Trident はそのポリシーを無視し、管理者が Kubernetes とバックエンドからクリーンアップすると想定します。これにより、ボリュームを削除する前にバックアップまたは検査を行うことができます。PV を削除しても、原因 Trident で元のボリュームが削除されないことに注意してください。REST API (tridentctl') を使用して削除してください。

Trident では CSI 仕様を使用したボリュームスナップショットの作成がサポートされています。ボリュームスナップショットを作成し、それをデータソースとして使用して既存の PVC のクローンを作成できます。これにより、 PVS のポイントインタイムコピーを Kubernetes にスナップショットの形で公開できます。作成した Snapshot を使用して新しい PVS を作成できます。「 +On-Demand Volume Snapshots + 」を見て、これがどのように機能するかを確認してください。

Tridentが提供するのも `cloneFromPVC` および `splitOnClone` クローンを作成するためのアノテーションこれらの注釈を使用して、CSI実装を使用せずにPVCのクローンを作成できます。

例 : ユーザがすでに「 m ysql 」という PVC を持っている場合、ユーザは「 trident.netapp.io/cloneFromPVC: mysql 」などの注釈を使用して「 m ysqlclone 」という新しい PVC を作成できます。このアノテーションセットを使用すると、 Trident はボリュームをゼロからプロビジョニングするのではなく、 MySQL PVC に対応するボリュームのクローンを作成します。

次の点を考慮してください。

- アイドルボリュームのクローンを作成することを推奨します。
- PVC とそのクローンは、同じ Kubernetes ネームスペースに存在し、同じストレージクラスを持つ必要があります。
- また 'ONTAP-NAS' および 'ONTAP-SAN' ドライバを使用すると 'pvc 注釈 trident.netapp.io/splitOnClone` を trident.netapp.io/cloneFromPVC` と組み合わせて設定することが望ましい場合がありますTrident は 'trident.netapp.io/splitOnClone` を true に設定した場合 ' クローン・ボリュームを親ボリュームからスプリットするため ' 一部のストレージ効率を失うことなく ' クローン・ボリュームのライフサイクルを親ボリュームから完全に分離しますtrident.netapp.io/splitOnClone` を設定したり 'false に設定したりしないと ' 親ボリュームとクローンボリューム間の依存関係を作成する代わりに ' バックエンドでのスペース消費が削減されますこれにより ' クローンを最初に削除しない限り ' 親ボリュームを削除できなくなりますクローンをスプリットするシナリオでは、空のデータベースボリュームをクローニングする方法が効果的です。このシナリオでは、ボリュームとそのクローンで使用するデータベースボリュームのサイズが大きく異な

っており、 ONTAP ではストレージ効率化のメリットはありません。

。 `sample-input` Directoryには、Tridentで使用するPVC定義の例が含まれています。を参照してください をクリックして、Tridentボリュームに関連付けられているパラメータと設定の完全な概要を確認します。

Kubernetes `PersistentVolume` オブジェクト

Kubernetes の 'PersistentVolume' オブジェクトは 'Kubernetes クラスタで利用できるようになったストレージの一部ですポッドに依存しないライフサイクルがあります。

> Trident は 'PersistentVolume' オブジェクトを作成し ' プロビジョニングするボリュームに基づいて自動的に Kubernetes クラスタに登録します自分で管理することは想定されていません。

Trident をベースとする「 torageClass 」を参照する PVC を作成すると、 Trident は対応するストレージクラスを使用して新しいボリュームをプロビジョニングし、そのボリュームに新しい PV を登録します。プロビジョニングされたボリュームと対応する PV の構成では、 Trident は次のルールに従います。

- Trident は、 Kubernetes に PV 名を生成し、ストレージのプロビジョニングに使用する内部名を生成します。どちらの場合も、名前がスコープ内で一意であることが保証されます。
- ボリュームのサイズは、 PVC で要求されたサイズにできるだけ近いサイズに一致しますが、プラットフォームによっては、最も近い割り当て可能な数量に切り上げられる場合があります。

Kubernetes `StorageClass` オブジェクト

Kubernetes の「 torageClass 」オブジェクトは、「 PersistentVolumeClaims 」内の名前によって指定され、一連のプロパティを持つストレージをプロビジョニングします。ストレージクラス自体が、使用するプロビジョニングツールを特定し、プロビジョニングツールが理解できる一連のプロパティを定義します。

管理者が作成および管理する必要がある 2 つの基本オブジェクトのうちの 1 つです。もう 1 つは Trident バックエンドオブジェクトです。

Trident を使用する Kubernetes の「 torageClass 」オブジェクトは次のようになります。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

これらのパラメータは Trident 固有で、クラスのボリュームのプロビジョニング方法を Trident に指示します。

ストレージクラスのパラメータは次のとおりです。

属性	を入力します	必須	説明
属性（ Attributes ）	[string] 文字列をマップします	いいえ	後述の「属性」セクションを参照してください
ストレージプール	[string] StringList をマップします	いいえ	内のストレージプールのリストへのバックエンド名のマッピング
AdditionalStoragePools	[string] StringList をマップします	いいえ	内のストレージプールのリストへのバックエンド名のマッピング
excludeStoragePools	[string] StringList をマップします	いいえ	内のストレージプールのリストへのバックエンド名のマッピング

ストレージ属性とその有効な値は、ストレージプールの選択属性と Kubernetes 属性に分類できます。

ストレージプールの選択の属性

これらのパラメータは、特定のタイプのボリュームのプロビジョニングに使用する Trident で管理されているストレージプールを決定します。

属性	を入力します	値	提供	リクエスト	でサポートされます
メディア ^1	文字列	HDD 、ハイブリッド、 SSD	プールにはこのタイプのメディアが含まれています。ハイブリッドは両方を意味します	メディアタイプが指定されました	ONTAPNAS 、ONTAPNAS エコノミー、ONTAP-NAS-flexgroup 、ONTAPSAN 、solidfire-san-SAN 、 solidfire-san-SAN のいずれかに対応しています
プロビジョニングタイプ	文字列	シン、シック	プールはこのプロビジョニング方法をサポートします	プロビジョニング方法が指定されました	シック：All ONTAP ；thin ：All ONTAP & solidfire-san-SAN

属性	を入力します	値	提供	リクエスト	でサポートされます
backendType	文字列	ONTAPNAS、ONTAPNASエコノミー、ONTAP-NAS-flexgroup、ONTAPSAN、solidfire-san-SAN、solidfire-san-SAN、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 ' 属性だけを使用して ' 特定のクラスのニーズを満たすために必要なストレージの特性をモデル化できますTrident は ' 指定した属性の *ALL* に一致するストレージ・プールを自動的に検出して選択します

「 attributes 」を使用してクラスに適切なプールを自動的に選択できない場合は、「 toragePools 」および「 additionalStoragePools 」パラメータを使用してプールをさらに改良したり、特定のプールセットを選択したりできます。

'toragePools' パラメータを使用すると ' 指定した属性に一致するプールのセットをさらに制限できますつまり

'attributes' パラメータと 'toragePools' パラメータで指定されたプールの交点をプロビジョニングに使用しますどちらか一方のパラメータを単独で使用することも、両方を同時に使用することも

「 additionalStoragePools 」パラメータを使用すると、「 attributes 」パラメータと「 toragePools 」パラメータで選択されたプールに関係なく、 Trident がプロビジョニングに使用するプールのセットを拡張できます。

excludeStoragePools' パラメータを使用して、 Trident がプロビジョニングに使用するプールのセットをフィルタリングできます。このパラメータを使用すると、一致するプールがすべて削除されます。

'toragePools' パラメータと 'additionalStoragePools' パラメータでは ' 各エントリは '<backend>:<storagePoolList>' の形式で指定したバックエンドのストレージプールのカンマ区切りリストですたとえば、「 additionalStoragePools 」の値は「 ontapnas_192.168.1.100 ： aggr1 、 aggr2 ； solidfire_192.168.1.101 ： bronze 」のようになります。これらのリストでは、バックエンド値とリスト値の両方に正規表現値を使用できます。tridentctl get backend を使用してバックエンドとそのプールのリストを取得できます

Kubernetes の属性

これらの属性は、動的プロビジョニングの際に Trident が選択するストレージプール / バックエンドには影響しません。代わりに、 Kubernetes Persistent Volume でサポートされるパラメータを提供するだけです。ワーカーノードはファイルシステムの作成操作を担当し、 xfsprogs などのファイルシステムユーティリティを必要とする場合があります。

属性	を入力します	値	説明	関連するドライバ	**Kubernetes** のバージョン
FSstype （英語）	文字列	ext4 、 ext3 、 xfs など	ブロックボリュームのファイルシステムのタイプ	solidfire-san-group、ontap/nas、ontap -nas-エコノミー、ontap -nas-flexgroup 、ontap -san 、ONTAP - SAN -経済性	すべて
allowVolumeExpansion の略	ブール値	true false	PVC サイズの拡張のサポートをイネーブルまたはディセーブルにします	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup 、 ONTAPSAN 、 ONTAP-SAN-エコノミー 、 solidfire-san-, gcvs, azure-netapp-files	1.11 以上
volumeBindingMode のようになりました	文字列	即時、 WaitForFirstConsumer	ボリュームバインドと動的プロビジョニングを実行するタイミングを選択します	すべて	1.19〜1.26

- 。 `fsType` パラメータは、SAN LUNに必要なファイルシステムタイプを制御する場合に使用します。また、Kubernetesでは、の機能も使用されます `fsType` ファイルシステムが存在することを示すために、ストレージクラスに格納します。ボリューム所有権は、を使用して制御できます `fsGroup` ポッドのセキュリティコンテキスト（使用する場合のみ）`fsType` が設定されます。を参照してください "Kubernetes ：ポッドまたはコンテナのセキュリティコンテキストを設定します" を使用したボリューム所有権の設定の概要については、を参照してください `fsGroup` コンテキスト（Context）。Kubernetesでが適用されます `fsGroup` 次の場合のみ値を指定します

  - 「 fsType 」はストレージクラスで設定されます。
  - PVC アクセスモードは RWO です。

  NFS ストレージドライバの場合、 NFS エクスポートにはファイルシステムがすでに存在します。fsGroup を使用するには ' ストレージ・クラスで fsType を指定する必要があります この値は 'NFS' に設定することも ' ヌル以外の任意の値に設定することもできます

- を参照してください "ボリュームを展開します" ボリューム拡張の詳細については、を参照してください。
- Trident インストーラバンドルには、「 `sample -input/storageclass-*.yaml 」で Trident で使用するストレージクラス定義の例がいくつか用意されています。Kubernetes ストレージクラスを削除すると、対応する Trident ストレージクラスも削除されます。

Kubernetes `VolumeSnapshotClass` オブジェクト

Kubernetes 'VolumeSnapshotClass' オブジェクトは 'S torageClasses' に似ていますこの Snapshot コピーは、複数のストレージクラスの定義に役立ちます。また、ボリューム Snapshot によって参照され、Snapshot を必要な Snapshot クラスに関連付けます。各ボリューム Snapshot は、単一のボリューム Snapshot クラスに関連付けられます。

スナップショットを作成するには 'VolumeSnapshotClass' を管理者が定義する必要がありますボリューム Snapshot クラスは、次の定義で作成されます。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

「 driver 」は、「 csi-snapclass 」クラスのボリュームスナップショットの要求が Trident によって処理される Kubernetes を指定します。「要素ポリシー」は、スナップショットを削除する必要がある場合に実行されるアクションを指定します。「削除ポリシー」が「削除」に設定されている場合、 Snapshot を削除すると、ボリューム Snapshot オブジェクトおよびストレージクラスタ上の基盤となる Snapshot は削除されます。または、「 Retain 」に設定すると、「 VolumeSnapshotContent 」と物理スナップショットが保持されます。

Kubernetes `VolumeSnapshot` オブジェクト

Kubernetes の VolumeSnapshot オブジェクトは ' ボリュームのスナップショットを作成する要求ですPVC が

ボリュームに対するユーザからの要求を表すのと同様に、ボリュームスナップショットは、ユーザが既存の PVC のスナップショットを作成する要求です。

ボリュームスナップショット要求が受信されると、 Trident はバックエンドでのボリュームのスナップショット作成を自動的に管理し、ユニークな「 VolumeSnapshotContent 」オブジェクトを作成することによってスナップショットを公開します。既存の PVC からスナップショットを作成し、新しい PVC を作成するときにスナップショットを DataSource として使用できます。

> VolumeSnapshot のライフサイクルはソース PVC とは無関係です。ソース PVC が削除されても、スナップショットは維持されます。スナップショットが関連付けられている PVC を削除すると、 Trident はその PVC のバッキングボリュームを **Deleting** 状態でマークしますが、完全には削除しません。関連付けられている Snapshot がすべて削除されると、ボリュームは削除されます。

Kubernetes `VolumeSnapshotContent` オブジェクト

Kubernetes の「 VolumeSnapshotContent 」オブジェクトは、すでにプロビジョニングされているボリュームから取得されたスナップショットを表します。これは「 PersistentVolume 」と似ており、ストレージ・クラスタ上でプロビジョニングされた Snapshot を表します。「 PersistentVolumeClaim 」および「 PersistentVolume 」オブジェクトと同様に、スナップショットが作成されると、「 VolumeContent Snapshot」 オブジェクトは「 VolumeSnapshot 」オブジェクトへの 1 対 1 のマッピングを保持します。これは、スナップショットの作成を要求しました。

「 VolumeSnapshotContent 」オブジェクトには、スナップショットを一意に識別する詳細（「 napshotHandle 」など）が含まれています。この「 napshotHandle 」は、 PV の名前と「 VolumeSnapshotContent 」オブジェクトの名前を組み合わせた一意のものです。

Trident では、スナップショット要求を受信すると、バックエンドにスナップショットが作成されます。スナップショットが作成されると、 Trident は「 VolumeSnapshotContent 」オブジェクトを構成し、そのスナップショットを Kubernetes API に公開します。

> 通常は、 `VolumeSnapshotContent` オブジェクト。ただし、次の場合は例外です。 "ボリュームSnapshotのインポート" Astra Trident以外で作成

Kubernetes `CustomResourceDefinition` オブジェクト

Kubernetes カスタムリソースは、管理者が定義した Kubernetes API 内のエンドポイントであり、類似するオブジェクトのグループ化に使用されます。Kubernetes では、オブジェクトのコレクションを格納するためのカスタムリソースの作成をサポートしています。これらのリソース定義を取得するには 'kubectl get CRDs' を実行します

カスタムリソース定義（ CRD ）と関連するオブジェクトメタデータは、 Kubernetes によってメタデータストアに格納されます。これにより、 Trident の独立したストアが不要になります。

Astra Tridentが使用 `CustomResourceDefinition` Tridentバックエンド、Tridentストレージクラス、Tridentボリュームなど、TridentオブジェクトのIDを保持するオブジェクト。これらのオブジェクトは Trident によって管理されます。また、 CSI のボリュームスナップショットフレームワークには、ボリュームスナップショットの定義に必要ないくつかの SSD が導入されています。

CRD は Kubernetes の構成要素です。上記で定義したリソースのオブジェクトは Trident によって作成されます。簡単な例として 'tridentctl' を使用してバックエンドを作成すると ' 対応する tridentBackendsCRD オブジェクトが Kubernetes によって消費されるように作成されます

Trident の CRD については、次の点に注意してください。

- Trident をインストールすると、一連の CRD が作成され、他のリソースタイプと同様に使用できるようになります。
- Tridentをアンインストールするには、を使用します `tridentctl uninstall` コマンドであるTridentポッドが削除されましたが、作成されたSSDはクリーンアップされません。を参照してください "Trident をアンインストールします" Trident を完全に削除して再構成する方法を理解する。

Astra Trident `StorageClass` オブジェクト

TridentではKubernetesに対応するストレージクラスが作成されます `StorageClass` を指定するオブジェクト `csi.trident.netapp.io` プロビジョニング担当者のフィールドに入力します。ストレージクラス名がKubernetesの名前と一致していること `StorageClass` 表すオブジェクト。

> Kubernetes では、 Trident をプロビジョニングツールとして使用する Kubernetes 「 torageClass 」が登録されると、これらのオブジェクトが自動的に作成されます。

ストレージクラスは、ボリュームの一連の要件で構成されます。Trident は、これらの要件と各ストレージプール内の属性を照合し、一致する場合は、そのストレージプールが、そのストレージクラスを使用するボリュームのプロビジョニングの有効なターゲットになります。

REST API を使用して、ストレージクラスを直接定義するストレージクラス設定を作成できます。ただし、Kubernetes の導入では、新しい Kubernetes の「 torageClass 」オブジェクトを登録するときに、これらのオブジェクトが作成されることを期待しています。

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 の「 torageClass 」オブジェクトです。

これらのオブジェクトの作成方法の詳細については、を参照してください。 "バックエンドの設定"。

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 'PersistentVolumes' に直接対応しますボリュームを作成するときは、そのボリュームにストレージクラスが含まれていることを確認します。このクラスによって、ボリュームをプロビジョニングできる場所とサイズが決まります。

- Kubernetes では、これらのオブジェクトが自動的に管理されます。Trident がプロビジョニングしたものを表示できます。
- 関連付けられた Snapshot がある PV を削除すると、対応する Trident ボリュームが * Deleting * 状態に更新されます。Trident ボリュームを削除するには、ボリュームの Snapshot を削除する必要があります。

ボリューム構成は、プロビジョニングされたボリュームに必要なプロパティを定義します。

属性	を入力します	必須	説明
バージョン	文字列	いいえ	Trident API のバージョン（「 1 」）
名前	文字列	はい。	作成するボリュームの名前
ストレージクラス	文字列	はい。	ボリュームのプロビジョニング時に使用するストレージクラス
サイズ	文字列	はい。	プロビジョニングするボリュームのサイズ（バイト単位）
プロトコル	文字列	いいえ	使用するプロトコルの種類：「 file 」または「 block 」
インターン名	文字列	いいえ	Trident が生成した、ストレージシステム上のオブジェクトの名前
cloneSourceVolume の実行中です	文字列	いいえ	ONTAP （ NAS 、 SAN ） & SolidFire - * ：クローン元のボリュームの名前
splitOnClone	文字列	いいえ	ONTAP （ NAS 、 SAN ）：クローンを親からスプリットします
Snapshot ポリシー	文字列	いいえ	ONTAP - * ：使用する Snapshot ポリシー
Snapshot リザーブ	文字列	いいえ	ONTAP - * ： Snapshot 用にリザーブされているボリュームの割合
エクスポートポリシー	文字列	いいえ	ONTAP-NAS* ：使用するエクスポートポリシー
snapshotDirectory の略	ブール値	いいえ	ONTAP-NAS* ： Snapshot ディレクトリが表示されているかどうか
unixPermissions	文字列	いいえ	ONTAP-NAS* ：最初の UNIX 権限

属性	を入力します	必須	説明
ブロックサイズ	文字列	いいえ	SolidFire - * ：ブロック / セクターサイズ
ファイルシステム	文字列	いいえ	ファイルシステムのタイプ

Trident は ' ボリュームの作成時に internalName を生成しますこの構成は 2 つのステップで構成されます。最初に、ストレージプレフィックス（デフォルトの「 trident 」またはバックエンド構成のプレフィックス）をボリューム名の前に付加し、「 <prefix> - <volume-name> 」という形式の名前を付けます。その後、名前の完全消去が行われ、バックエンドで許可されていない文字が置き換えられます。ONTAP バックエンドでは、ハイフンをアンダースコアで置き換えます（つまり、内部名は「 <prefix>_<volume-name>` 」になります）。Element バックエンドの場合、アンダースコアはハイフンに置き換えられます。

ボリューム設定を使用して、 REST API を使用してボリュームを直接プロビジョニングできますが、Kubernetes 環境では、ほとんどのユーザが標準の Kubernetes の「 PersistentVolumeClaim 」メソッドを使用することを想定しています。Trident は、プロビジョニングプロセスの一環として、このボリュームオブジェクトを自動的に作成します。

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 'VolumeSnapshotContent' オブジェクトに直接対応します各 Snapshot には、 Snapshot のデータのソースであるボリュームが関連付けられます。

個々の「スナップショット」オブジェクトには、以下のプロパティが含まれています。

属性	を入力します	必須	説明
バージョン	文字列	はい。	Trident API のバージョン（「 1 」）
名前	文字列	はい。	Trident Snapshot オブジェクトの名前
インターン名	文字列	はい。	ストレージシステム上の Trident Snapshot オブジェクトの名前
ボリューム名	文字列	はい。	Snapshot を作成する永続的ボリュームの名前
ボリュームの内部名	文字列	はい。	ストレージシステムに関連付けられている Trident ボリュームオブジェクトの名前

Kubernetes では、これらのオブジェクトが自動的に管理されます。Trident がプロビジョニングしたものを表示できます。

Kubernetes の「 VolumeSnapshot 」オブジェクト要求が作成されると、 Trident は元のストレージシステム上にスナップショットオブジェクトを作成することによって動作します。このスナップショットオブジェクトの「 internalName 」は、プレフィックス「 snapshot-` と「 VolumeSnapshot 」オブジェクトの「 UID 」を組み合わせることによって生成されます（例：「 snapshot-e8d8d8a0ca-9826-11e9-9807-525400f3f660 」

）。「 volumeName 」と「 volumeInternalName 」には、バッキングボリュームの詳細を取得して値を設定します。

Astra Trident `ResourceQuota` オブジェクト

Tridentのデーモンは'system-node -criticalのPriority Classを消費しますこれはKubernetesで最も高い優先順位クラスですこのクラスにより'Astra Tridentは'正常なノードのシャットダウン時にボリュームを識別してクリーンアップし'リソースの負荷が高いクラスタではTridentのデマ起動ポッドがより低い優先順位でワークロードをプリエンプトできるようになります

これを実現するために、Astra Tridentは「ResourceQuota」オブジェクトを使用して、Tridentのデミスタに対する「システムノードクリティカル」の優先クラスが満たされていることを確認します。Astra Tridentは、展開とデミスの作成を行う前に、「ResourceQuota」オブジェクトを検索し、検出されない場合はそのオブジェクトを適用します。

デフォルトのリソース割り当ておよび優先クラスをより詳細に制御する必要がある場合は'custom.yamlを生成するか'Helmチャートを使用してResourceQuotaオブジェクトを構成します

次に示すのは'ResourceQuota'オブジェクトがTridentのデマ作用を優先する例です

```
apiVersion: <version>
kind: ResourceQuota
metadata:
  name: trident-csi
  labels:
    app: node.csi.trident.netapp.io
spec:
  scopeSelector:
     matchExpressions:
       - operator : In
         scopeName: PriorityClass
         values: ["system-node-critical"]
```

リソースクォータの詳細については、を参照してください。 "Kubernetes：リソースクォータ"。

クリーンアップ `ResourceQuota` インストールが失敗した場合

まれに'ResourceQuotaオブジェクトが作成された後にインストールが失敗する場合は'最初に実行します "アンインストール中です" を再インストールします。

それでも解決しない場合は'ResourceQuotaオブジェクトを手動で削除します

取り外します `ResourceQuota`

独自のリソース割り当てを制御する場合は、コマンドを使用してAstraのResourceQuotaオブジェクトを削除できます。

```
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

アクセス権	説明
権限があります	CSIでは、マウントポイントが双方向である必要があります。つまり、Tridentノードポッドで特権コンテナを実行する必要があります。詳細については、を参照してください "Kubernetes：マウントの伝播"。
ホストネットワーク	iSCSIデーモンに必要です。「iscsiadm」はiSCSIマウントを管理し、ホストネットワークを使用してiSCSIデーモンと通信します。
ホストIPC	NFSは'IPC（プロセス間通信）を使用して'nfsdと通信します
ホストPID	NFSの場合'rpc-statd'を起動するために必要ですAstra Tridentは'ホスト・プロセスに対して'rpc-statd'がNFSボリュームをマウントする前に実行されているかどうかを問い合わせます
機能	SYS_Admin'機能は'特権コンテナのデフォルト機能の一部として提供されますたとえば、Dockerは特権コンテナにこれらの機能を設定します。「CapPrm：0000003ffffffffffffff`Capff：0000003ffffffffff」
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。

アクセス権	説明
SELinux	OpenShiftでは、特権コンテナは「SPC_t」（「スーパー特権コンテナ」）ドメインで実行され、非特権コンテナは「container_t」ドメインで実行されます。containerd'に'container-bSELinuxがインストールされている場合'すべてのコンテナは'spc_t'ドメイン内で実行されますこのドメインでは'SELinuxが効果的に無効になりますそのため、Astra Trident は「seLinuxOptions」をコンテナに追加しません。
DAC	特権コンテナは、ルートとして実行する必要があります。CSIに必要なUNIXソケットにアクセスするために、非特権コンテナはrootとして実行されます。

PODセキュリティ標準（PSS）

ラベル	説明	デフォルト
'pod -security.esvus.io/enforce `po-security.Kubernetes io/enforce-version	Tridentコントローラとノードをインストールネームスペースに登録できるようにします。ネームスペースラベルは変更しないでください。	「enforce：特権」「enforce-version：」は、現在のクラスタのバージョンまたはテストされたPSSの最新バージョンです

名前空間ラベルを変更すると、ポッドがスケジュールされず、「Error creating：…」または「Warning：trident-csi-…」が表示される場合があります。この場合は'Privilegedの名前空間ラベルが変更されていないかどうかを確認してくださいその場合は、Tridentを再インストールします。

PoDセキュリティポリシー（PSP）

フィールド	説明	デフォルト
'allowPrivilegeEscalation'	特権コンテナは、特権昇格を許可する必要があります。	「真」
allowedCSIDrivers	TridentはインラインCSIエフェメラルボリュームを使用しません。	空です
「allowedCapabilities」を参照してください	権限のないTridentコンテナにはデフォルトよりも多くの機能が必要ないため、特権コンテナには可能なすべての機能が付与されます。	空です
「allowedFlexVolumes」を参照してください	Tridentはを利用しません "FlexVolドライバ"そのため、これらのボリュームは許可されるボリュームのリストに含まれていません。	空です
「allowedHostPaths」を参照してください	Tridentノードポッドでノードのルートファイルシステムがマウントされるため、このリストを設定してもメリットはありません。	空です

フィールド	説明	デフォルト
allowedProcMountTypes	Tridentでは’ProcMountTypes’は使用しません	空です
"allowedUnsafeSysctls"	Tridentには安全でないsysctls’は必要ありません。	空です
defaultAddCapabilities	特権コンテナに追加する機能は必要ありません。	空です
defaultAllowPrivilegeEscalation`	権限の昇格は、各Tridentポッドで処理されます。	「偽」
「forbiddenSysctls」と入力します	sysctlsは許可されません	空です
「fsGroup」と入力します	Tridentコンテナはrootとして実行されます。	「RunAsAny」
「hostIPC」	NFSボリュームをマウントするには’nfsdと通信するためにホストIPCが必要です	「真」
「ホストネットワーク」	iscsiadmには、iSCSIデーモンと通信するためのホストネットワークが必要です。	「真」
「hostPID」	ノードでRPC-statdが実行されているかどうかを確認するには’ホストPIDが必要です	「真」
「hostPorts」	Tridentはホストポートを使用しません。	空です
「特権」	Tridentノードのポッドでは、ボリュームをマウントするために特権コンテナを実行する必要があります。	「真」
「readOnlyRootFilesystem」	Tridentノードのポッドは、ノードのファイルシステムに書き込む必要があります。	「偽」
DDropCapabilitiesが必要です	Tridentノードのポッドは特権コンテナを実行するため、機能をドロップすることはできません。	「 NONE 」
runAsGroup`	Tridentコンテナはrootとして実行されます。	「RunAsAny」
runAsUser	Tridentコンテナはrootとして実行されます。	runAsany`
runtimeClass'	Tridentは’RuntimeClasses’を使用しません	空です

フィールド	説明	デフォルト
SELinux	Tridentは’seLinuxOptions’を設定していませんこれは’コンテナランタイムとKubernetesディストリビューションがSELinuxを処理する方法には現在のところ違いがあるためです	空です
「supplementalGroups」を参照してください	Tridentコンテナはrootとして実行されます。	「RunAsAny」
「ボリューム」	Tridentポッドには、このボリュームプラグインが必要です。	「hostPath」、「projected」、「emptyDir」

セキュリティコンテキストの制約（SCC）

ラベル	説明	デフォルト
allowHostDirVolumePlugin	Tridentノードのポッドは、ノードのルートファイルシステムをマウントします。	「真」
"allowHostIPC"	NFSボリュームをマウントするには’nfsdと通信するためにホストIPCが必要です	「真」
「allowHostNetwork」を参照してください	iscsiadmには、iSCSIデーモンと通信するためのホストネットワークが必要です。	「真」
「allowHostPID」を参照してください	ノードでRPC-statdが実行されているかどうかを確認するには’ホストPIDが必要です	「真」
"allowHostPorts`	Tridentはホストポートを使用しません。	「偽」
'allowPrivilegeEscalation'	特権コンテナは、特権昇格を許可する必要があります。	「真」
allowPrivilegeContainer]を参照してください	Tridentノードのポッドでは、ボリュームをマウントするために特権コンテナを実行する必要があります。	「真」
"allowedUnsafeSysctls"	Tridentには安全でないsysctls’は必要ありません。	「 NONE 」
「allowedCapabilities」を参照してください	権限のないTridentコンテナにはデフォルトよりも多くの機能が必要ないため、特権コンテナには可能なすべての機能が付与されます。	空です
defaultAddCapabilities	特権コンテナに追加する機能は必要ありません。	空です
「fsGroup」と入力します	Tridentコンテナはrootとして実行されます。	「RunAsAny」

ラベル	説明	デフォルト
「グループ」	このSCCはTridentに固有で、ユーザにバインドされています。	空です
「readOnlyRootFilesystem」	Tridentノードのポッドは、ノードのファイルシステムに書き込む必要があります。	「偽」
DDropCapabilitiesが必要です	Tridentノードのポッドは特権コンテナを実行するため、機能をドロップすることはできません。	「 NONE 」
runAsUser	Tridentコンテナはrootとして実行されます。	「RunAsAny」
「seLinuxContext」	Tridentは’seLinuxOptions’を設定していませんこれは’コンテナランタイムとKubernetesディストリビューションがSELinuxを処理する方法には現在のところ違いがあるためです	空です
「seccompProfiles」	特権のあるコンテナは常に「閉鎖的」な状態で実行されます。	空です
「supplementalGroups」を参照してください	Tridentコンテナはrootとして実行されます。	「RunAsAny」
「ユーザー」	このSCCをTridentネームスペースのTridentユーザにバインドするエントリが1つあります。	該当なし
「ボリューム」	Tridentポッドには、このボリュームプラグインが必要です。	「hostPath」, downwardAPI, projected , emptyDir

法的通知

著作権に関する声明、商標、特許などにアクセスできます。

著作権

"https://www.netapp.com/company/legal/copyright/"

商標

NetApp 、 NetApp のロゴ、および NetApp の商標ページに記載されているマークは、 NetApp, Inc. の商標です。その他の会社名および製品名は、それぞれの所有者の商標である場合があります。

"https://www.netapp.com/company/legal/trademarks/"

特許

ネットアップが所有する特許の最新リストは、次のサイトで入手できます。

https://www.netapp.com/pdf.html?item=/media/11887-patentspage.pdf

プライバシーポリシー

"https://www.netapp.com/company/legal/privacy-policy/"

オープンソース

ネットアップの Astra Trident 向けソフトウェアで使用されているサードパーティの著作権とライセンスは、の各リリースの通知ファイルで確認できます https://github.com/NetApp/trident/。

著作権に関する情報

Copyright © 2024 NetApp, Inc. All Rights Reserved. Printed in the U.S.このドキュメントは著作権によって保護されています。著作権所有者の書面による事前承諾がある場合を除き、画像媒体、電子媒体、および写真複写、記録媒体、テープ媒体、電子検索システムへの組み込みを含む機械媒体など、いかなる形式および方法による複製も禁止します。

ネットアップの著作物から派生したソフトウェアは、次に示す使用許諾条項および免責条項の対象となります。

このソフトウェアは、ネットアップによって「現状のまま」提供されています。ネットアップは明示的な保証、または商品性および特定目的に対する適合性の暗示的保証を含み、かつこれに限定されないいかなる暗示的な保証も行いません。ネットアップは、代替品または代替サービスの調達、使用不能、データ損失、利益損失、業務中断を含み、かつこれに限定されない、このソフトウェアの使用により生じたすべての直接的損害、間接的損害、偶発的損害、特別損害、懲罰的損害、必然的損害の発生に対して、損失の発生の可能性が通知されていたとしても、その発生理由、根拠とする責任論、契約の有無、厳格責任、不法行為（過失またはそうでない場合を含む）にかかわらず、一切の責任を負いません。

ネットアップは、ここに記載されているすべての製品に対する変更を随時、予告なく行う権利を保有します。ネットアップによる明示的な書面による合意がある場合を除き、ここに記載されている製品の使用により生じる責任および義務に対して、ネットアップは責任を負いません。この製品の使用または購入は、ネットアップの特許権、商標権、または他の知的所有権に基づくライセンスの供与とはみなされません。

このマニュアルに記載されている製品は、1つ以上の米国特許、その他の国の特許、および出願中の特許によって保護されている場合があります。

権利の制限について：政府による使用、複製、開示は、DFARS 252.227-7013（2014年2月）およびFAR 5252.227-19（2007年12月）のRights in Technical Data -Noncommercial Items（技術データ - 非商用品目に関する諸権利）条項の(b)(3)項、に規定された制限が適用されます。

本書に含まれるデータは商用製品および / または商用サービス（FAR 2.101の定義に基づく）に関係し、データの所有権はNetApp, Inc.にあります。本契約に基づき提供されるすべてのネットアップの技術データおよびコンピュータ ソフトウェアは、商用目的であり、私費のみで開発されたものです。米国政府は本データに対し、非独占的かつ移転およびサブライセンス不可で、全世界を対象とする取り消し不能の制限付き使用権を有し、本データの提供の根拠となった米国政府契約に関連し、当該契約の裏付けとする場合にのみ本データを使用できます。前述の場合を除き、NetApp, Inc.の書面による許可を事前に得ることなく、本データを使用、開示、転載、改変するほか、上演または展示することはできません。国防総省にかかる米国政府のデータ使用権については、DFARS 252.227-7015(b)項（2014年2月）で定められた権利のみが認められます。

商標に関する情報

NetApp、NetAppのロゴ、http://www.netapp.com/TMに記載されているマークは、NetApp, Inc.の商標です。その他の会社名と製品名は、それを所有する各社の商標である場合があります。