



Astra Trident 22.10

Astra Trident

NetApp
April 16, 2024

目次

Astra Trident 22.10	1
リリースノート	2
22.10の新機能	2
2007年22月の変更	3
2004年10月22日の変更	4
22.01.1 の変更	5
22.01.0での変更点	5
21.10.1 の変更点	6
21.10.0ノヘンコウテン	6
既知の問題	7
詳細については、こちらをご覧ください	8
概念	9
Astra Trident の詳細をご確認ください	9
ONTAP ドライバ	10
プロビジョニング	11
ボリューム Snapshot	12
仮想ストレージプール	12
ボリュームアクセスグループ	14
はじめに	15
ぜひお試しください	15
要件	15
導入の概要	20
Trident オペレータとともに導入	23
tridentctl を使用して導入します	34
次の手順	38
Trident で Astra を管理	43
Astra Trident をアップグレード	43
オペレータにアップグレードしてください	45
tridentctl を使用してアップグレードします	53
Astra Trident をアンインストール	56
Trident をダウングレード	58
Astra Trident を使用	62
ワーカーノードを準備します	62
バックエンドを設定	66
kubectl を使用してバックエンドを作成します	140
kubectl を使用してバックエンド管理を実行します	147
tridentctl を使用してバックエンド管理を実行します	148
バックエンド管理オプション間を移動します	150
ストレージクラスを管理する	156

ボリューム操作を実行する	158
ネームスペース間でNFSボリュームを共有します	183
Astra Trident を監視	187
Trident for Docker が必要です	192
導入の前提条件	192
Astra Trident を導入	195
Astra Trident をアップグレードまたはアンインストールする	199
ボリュームを操作します	201
ログを収集します	210
複数の Astra Trident インスタンスを管理	211
ストレージ構成オプション	212
既知の問題および制限事項	228
よくある質問	230
一般的な質問	230
Kubernetes クラスタに Astra Trident をインストールして使用	230
トラブルシューティングとサポート	232
Astra Trident をアップグレード	233
バックエンドとボリュームを管理	233
サポート	239
トラブルシューティング	240
全般的なトラブルシューティング	240
オペレータを使用して失敗した Trident の導入をトラブルシューティングします	242
を使用したTridentの導入に失敗した場合のトラブルシューティング <code>tridentctl</code>	244
ベストプラクティスと推奨事項	245
導入	245
ストレージ構成	245
Astra Trident を統合	252
データ保護	263
セキュリティ	268
参照	272
Astra Trident ポート	272
Astra Trident REST API	272
コマンドラインオプション	273
ネットアップの製品が Kubernetes と統合されます	274
Kubernetes オブジェクトと Trident オブジェクト	275
<code>tridentctl</code> コマンドとオプション	288
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)	294
以前のバージョンのドキュメント	299
法的通知	300
著作権	300
商標	300

特許	300
プライバシーポリシー	300
オープンソース	300

Astra Trident 22.10

リリースノート

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



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

22.10の新機能

- 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。Tridentがアクティブなワークロードに影響を与えずに安全であると判断した場合、既存のボリュームは新しいigroupスキームに適切切り替えられます。
- TridentのインストールにResourceQuotaが含まれ、PriorityClassの消費がデフォルトで制限されたときにTrident DemonSetがスケジュールされるようになりました。
- ANFドライバへのネットワーク機能のサポートが追加されました。 (["GitHub問題#717"](#))
- ONTAP ドライバにTech Previewの自動MetroCluster スイッチオーバー検出機能を追加。 (["GitHub問題#228"](#))

非推奨

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

削除します

- AWS CVSドライバ（22.04以降で廃止）が削除されました。
- Kubernetes
 - ノードのポッドから不要なSYS_Admin機能を削除。
 - nodeprepをシンプルなホスト情報とアクティブなサービス検出に絞り込み、ベストエフォートを実現
ワーカーノードでNFS / iSCSIサービスを使用できることを確認します。

ドキュメント

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

2004年10月22日の変更

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



以前のTridentリリースからアップグレードしてAzure NetApp Files を使用している場合は、を参照してください location configパラメータは'必須のシングルトンフィールド'になりました

の修正

- iSCSI イニシエータ名の解析が改善されました。 ("[GitHub問題#681](#)")
- CSI ストレージクラスのパラメータが許可されていない問題 を修正しました。 ("[GitHub問題#598](#)")
- Trident CRD での重複キー宣言が修正されました。 ("[GitHub問題#671](#)")
- 不正確な CSI スナップショットログを修正しました。 ("[GitHub問題#629](#)") を選択します
- 削除したノードでボリュームを非公開にする問題 を修正しました。 ("[GitHub問題#691](#)")
- ブロックデバイスでのファイルシステムの不整合の処理が追加されました。 ("[GitHub問題#656](#)")
- を設定するときに、自動サポート画像をプルする固定問題 imageRegistry インストール中にフラグを付けます。 ("[GitHub問題#715](#)")
- ANF ドライバが複数のエクスポートルールでボリュームのクローニングに失敗した修正済み問題。

拡張機能

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

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

削除します

E シリーズドライバ (20.07 以降無効) が削除されました。

22.01.1 の変更

の修正

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

22.01.0での変更点

の修正

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

拡張機能

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

非推奨

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

21.10.1 の変更点



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

の修正

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

21.10.0 ノヘンコウテン

の修正

- XFS ボリュームのクローンをソースボリュームと同じノードにマウントできない固定問題（GitHub 問題 514）
- Astra Trident がシャットダウン時に致命的なエラーを記録した修正版問題（GitHub 問題 597）。
- Kubernetes 関連の修正：
 - を使用してSnapshotを作成する場合、リストアサイズの最小値としてボリュームの使用済みスペースを返します `ontap-nas` および `ontap-nas-flexgroup` ドライバ（GitHub問題 645）。
 - 問題を修正 Failed to expand filesystem ボリュームのサイズ変更後にエラーがログに記録されました（GitHub問題 560）。
 - ポッドが固定される問題を修正 Terminating 状態（GitHub問題 572）。
 - のケースを修正しました `ontap-san-economy FlexVol` はスナップショットLUNの一部である場合があります（GitHub問題 533）。
 - 異なるイメージを持つ固定カスタム YAML インストーラ問題（GitHub 問題 613）。
 - Snapshot サイズの計算方法を固定（GitHub 問題 611）。
 - 問題は修正され、Astra Trident のすべてのインストーラが OpenShift としてプレーン Kubernetes を識別できるようになりました（GitHub 問題 639）。
 - Kubernetes API サーバにアクセスできない場合に、Trident オペレータが更新を停止するよう修正しました（GitHub 問題 599）。

拡張機能

- のサポートが追加されました `unixPermissions` GCP - CVSパフォーマンスボリュームのオプション。
- GCP でのスケール最適化 CVS ボリュームのサポートが 600GiB から 1TiB に追加されました。
- Kubernetes 関連の機能拡張：
 - Kubernetes 1.22 のサポートが追加されました。
 - Trident の operator と Helm チャートを Kubernetes 1.22 （GitHub 問題 628）と連携させるように設定
 - に演算子の画像を追加しました `tridentctl` [画像] コマンド（GitHub問題 570）。

実験的な機能強化

- でボリュームレプリケーションのサポートが追加されました `ontap-san` ドライバ。
- のテクニカルプレビュー* RESTサポートを追加 `ontap-nas-flexgroup`、`ontap-san`` および ``ontap-nas-economy` ドライバ。

既知の問題

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

- Astra Tridentでは空白が強制されるようになりました `fsType` (`fsType=""`) を含むボリューム `fsType` ストレージクラスで指定されています。Tridentでは、Kubernetes 1.17以降を使用している場合は空白の入力がサポートされます `fsType` NFSボリューム。iSCSIボリュームの場合、を設定する必要があります `fsType` ストレージクラスで、を適用する場合 `fsGroup` セキュリティコンテキストの使用。
- 複数のAstra Tridentインスタンスでバックエンドを使用する場合は、各バックエンド構成ファイルに異なる値を設定する必要があります `storagePrefix` ONTAP バックエンドの値を指定するか、別のを使用します `TenantName` SolidFire バックエンドの場合：Astra Trident は、Astra Trident の他のインスタンスが作成したボリュームを検出できません。ONTAP または SolidFire バックエンドに既存のボリュームを作成しようとする成功します。Astra Trident は、ボリューム作成をべき等の操作として扱います。状況 `storagePrefix` または `TenantName` 同じバックエンドに作成されたボリュームでは名前が競合する可能性があるため、同じ名前を変更しないでください。
- Astra Tridentのインストール時（を使用 `tridentctl` またはTrident Operator）を使用し、を使用します `tridentctl` Astra Tridentを管理するには、が次の条件を満たしている必要があります `KUBECONFIG` 環境変数が設定されています。これは、Kubernetesクラスタにそれを示すために必要です `tridentctl` 対策を検討してください。複数のKubernetes環境を使用する場合は、を確認してください `KUBECONFIG` ファイルは正確に取得されます。
- iSCSI PVS のオンラインスペース再生を実行するには、作業ノード上の基盤となる OS がボリュームにマウントオプションを渡す必要があります。これは、が必要なRHEL / RedHat CoreOSインスタンスに該当します `discard "マウントオプション"`; `discard mountOption` がに含まれていることを確認します `[StorageClass^]`をクリックして、オンラインブロックの破棄をサポートします。
- Kubernetes クラスタごとに複数のAstra Trident インスタンスがある場合、Astra Trident は他のインスタンスと通信できず、作成した他のボリュームを検出できません。そのため、1つのクラスタ内で複数のインスタンスを実行している場合、予期しない動作が発生したり、誤ったりすることがあります。Kubernetes クラスタごとに Trident のインスタンスが1つだけ必要です。
- If Astra Tridentベース `StorageClass` TridentがオフラインのときにKubernetesからオブジェクトが削除

されると、対応するストレージクラスがオンラインに戻ってもTridentから削除されることはありません。これらのストレージクラスは、を使用して削除してください `tridentctl` またはREST API。

- 対応する PVC を削除する前に Astra Trident によってプロビジョニングされた PV を削除しても、Astra Trident は自動的に元のボリュームを削除しません。ボリュームは、から削除する必要があります `tridentctl` またはREST API。
- FlexGroup では、プロビジョニング要求ごとに一意のアグリゲートセットがないかぎり、同時に複数の ONTAP をプロビジョニングすることはできません。
- IPv6経由でAstra Tridentを使用する場合は、と指定する必要があります `managementLIF` および `dataLIF` バックエンドの定義を角かっこで囲みます。例：
`[fd20:8b1e:b258:2000:f816:3eff:feec:0]`。
- を使用する場合 `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"](#)

概念

Astra Trident の詳細をご確認ください

Astra Trident は、ネットアップがの一部として管理している、完全にサポートされているオープンソースプロジェクトです "[Astra 製品ファミリー](#)"。Container Storage Interface（CSI）などの業界標準のインターフェイスを使用して、コンテナ化されたアプリケーションの永続性に対する要求を満たすことができるように設計されています。

概要

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

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

サポートされる Kubernetes クラスタアーキテクチャ

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

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

アストラとは

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

Astra のページで無料トライアルに登録できます。

を参照してください。

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

ONTAP ドライバ

Astra Trident は、ONTAP クラスタとの通信に使用する 5 つの ONTAP ストレージドライバを提供します。各ドライバーがボリュームの作成、アクセス制御、機能をどのように処理するかについて、詳細をご覧ください。

ONTAP ストレージドライバについて説明します

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
ontap-nas	NFS	ファイルシステム	RWO 、 ROX 、 RWX	"" 、 NFS です
ontap-nas-economy	NFS	ファイルシステム	RWO 、 ROX 、 RWX	"" 、 NFS です
ontap-nas-flexgroup	NFS	ファイルシステム	RWO 、 ROX 、 RWX	"" 、 NFS です
ontap-san	iSCSI	ブロック	RWO 、 ROX 、 RWX	ファイルシステムなし。 rawブロックデバイスです
ontap-san	iSCSI	ファイルシステム	RWO 、 ROX rwxはファイルシステム ボリュームモードでは使用できません。	xfs、 ext3、 ext4
ontap-san-economy	iSCSI	ブロック	RWO 、 ROX 、 RWX	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-economy	iSCSI	ファイルシステム	RWO、ROX rwxはファイルシステムボリュームモードでは使用できません。	xfs、ext3、ext4



ONTAP バックエンドは、セキュリティロール（ユーザ名とパスワード）のログインクレデンシャル、またはONTAP クラスタにインストールされている秘密鍵と証明書を使用して認証できます。を使用して既存のバックエンドを更新し、認証モードを移行することができます

```
tridentctl update backend。
```

プロビジョニング

Trident の Astra プロビジョニングの主なフェーズは 2 つあります。最初のフェーズでは、ストレージクラスを適切なバックエンドストレージプールのセットに関連付け、プロビジョニング前の必要な準備として実行します。2 番目のフェーズでは、ボリュームの作成自体が行われます。このフェーズでは、保留中のボリュームのストレージクラスに関連付けられたストレージプールからストレージプールを選択する必要があります。

ストレージクラスの関連付け

バックエンドストレージプールをストレージクラスに関連付けるには、ストレージクラスの要求された属性とその両方が必要です `storagePools`、`additionalStoragePools` および `excludeStoragePools` リスト。ストレージクラスを作成すると、Trident はバックエンドごとに提供される属性とプールを、ストレージクラスから要求された属性とプールと比較します。要求された属性とプール名がストレージプールの属性と名前ですべて一致した場合、Astra Trident がそのストレージプールを、そのストレージクラスに適した一連のストレージプールに追加します。さらに、Trident の Astra では、にリストされているすべてのストレージプールが追加されます `additionalStoragePools` 属性がストレージクラスの要求した属性の一部または全部を満たしていない場合も、そのセットにリストされます。を使用する必要があります `excludeStoragePools` ストレージクラスに対して使用するストレージプールを上書きおよび削除するリスト。Astra Trident では、新しいバックエンドを追加するたびに同様のプロセスが実行され、ストレージプールが既存のストレージクラスのストレージクラスを満たしているかどうかを確認され、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ています。成功した場合は正常に返され、プロセスで発生したエラーが記録されます。Astra Trident は、要求されたストレージクラスとプロトコルで使用可能なすべてのストレージプールで * プロビジョニングに失敗した場合にのみ、障害 * を返し

ます。

ボリューム Snapshot

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

ボリュームSnapshotの作成方法について説明します

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

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

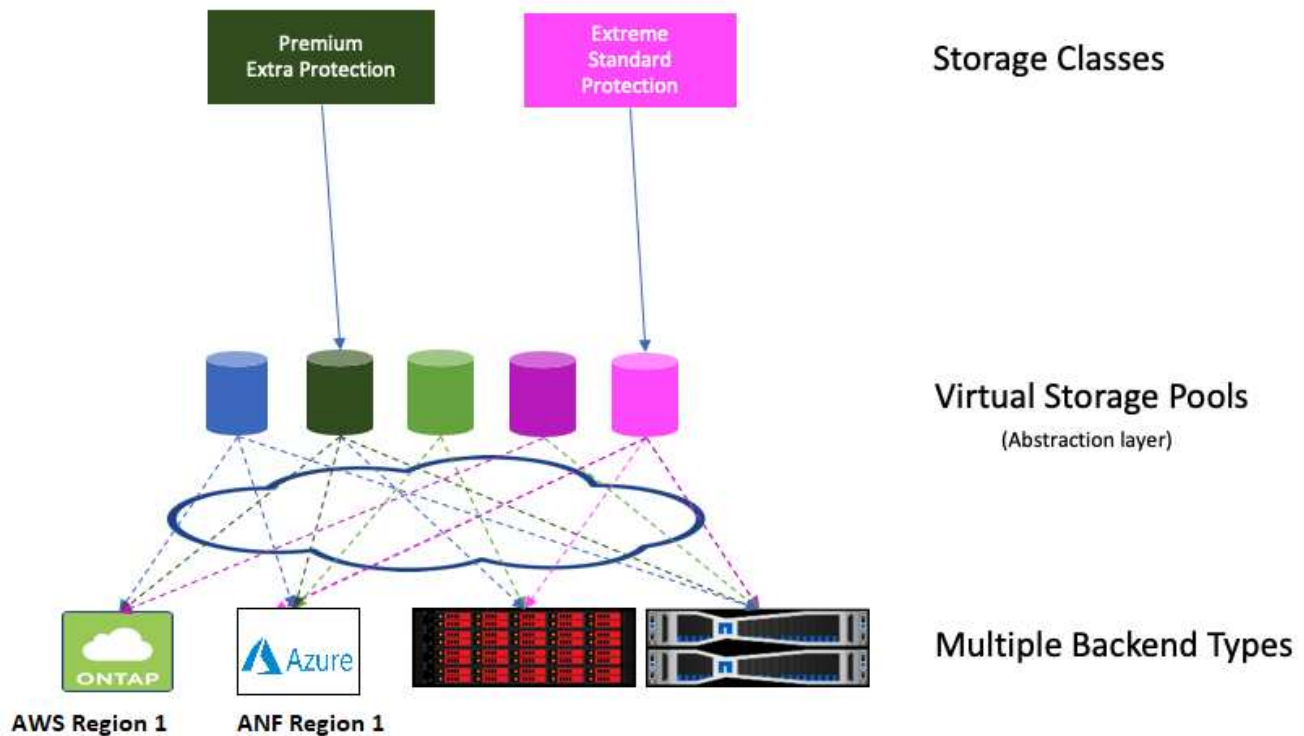
仮想ストレージプール

仮想ストレージプールは、Astra Trident のストレージバックエンドと Kubernetes の間に抽象化レイヤを提供 `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。

仮想ストレージ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ストレージプールを定義できます。



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



仮想ストレージプールを定義する場合は、バックエンド定義で既存の仮想プールの順序を変更しないでください。
また、既存の仮想プールの属性を編集または変更したり、新しい仮想プールを定義したりしないことを推奨します。

ほとんどの項目はバックエンド固有の用語で指定されます。アスペクト値は、バックエンドのドライバの外部には表示されず、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、ラベルは一意的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。

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

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

ボリュームアクセスグループ

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



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

ボリュームアクセスグループについて学習する

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

Trident が設定されたアクセスグループに新しいボリュームを関連付けても、アクセスグループ自体は作成も管理もされません。ストレージバックエンドが Astra Trident に追加される前に、アクセスグループが存在している必要があります。また、Kubernetes クラスタ内の、バックエンドでプロビジョニングされたボリュームをマウントできるすべてのノードの iSCSI IQN が含まれている必要があります。ほとんどのインストール環境では、クラスタ内のすべてのワーカーノードがこれに含まれます。

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

デフォルトを使用している構成から構成を変更する場合 trident 他のユーザも使用するアクセスグループには、のIDを追加します trident リスト内のアクセスグループ。

はじめに

ぜひお試しください

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

試乗について学びます

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

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



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

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

要件

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



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

Astra Tridentに関する重要な情報22.10

- 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 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

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1.9、1.10、1.11
- Kubernetes 1.20~1.25
- Mirantis Kubernetes Engine 3.5
- OpenShift 4.8、4.9、4.10、4.11

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1.9、1.10、1.11
- Kubernetes 1.20~1.25
- OpenShift 4.8、4.9、4.10、4.11

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

サポートされるバックエンド（ストレージ）

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

- NetApp ONTAP 対応の Amazon FSX
- Azure NetApp Files の特長
- Cloud Volumes ONTAP
- Cloud Volumes Service for GCP
- FAS/AFF / Select 9.3 以降
- ネットアップオール SAN アレイ（ASA）
- NetApp HCI / Elementソフトウェア11以降

機能の要件

次の表は、このリリースので使用できる機能をまとめたものです。
Astra Tridentと、それがサポートするKubernetesのバージョン

フィーチャー（Feature）	Kubernetes のバージョン	フィーチャーゲートが必要ですか？
CSI Trident	1.20-1.25	いいえ
ボリューム Snapshot	1.20-1.25	いいえ
ボリューム Snapshot からの PVC	1.20-1.25	いいえ
iSCSI PV のサイズ変更	1.20-1.25	いいえ
ONTAP 双方向 CHAP	1.20-1.25	いいえ

フィーチャー（Feature）	Kubernetes のバージョン	フィーチャーゲートが必要ですか？
動的エクスポートポリシー	1.20-1.25	いいえ
Trident のオペレータ	1.20-1.25	いいえ
自動ワーカーノード準備（ベータ版）	1.20-1.25	いいえ
CSI トポロジ	1.20-1.25	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はおりませんが、動作確認済みの特徴は次のとおりです。

- OpenShift Container Platform でサポートされている Red Hat CoreOS（RHCOS）バージョン
- RHELまたはCentOS 7.
- Ubuntu 18.04以降（最新22.04）
- Windows Server 2019

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する必要があります。

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

ホストの設定

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

ストレージシステムの構成：

Trident を使用するには、バックエンド構成でストレージシステムを使用する前に、一部の変更が必要になることがあります。を参照してください ["こちらをご覧ください"](#) を参照してください。

コンテナイメージと対応する Kubernetes バージョン

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

Kubernetes のバージョン	コンテナイメージ
v1.20.0	<ul style="list-style-type: none"> • docker.io/netapp/trident : 22.10.0 • docker.io / netapp/trident-autosupport : 22.10 • registry.k8s.io/sig-storage/csi-provisioner : v3.3.0 • registry.k8s.io/sig-storage/csi-attacher : v4.0.0 • registry.k8s.io/sig-storage/csi-resizer : v1.6.0 • registry.k8s.io/sig-storage/csi-snapshotter : v6.1.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar : v2.5.1 • docker.io/netapp/trident-operator : 22.10.0 (オプション)
v1.21.0	<ul style="list-style-type: none"> • docker.io/netapp/trident : 22.10.0 • docker.io / netapp/trident-autosupport : 22.10 • registry.k8s.io/sig-storage/csi-provisioner : v3.3.0 • registry.k8s.io/sig-storage/csi-attacher : v4.0.0 • registry.k8s.io/sig-storage/csi-resizer : v1.6.0 • registry.k8s.io/sig-storage/csi-snapshotter : v6.1.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar : v2.5.1 • docker.io/netapp/trident-operator : 22.10.0 (オプション)
v1.22.0	<ul style="list-style-type: none"> • docker.io/netapp/trident : 22.10.0 • docker.io / netapp/trident-autosupport : 22.10 • registry.k8s.io/sig-storage/csi-provisioner : v3.3.0 • registry.k8s.io/sig-storage/csi-attacher : v4.0.0 • registry.k8s.io/sig-storage/csi-resizer : v1.6.0 • registry.k8s.io/sig-storage/csi-snapshotter : v6.1.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar : v2.5.1 • docker.io/netapp/trident-operator : 22.10.0 (オプション)

Kubernetes のバージョン	コンテナイメージ
v1.3.0	<ul style="list-style-type: none"> • docker.io/netapp/trident : 22.10.0 • docker.io / netapp/trident-autosupport : 22.10 • registry.k8s.io/sig-storage/csi-provisioner : v3.3.0 • registry.k8s.io/sig-storage/csi-attacher : v4.0.0 • registry.k8s.io/sig-storage/csi-resizer : v1.6.0 • registry.k8s.io/sig-storage/csi-snapshotter : v6.1.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar : v2.5.1 • docker.io/netapp/trident-operator : 22.10.0 (オプション)
v1.24.0	<ul style="list-style-type: none"> • docker.io/netapp/trident : 22.10.0 • docker.io / netapp/trident-autosupport : 22.10 • registry.k8s.io/sig-storage/csi-provisioner : v3.3.0 • registry.k8s.io/sig-storage/csi-attacher : v4.0.0 • registry.k8s.io/sig-storage/csi-resizer : v1.6.0 • registry.k8s.io/sig-storage/csi-snapshotter : v6.1.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar : v2.5.1 • docker.io/netapp/trident-operator : 22.10.0 (オプション)
v1.25.0	<ul style="list-style-type: none"> • docker.io/netapp/trident : 22.10.0 • docker.io / netapp/trident-autosupport : 22.10 • registry.k8s.io/sig-storage/csi-provisioner : v3.3.0 • registry.k8s.io/sig-storage/csi-attacher : v4.0.0 • registry.k8s.io/sig-storage/csi-resizer : v1.6.0 • registry.k8s.io/sig-storage/csi-snapshotter : v6.1.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar : v2.5.1 • docker.io/netapp/trident-operator : 22.10.0 (オプション)



Kubernetesバージョン1.20以降では、検証済みを使用してください

registry.k8s.gcr.io/sig-storage/csi-snapshotter:v6.x イメージは、の場合にのみ作成します v1 のバージョンが処理しています

volumesnapshots.snapshot.storage.k8s.gcr.io CRD。状況に応じて v1beta1 バージョンは、の有無にかかわらず、CRDに対応しています v1 バージョン：検証済みを使用します registry.k8s.gcr.io/sig-storage/csi-snapshotter:v3.x イメージ (Image) :

導入の概要

Tridentのオペレータが、またはと連携してAstra Tridentを導入できます `tridentctl`。



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。

Astra Tridentに関する重要な情報22.10

- 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 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

導入方法を選択します

使用する導入方法を決定するには、次の点を考慮してください。

Tridentのオペレータが対応するタイミング

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

Trident オペレータには、以下に示すような利点があります。

自己回復機能

Trident の Astra インストールを監視し、導入が削除されたときや誤って変更された場合など、問題に対処する手段を積極的に講じることができます。オペレータが配置として設定されている場合は、「」を参照してください `trident-operator-<generated-id>` ポッドが作成されました。このポッドでは、を関連付けま

す TridentOrchestrator Astra TridentをインストールしたCRでは、常に1つのアクティブな状態が保証されます TridentOrchestrator。つまり、オペレータは、Astra Trident のインスタンスがクラスタ内に 1 つしかないことを確認し、セットアップを制御して、インストールがべきでないことを確認します。インストールに変更が加えられると（展開またはノードのデミスタなど）、オペレータはそれらを識別し、個別に修正します。

既存のインストール環境を簡単に更新できます

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

たとえば、Astra Trident を有効にしてデバッグログを生成する必要があるシナリオを考えてみましょう。

これを行うには、にパッチを適用します TridentOrchestrator をクリックして設定します spec.debug 終了: true:

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

実行後 TridentOrchestrator が更新され、オペレータが既存のインストールの更新とパッチを処理します。これにより、新しいポッドの作成がトリガーされ、それに応じてインストールが変更される場合があります。

Kubernetes のアップグレードを自動的に処理

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



クラスタがサポート対象外のバージョンにアップグレードされた場合、オペレータによって Astra Trident はインストールされません。Astra Trident がすでにオペレータとともにインストールされている場合、サポート対象外の Kubernetes バージョンに Astra Trident がインストールされていることを示す警告が表示されます。

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

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。

Helmを使用する状況

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

を使用する状況 tridentctl

既存の環境をにアップグレードする必要がある場合や、高度にカスタマイズする場合は、の使用を検討してください ["Tridentctl"](#)。これは、従来の方法であった Astra Trident を導入する方法です。

導入方法間での移動に関する考慮事項

導入方法を切り替える必要があるシナリオを想像するのは難しいことはありません。から移動する前に、次の点を考慮してください `tridentctl` オペレータベースの展開への展開、またはその逆の展開：

- Astra Trident のアンインストールには、常に同じ方法を使用します。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、削除して使用する場合 `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。

導入モードを理解する

Trident を導入する方法は 3 種類あります。

標準的な導入

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

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

このような標準的な導入は、次の 2 つの方法で実行できます。

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

このインストールモードは、Astra Trident をインストールする最も簡単な方法であり、ネットワークの制限を課すことのないほとんどの環境で機能します。

オフラインでの導入

エアギャップ展開を実行するには、を使用します `--image-registry` 呼び出し時にフラグを設定します `tridentctl install` をクリックして、プライベートイメージレジストリを指定します。Trident のオペレータを使用して導入する場合は、と指定することもできます `spec.imageRegistry` をクリックします `TridentOrchestrator`。このレジストリにはが含まれている必要があります ["Trident の画像"](#)、["Trident](#)

AutoSupport の画像"および CSI のサイドカーイメージ（ Kubernetes バージョンで必要な場合）

を使用して導入をカスタマイズできます `tridentctl` Tridentのリソースのマニフェストを生成します。導入、開始、サービスアカウント、Astra Trident がインストールの一部として作成するクラスターロールが含まれます。

導入環境のカスタマイズの詳細については、次のリンクを参照してください。

- ["オペレータベースの展開をカスタマイズします"](#)

*



プライベートイメージリポジトリを使用する場合は、を追加する必要があります `/sig-storage` プライベートレジストリURLの末尾に移動します。のプライベートレジストリを使用する場合 `tridentctl` は、を使用する必要があります `--trident-image` および `--autosupport-image` と組み合わせて使用します `--image-registry`。Tridentオペレータを使用してAstra Tridentを導入する場合は、Orchestrator CRに含まれていることを確認します `tridentImage` および `autosupportImage` をインストールパラメータに指定します。

リモート導入

次に、リモート導入プロセスの概要を示します。

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

その他の既知の設定オプション

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

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

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

Trident オペレータとともに導入

Tridentのオペレータが、Astra Tridentを導入できます。

Astra Tridentに関する重要な情報22.10

- 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 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

Tridentのオペレータ導入オプション

Tridentオペレータは、次のいずれかの方法で導入できます。

- Tridentの使用 "[Helmチャート](#)"：Helm ChartがTridentオペレータを導入し、Tridentをワンステップでインストールします。
- 手動：Trident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ファイルを提供します。
 - Kubernetes 1.24以前を実行しているクラスタの場合は、を使用します "[Bundle_pre_1_25.yaml](#)"。
 - Kubernetes 1.25以上を実行するクラスタの場合は、を使用します "[bundle_post_1_25.yaml](#)"。



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

前提条件を確認する

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

- サポートされているバージョンのKubernetesを実行している、サポートされているKubernetesクラスタに對するすべての権限が必要です。を確認します "[要件](#)"。
- サポートされているネットアップストレージシステムを利用できるようにしておきます。
- すべての Kubernetes ワーカーノードからボリュームをマウントできます。
- を搭載したLinuxホストがある `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。
- を設定しておきます `KUBECONFIG` Kubernetesクラスタ構成を参照する環境変数。
- を有効にしておきます "[Astra Trident に必要な機能ゲート](#)"。
- Kubernetes と Docker Enterprise を併用する場合は、 "[CLI へのアクセスを有効にする手順は、ユーザが行ってください](#)"。

それはすべてですか？ 最高！ それでは始めましょう。

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

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

必要なもの

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

- A "サポートされるKubernetesバージョン"
- Helm バージョン 3

手順

1. Trident の Helm リポジトリを追加：

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

2. を使用します `helm install` コマンドを使用し、導入環境の名前を指定します。
次の例を参照してください。

```
helm install <name> netapp-trident/trident-operator --version 22.10.0  
--create-namespace --namespace <trident-namespace>
```



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

インストール中に設定データを渡すには、次の 2 つの方法があります。

- `--values` (または `-f`):オーバーライドを使用して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` コマンドを実行します

```
helm install <name> netapp-trident/trident-operator --version 22.10.0  
--create-namespace --namespace --set tridentDebug=true
```

。 `values.yaml` File。Helmチャートの一部で、キーのリストとデフォルト値が表示されます。

`helm list` 名前、ネームスペース、グラフ、ステータス、アプリケーションのバージョン、リビジョン番号など。

Tridentオペレータを手動で導入し、Tridentをインストール

Trident のオペレータを手動で導入するには、以下の手順を実行します。

ステップ 1 : Kubernetes クラスタを確認する

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



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

Kubernetesのバージョンを確認するには、次のコマンドを実行します。

```
kubectl version
```

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

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

Docker Hub のイメージを使用してポッドを起動し、ポッドネットワーク経由でストレージシステムにアクセスできるかどうかを確認するには、次のコマンドを実行します。

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

手順 2 : オペレータをダウンロードして設定します



21.01 以降、Trident Operator はクラスタを対象とします。TridentのオペレータがTridentをインストールするには、を作成する必要があります `TridentOrchestrator` カスタムリソース定義 (CRD) およびその他のリソースの定義。Astra Trident をインストールする前に、次の手順を実行してオペレータをセットアップする必要があります。

1. からTridentインストーラバンドルの最新バージョン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/v22.10.0/trident-
installer-22.10.0.tar.gz
tar -xf trident-installer-22.10.0.tar.gz
cd trident-installer
```

2. 適切なCRDマニフェストを使用して、を作成します `TridentOrchestrator` CRD。次に、を作成します `TridentOrchestrator` 後でカスタムリソース (Custom Resource) をクリックして、演算子によってインストールをインスタンス化する。

次のコマンドを実行します。

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

3. のあとに入力します TridentOrchestrator CRDが作成され、オペレータの展開に必要な次のリソースを作成します。

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

Trident インストーラには、これらのリソースを定義するマニフェストが含まれています。デフォルトでは、オペレータはに配置されます `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。次のマニフェストを使用してネームスペースを作成してください。

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

4. デフォルト以外の名前空間に演算子を配置します `trident` ネームスペースの場合はを更新する必要があります `serviceaccount.yaml`、`clusterrolebinding.yaml` および `operator.yaml` マニフェストを作成し、を生成します `bundle.yaml`。

次のコマンドを実行してYAMLマニフェストを更新し、を生成します `bundle.yaml` を使用する `kustomization.yaml`：

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

次のコマンドを実行してリソースを作成し、オペレータを配置します。

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

5. 展開後にオペレータのステータスを確認するには、次の手順を実行します。

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

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
trident-operator	1/1	1	1	3m

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

NAME	READY	STATUS	RESTARTS
AGE			
trident-operator-54cb664d-lnjxh	1/1	Running	0
3m			

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



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

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

これで、オペレータを使って Astra Trident をインストールする準備ができました。これには作成が必要です TridentOrchestrator。Tridentのインストーラには、作成用の定義例が付属しています TridentOrchestrator。これがの設置作業から始まります trident ネームスペース：


```

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:22.10
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:          true
    Image Pull Secrets:
    Image Registry:
    k8sTimeout:     30
    Kubelet Dir:    /var/lib/kubelet
    Log Format:     text
    Silence Autosupport:  false
    Trident Image:  netapp/trident:21.04.0
  Message:          Trident installed Namespace:
trident
  Status:           Installed
  Version:          v21.04.0
Events:
  Type Reason Age From Message ---- -
Installing 74s trident-operator.netapp.io Installing Trident Normal
Installed 67s trident-operator.netapp.io Trident installed

```

Tridentオペレータは、の属性を使用して、Astra Tridentのインストール方法をカスタマイズできます
TridentOrchestrator 仕様を参照してください "[Trident の導入をカスタマイズ](#)".

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。

ステータス	説明
インストール中です	このツールを使用してAstra Tridentをインストールしている TridentOrchestrator CR。
インストール済み	Astra Trident のインストールが完了しました。
アンインストール中です	OperatorはAstra Tridentをアンインストールしています。理由はです spec.uninstall=true。
アンインストール済み	Astra Trident がアンインストールされました。
失敗しました	オペレータがインストール、パッチ適用、アップデート、またはアンインストールできませんでした Astra Trident。オペレータはこの状態からのリカバリを自動的に試行します。この状態が解消されない場合は、トラブルシューティングが必要です。
更新中です	オペレータが既存のインストールを更新しています。
エラー	。TridentOrchestrator は使用されません。もう一つ存在します。

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。オペレータのログを確認する必要があります。を参照してください ["トラブルシューティング"](#) セクション。

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

```
kubectl get pod -n trident
```

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

を使用することもできます 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 |
+-----+-----+
| 21.04.0        | 21.04.0        |
+-----+-----+
```

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



導入時の問題のトラブルシューティングについては、を参照してください ["トラブルシューティング"](#) セクション。

Trident オペレータの環境をカスタマイズ

Tridentオペレータは、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 arguments allow`、の使用を検討する必要があります `tridentctl` 必要に応じて変更できるカスタムYAMLマニフェストを生成します。



`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示します。このパラメータ * は、Astra Trident のインストール後に更新できません *。これを実行すると、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。

設定オプション

このテーブルの詳細 `TridentOrchestrator` 属性：

パラメータ	説明	デフォルト
<code>namespace</code>	Astra Trident をインストールするネームスペース	デフォルト
<code>debug</code>	Astra Trident のデバッグを有効にします	いいえ
<code>windows</code>	をに設定します <code>true</code> Windowsワーカーノードへのインストールを有効にします。	いいえ
<code>IPv6</code>	IPv6 経由の Astra Trident をインストール	いいえ
<code>k8sTimeout</code>	Kubernetes 処理のタイムアウト	30 秒
<code>silenceAutosupport</code>	AutoSupportバンドルをNetAppに送信しない 自動	いいえ
<code>enableNodePrep</code>	ワーカーノードの依存関係を自動的に管理 (* beta *)	いいえ
<code>autosupportImage</code>	AutoSupport テレメトリのコンテナイメージ	「NetApp/trident-autosupport : 22.10.0」
<code>autosupportProxy</code>	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"

パラメータ	説明	デフォルト
uninstall	Astra Trident のアンインストールに使用するフラグ	いいえ
logFormat	Astra Trident のログ形式が使用 [text、JSON]	テキスト (Text)
tridentImage	インストールする Astra Trident イメージ	「NetApp / Trident : 21.04」
imageRegistry	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	ホスト上の kubelet ディレクトリへのパス	「/var/lib/kubelet」
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	
controllerPluginNodeSelector	Trident Controller CSI プラグインを実行しているポッドの追加ノードセクタ。 pod.spec.nodeSelector と同じ形式を使用します。	デフォルトはありません。オプションです
controllerPluginTolerations	Trident Controller CSI プラグインを実行しているポッドに対する許容値を上書きします。POD .spec.Tolerations と同じ形式を使用します。	デフォルトはありません。オプションです
nodePluginNodeSelector	Trident ノード CSI プラグインを実行しているポッドの追加ノードセクタ。pod.spec.nodeSelector と同じ形式を使用します。	デフォルトはありません。オプションです
nodePluginTolerations	Trident Node CSI プラグインを実行しているポッドに対する許容値を上書きします。POD .spec.Tolerations と同じ形式を使用します。	デフォルトはありません。オプションです



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

構成例

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

例1：基本的なカスタム構成

次に、基本的なカスタム構成の例を示します。

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

例2：ノードセクタを使用して導入します

次の例では、ノードセクタを使用してTridentを導入する方法を示します。

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

例3：Windowsワーカーノードに導入する

この例は、Windowsワーカーノードへの導入を示しています。

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

tridentctl を使用して導入します

を使用して、Astra Tridentを導入できます `tridentctl`。をよく理解しておくことをお勧めします ["基本概念"](#)。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22.10

- 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 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

前提条件を確認する

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

- サポート対象のKubernetesクラスタに対するすべての権限。
- 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 環境変数は、Kubernetes クラスターの構成を指します。
- "Astra Trident に必要な機能ゲート" が有効になります。
- 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"。

ステップ 1 : Kubernetes クラスターを確認する

Linux ホストにログインし、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 クラスター" また、必要な権限があります。



OpenShift で、使用できます `oc` ではなく `kubectl` 以降に示すすべての例では、実行して、最初に `* system:admin *` としてログインする必要があります `oc login -u system:admin` または `oc login -u kube-admin`。

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

```
kubectl version
```

Kubernetes クラスター管理者の権限を確認するには、次のコマンドを実行します。

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

Docker Hub のイメージを使用してポッドを起動し、ポッドネットワーク経由でストレージシステムにアクセスできるかどうかを確認するには、次のコマンドを実行します。

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

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

手順 2 : インストーラをダウンロードして展開します



Trident インストーラは Trident ポッドを作成し、そのステートを維持するために使用される CRD オブジェクトを構成し、プロビジョニングやクラスターホストへのボリュームの接続などのアクションを実行する CSI サイドカーを初期化します。

Trident インストーラバンドルの最新バージョンは、からダウンロードして展開できます "GitHub の [_Assets_section](#) を参照してください"。

たとえば、最新バージョンが 22.10.0 の場合、次のようになります。

```
wget https://github.com/NetApp/trident/releases/download/v22.10.0/trident-
installer-22.10.0.tar.gz
tar -xf trident-installer-22.10.0.tar.gz
cd trident-installer
```

手順 3 : 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

```
./tridentctl install -n trident
....
INFO Starting Trident installation.                namespace=trident
INFO Created service account.
INFO Created cluster role.
INFO Created cluster role binding.
INFO Added finalizers to custom resource definitions.
INFO Created Trident service.
INFO Created Trident secret.
INFO Created Trident deployment.
INFO Created Trident daemonset.
INFO Waiting for Trident pod to start.
INFO Trident pod started.                          namespace=trident
pod=trident-csi-679648bd45-cv2mx
INFO Waiting for Trident REST interface.
INFO Trident REST interface is up.                 version=22.10.0
INFO Trident installation succeeded.
....
```



WindowsノードでAstra Tridentを実行できるようにするには、を追加します --windows インストールコマンドへのフラグ：\$./tridentctl install --windows -n trident。

インストーラが完了すると、次のような出力が表示されます。Kubernetesクラスタ内のノードの数によっては、ポッドがさらに存在する場合があります。


```
kubectl get pod -n trident
```

NAME	READY	STATUS	RESTARTS	AGE
trident-csi-679648bd45-cv2mx	4/4	Running	0	5m29s
trident-csi-vgc8n	2/2	Running	0	5m29s


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

SERVER VERSION	CLIENT VERSION
22.10.0	22.10.0

Astra Tridentの設定を完了するには、に進みます ["導入後のタスク"](#)。

インストーラが正常に完了しない場合、または `trident-csi-<generated id>` ステータス* `RUNNING` *がなく、プラットフォームがインストールされていません。



導入時の問題のトラブルシューティングについては、を参照してください ["トラブルシューティング"](#)。

tridentctl 展開をカスタマイズします

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

インストーラの詳細を確認してください

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

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

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

- `trident-clusterrolebinding.yaml`
- `trident-deployment.yaml`
- `trident-crds.yaml`
- `trident-clusterrole.yaml`
- `trident-daemonset.yaml`
- `trident-service.yaml`

- trident-namespace.yaml
- trident-serviceaccount.yaml
- trident-resourcequota.yaml

これらのファイルを生成したら、必要に応じて変更し、を使用できます `--use-custom-yaml` をクリックして、カスタム導入環境をインストールします。

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

次の手順

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

手順 1：バックエンドを作成する

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

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

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

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

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

```
./tridentctl -n trident logs
```

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

手順 2：ストレージクラスを作成する

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

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

をベースにしたストレージクラスが最もシンプルになりました sample-input/storage-class-csi.yaml.template インストーラに付属のファイル `BACKEND_TYPE` ストレージドライバの名前を指定します。

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

cp sample-input/storage-class-csi.yaml.template sample-input/storage-class-
basic-csi.yaml

# Modify __BACKEND_TYPE__ with the storage driver field above (e.g.,
ontap-nas)
vi sample-input/storage-class-basic-csi.yaml
```

これはKubernetesオブジェクトなので、を使用します `kubectl` をクリックしてKubernetesで作成します。

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

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

```

kubectl get sc basic-csi
NAME          PROVISIONER          AGE
basic-csi     csi.trident.netapp.io 15h

./tridentctl -n trident get storageclass basic-csi -o json
{
  "items": [
    {
      "Config": {
        "version": "1",
        "name": "basic-csi",
        "attributes": {
          "backendType": "ontap-nas"
        },
        "storagePools": null,
        "additionalStoragePools": null
      },
      "storage": {
        "ontapnas_10.0.0.1": [
          "aggr1",
          "aggr2",
          "aggr3",
          "aggr4"
        ]
      }
    }
  ]
}

```

手順 3：最初のボリュームをプロビジョニングします

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

作成したストレージクラスを使用するボリュームの PVC を作成します。

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

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

```
kubectl get pvc --watch
```

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

手順 4：ボリュームをポッドにマウントする

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

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

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

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

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

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

ボリュームを削除するには、要求を削除します。

```
kubectl delete pvc basic
```

これで、次のような追加タスクを実行できます。

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

Trident で Astra を管理

Astra Trident をアップグレード

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



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

アップグレード先のバージョンを確認します

- にアップグレードできます YY.MM からリリースします YY-1.MM リリースとリリース間の関係。たとえば、19.07 以降（19.07.1 などのドットリリースを含む）から 20.07 への直接アップグレードを実行できます。
- 以前のリリースを使用している場合は、複数の手順からなるアップグレードを実行する必要があります。そのためには、最初に 4 つのリリースウィンドウに対応する最新リリースにアップグレードする必要があります。たとえば '18.07 を実行していて '20.07 リリースにアップグレードする場合は '次' に示すように '複数ステップのアップグレード・プロセスを実行します
 - 最初のアップグレードは 18.07 から 19.07 へ。特定のアップグレード手順については、該当するリリースのドキュメントを参照してください。
 - その後 '19.07 から 20.07 にアップグレードします



バージョン19.04以前のアップグレードでは、Astra Trident独自のメタデータを移行する必要があります etcd をCRDオブジェクトに追加します。アップグレードの仕組みについては、リリースのドキュメントを確認してください。



アップグレードするときは、この作業を行うことが重要です parameter.fsType インチ StorageClasses Astra Tridentが使用。削除して再作成することができます StorageClasses 実行前のボリュームの中断はなし。これは、強制的 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。。 "[入力例](#)：" directory には、などの例があります [storage-class-basic.yaml.template]および[storage-class-bronze-default.yaml]をクリックします。詳細については、を参照してください "[既知の問題](#)"。

どのアップグレードパスを選択すればよいですか？

次のいずれかのパスを使用してアップグレードできます。

- Trident 演算子を使用する。
- を使用します tridentctl。



CSI のボリュームスナップショットは、Kubernetes 1.20 以降の GA 機能になりました。Astra Trident をアップグレードする場合、アップグレードを実行する前に、以前のスナップショット CRS と CRD（ボリューム Snapshot クラス、ボリューム Snapshot、ボリューム Snapshot コンテンツ）をすべて削除する必要があります。を参照してください ["この blog"](#) アルファスナップショットを beta/GA 仕様に移行する手順を理解する。

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

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



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

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

演算子に変更があります

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

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

詳細については、こちらをご覧ください

- ["Trident オペレータを使用してアップグレード"](#)
*

オペレータにアップグレードしてください

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

必要なもの

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

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



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

クラスタを対象としたオペレータ環境をアップグレードします

次の手順に従って、* Trident 21.01以降*からアップグレードします。

手順

1. 現在の Astra Trident インスタンスのインストールに使用した Trident オペレータを削除たとえば、21.01 からアップグレードする場合は、次のコマンドを実行します。

```
kubectl delete -f 22.01/trident-installer/deploy/BUNDLE.YAML -n trident
```

2. 必要に応じて、を編集できます `TridentOrchestrator` Tridentのインストール時に作成したオブジェクトで、インストールパラメータを変更します。これには、カスタムTridentイメージの変更、コンテナイメージをプルするためのプライベートイメージレジストリ、デバッグログの有効化、イメージプルシークレットの指定などの変更が含まれます。
3. 環境に適したバンドルYAMLファイルとからAstra Tridentバージョンを使用して、Astra Tridentをインストールします <https://github.com/NetApp/trident/tree/stable/vXX.XX> /`deploy/BUNDLE.YAML` ここで、`vXX.XX` は、バージョン番号です（例 `v22.10`）および `BUNDLE.YAML` はバンドルYAMLファイル名です。



- Kubernetes 1.24以前を実行しているクラスタの場合は、を使用します ["Bundle_pre_1_25.yaml"](#)。
- Kubernetes 1.25以上を実行するクラスタの場合は、を使用します ["bundle_post_1_25.yaml"](#)。

たとえば、Kubernetes 1.25用にAstra Trident 22.10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 22.10.0/trident-installer/deploy/bundle_post_1_25.yaml  
-n trident
```

この手順の一環として、Tridentオペレータが既存のAstra Tridentインストールを特定し、オペレータと同じバージョンにアップグレードします。

名前空間を対象としたオペレータインストールをアップグレードします

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

手順

1. 既存の Trident インストールのステータスを確認そのためには、の*ステータス*を確認してください TridentProvisioner。ステータスがになっている必要があります Installed。

```
kubectl describe tprov trident -n trident | grep Message: -A 3  
Message:  Trident installed  
Status:    Installed  
Version:   v20.10.1
```



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

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

```
# Download the release required [22.10.0]
mkdir 22.10.0
cd 22.10.0
wget
https://github.com/NetApp/trident/releases/download/v22.10.0/trident-
installer-22.10.0.tar.gz
tar -xf trident-installer-22.10.0.tar.gz
cd trident-installer
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
```

3. マニフェストを使用して、名前空間を対象とした演算子を削除します。この手順を完了するには、名前空間を対象とした演算子をから配備するために使用するバンドルYAMLファイルが必要です

<https://github.com/NetApp/trident/tree/stable/vXX.XX/deploy/BUNDLE.YAML> ここで、`vXX.XX` は、バージョン番号です（例 `v22.10`） および `BUNDLE.YAML` はバンドルYAMLファイル名です。



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

`tridentImage`、`autosupportImage`、プライベートイメージリポジトリ、および提供 `imagePullSecrets`) 名前空間を対象とした演算子を削除した後、クラスタを対象とした演算子をインストールする前。更新可能なパラメータの一覧については、を参照してください ["設定オプション"](#)。

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

#Delete the namespace-scoped operator
kubectl delete -f deploy/<BUNDLE.YAML>
serviceaccount "trident-operator" deleted
clusterrole.rbac.authorization.k8s.io "trident-operator" deleted
clusterrolebinding.rbac.authorization.k8s.io "trident-operator" deleted
deployment.apps "trident-operator" deleted
podsecuritypolicy.policy "tridentoperatorpods" deleted

#Confirm the Trident operator was removed
kubectl get all -n trident
```

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

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

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

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

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

この段階では、を実行します trident-operator-xxxxxxxxxx-xxxxxx ポッドが削除されました。

4. (オプション) インストールパラメータを変更する必要がある場合は、を更新します
TridentProvisioner 仕様これらの変更には、コンテナイメージをからプルするためのプライベートイメージレジストリの変更、デバッグログの有効化、イメージプルシークレットの指定などがあります。

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

5. クラスタを対象とした演算子をインストールします。



クラスタを対象としたオペレータをインストールすると、の移行が開始されます
TridentProvisioner オブジェクトの移動先 TridentOrchestrator オブジェクトを
削除します TridentProvisioner オブジェクトと tridentprovisioner CRD、およ
びAstra Tridentを、使用しているクラスタ対象オペレータのバージョンにアップグレードし
ます。次の例では、Tridentを22.10.0にアップグレードしています。



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

```
#Ensure you are in the correct directory
pwd
/root/22.10.0/trident-installer

#Install the cluster-scoped operator in the **same namespace**
kubectl create -f deploy/<BUNDLE.YAML>
serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created

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

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

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

#Confirm Trident has been updated to the desired version
kubectl describe torc trident | grep Message -A 3
Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v22.10.0
```

Helm ベースのオペレータインストールをアップグレードします

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

手順

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

```
helm upgrade <name> trident-operator-22.10.0.tgz
```

ここで、trident-operator-22.10.0.tgz アップグレード後のバージョンが反映されます。

3. を実行します helm list グラフとアプリのバージョンが両方ともアップグレードされていることを確認します。



アップグレード中に構成データを渡すには、を使用します --set。

たとえば、のデフォルト値を変更するには、のように指定します `tridentDebug` を使用して、次のコマンドを実行します。

```
helm upgrade <name> trident-operator-22.10.0-custom.tgz --set  
tridentDebug=true
```

を実行した場合 `tridentctl logs` デバッグメッセージが表示されます。



初期インストール時にデフォルト以外のオプションを設定する場合は、オプションが upgrade コマンドに含まれていることを確認してください。含まれていない場合は、値がデフォルトにリセットされます。

オペレータ以外のインストールからアップグレードします

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

手順

1. 最新の Astra Trident リリースをダウンロード

```
# Download the release required [22.10.0]  
mkdir 22.10.0  
cd 22.10.0  
wget  
https://github.com/NetApp/trident/releases/download/v22.10.0/trident-  
installer-22.10.0.tar.gz  
tar -xf trident-installer-22.10.0.tar.gz  
cd trident-installer
```

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

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

3. オペレータを配備します。

```
#Install the cluster-scoped operator in the **same namespace**
kubectl create -f deploy/<BUNDLE.YAML>
serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created

#Examine the pods in the Trident namespace
```

NAME	READY	STATUS	RESTARTS	AGE
trident-csi-79df798bdc-m79dc	6/6	Running	0	150d
trident-csi-xrst8	2/2	Running	0	150d
trident-operator-5574dbbc68-nthjv	1/1	Running	0	1m30s

4. を作成します TridentOrchestrator Astra Tridentのインストール用にCR。

```
#Create a tridentOrchestrator to initiate a Trident install
cat deploy/crds/tridentorchestrator_cr.yaml
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident

kubectl create -f deploy/crds/tridentorchestrator_cr.yaml

#Examine the pods in the Trident namespace
```

NAME	READY	STATUS	RESTARTS	AGE
trident-csi-79df798bdc-m79dc	6/6	Running	0	1m
trident-csi-xrst8	2/2	Running	0	1m
trident-operator-5574dbbc68-nthjv	1/1	Running	0	5m41s

```
#Confirm Trident was upgraded to the desired version
kubectl describe torc trident | grep Message -A 3
Message:          Trident installed
Namespace:        trident
Status:           Installed
Version:          v22.10.0
```

既存のバックエンドと PVC は自動的に使用可能

tridentctl を使用してアップグレードします

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

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。

- Trident 20.01 以降では、のベータ版のみが提供されます **"ボリューム Snapshot"** がサポートされます。Kubernetes 管理者は、従来のアルファスナップショットを保持するために、アルファスナップショットオブジェクトを安全にバックアップするか、ベータ版に変換するように注意する必要があります。
- ボリュームスナップショットのベータリリースでは、一連の新しい CRD とスナップショットコントローラが導入されています。どちらも Astra Trident をインストールする前にセットアップする必要があります。 **"この blog"** alpha ボリュームの Snapshot をベータ版に移行する手順について説明します。
- Astra Trident のアンインストールと再インストールはアップグレードとして機能します。Trident をアンインストールしても、Astra Trident 環境で使用されている Persistent Volume Claim （PVC；永続的ボリューム要求）と Persistent Volume （PV；永続的ボリューム）は削除されません。Astra Trident がオフラインの間は、すでにプロビジョニング済みの PVS を引き続き使用でき、Astra Trident は、オンラインに戻った時点で作成された PVC に対してボリュームをプロビジョニングします。



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

アップグレード後の次の手順

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

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

CSI タイプにボリュームをアップグレードする場合は、次の点を考慮してください。

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

ボリュームのアップグレードの例

次の例は、ボリュームのアップグレードを実行する方法を示しています。

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

```
kubectl get pv
```

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

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

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

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

Name: default-pvc-2-a8486

Labels: <none>

Annotations: pv.kubernetes.io/provisioned-by: netapp.io/trident
volume.beta.kubernetes.io/storage-class: standard

Finalizers: [kubernetes.io/pv-protection]

StorageClass: standard

Status: Bound

Claim: default/pvc-2

Reclaim Policy: Delete

Access Modes: RWO

VolumeMode: Filesystem

Capacity: 1073741824

Node Affinity: <none>

Message:

Source:

Type: NFS (an NFS mount that lasts the lifetime of a pod)

Server: 10.xx.xx.xx

Path: /trid_1907_alpha_default_pvc_2_a8486

ReadOnly: false

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

プグレードする必要があります。

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

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

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

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

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

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

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

Astra Trident をアンインストール

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

Helm を使用してアンインストールします

Helmを使用してAstra Tridentをインストールした場合は、を使用してアンインストールできます `helm uninstall`。

```
#List the Helm release corresponding to the Astra Trident install.
helm ls -n trident
NAME                NAMESPACE          REVISION          UPDATED
STATUS              CHART                APP VERSION
trident             trident             1                2021-04-20
00:26:42.417764794 +0000 UTC deployed    trident-operator-21.07.1
21.07.1

#Uninstall Helm release to remove Trident
helm uninstall trident -n trident
release "trident" uninstalled
```

Trident オペレータを使用してをアンインストールします

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

- **編集 TridentOrchestrator** アンインストールフラグを設定するには：を編集できます
TridentOrchestrator をクリックして設定します spec.uninstall=true。を編集します
TridentOrchestrator CRおよびを設定します uninstall 次のようなフラグを設定します。

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

をクリックします uninstall フラグはに設定されています `true` は、TridentオペレータがTridentをアンインストールしますが、TridentOrchestrator自体は削除されません。必要に応じて、TridentOrchestratorをクリーンアップし、新しいTridentOrchestratorを作成する必要があります。
Tridentをもう一度インストールします。

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

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

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



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

を使用してをアンインストールします `tridentctl`

を実行します `uninstall` のコマンド `tridentctl` 次のように、Astra Tridentに関連付けられているすべてのリソースを削除します。ただし、CRDと関連オブジェクトは削除されます。そのため、インストーラを再実行して、より新しいバージョンに簡単に更新できます。

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

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

Trident をダウングレード

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

ダウングレードするタイミング

次のような理由でダウングレードを検討してください。

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

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

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

ダウングレードしない場合

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

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

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

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

ドできます。

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

クラスタを対象とした演算子にダウングレードします

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

手順

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

名前空間を対象とした演算子にダウングレードします

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

手順

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

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

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

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

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

```
kubectl get crd tridentorchestrators.trident.netapp.io
```

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

```
kubectl delete crd tridentorchestrators.trident.netapp.io
```

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

Astra Trident がアンインストールされました。

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

Helm を使用してダウングレードしてください

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

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

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

tridentctl

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



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

手順

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

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

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

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

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

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

ダウングレードプロセスが完了します。

Astra Trident を使用

ワーカーノードを準備します

Kubernetes クラスタ内のすべてのワーカーノードが、ポッド用にプロビジョニングしたボリュームをマウントできる必要があります。を使用する場合 `ontap-nas`、`ontap-nas-economy` または `ontap-nas-flexgroup` バックエンドの1つを推進するのに、ワーカーノードにはNFSツールが必要です。それ以外の場合は iSCSI ツールが必要です。

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



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

ノードサービスの検出

22.07以降、Astra Tridentは、ノードでiSCSIサービスまたはNFSサービスを実行できるかどうかを自動的に検出しようとします。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、コマンドを使用します。

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

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も特定します。検出されたサービスを表示するには、次のコマンドを使用します。

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



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ん。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。

NFS ボリューム

プロトコル	オペレーティングシステム	コマンド
NFS	RHEL / CentOS 7.	<code>sudo yum install -y nfs-utils</code>
NFS	Ubuntu	<code>sudo apt-get install -y nfs-common</code>



ブート時に NFS サービスが開始されていることを確認してください。

iSCSI ボリューム

iSCSI ボリュームを使用するときは、次の点に注意してください。

- Kubernetes クラスタ内の各ノードには一意の IQN を割り当てる必要があります。* これは必須の前提条件です*。
- RHCOSバージョン4.5以降またはRHEL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、を指定してください `discard StorageClass`の`mountOption`を使用して、インラインのスペース再生を実行します。を参照してください ["RedHat のマニュアル"](#)。

プロトコル	オペレーティングシステム	コマンド
iSCSI	RHEL/CentOS	<ol style="list-style-type: none"> 次のシステムパッケージをインストールします。 <pre>sudo yum install -y lsscsi iscsi-initiator- utils sg3_utils device- mapper-multipath</pre> 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。 <pre>rpm -q iscsi-initiator- utils</pre> スキャンを手動に設定： <pre>sudo sed -i 's/^\(node.session.scan \).*\/\1 = manual/' /etc/iscsi/iscsid.conf</pre> マルチパスを有効化： <pre>sudo mpathconf --enable --with_multipathd y --find_multipaths n</pre> <div>  <p>確認します etc/multipath.conf が含まれます find_multipaths no の下 defaults。</p> </div> を確認します iscsid および multipathd 実行中： <pre>sudo systemctl enable --now iscsid multipathd</pre> を有効にして開始します iscsi： <pre>sudo systemctl enable --now iscsi</pre>

プロトコル	オペレーティングシステム	コマンド
iSCSI	Ubuntu	<ol style="list-style-type: none"> 次のシステムパッケージをインストールします。 <pre>sudo apt-get install -y open-iscsi lsscsi sg3- utils multipath-tools scsitools</pre> 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。 <pre>dpkg -l open-iscsi</pre> スキャンを手動に設定： <pre>sudo sed -i 's/^\(node.session.scan \).*\/\1 = manual/' /etc/iscsi/iscsid.conf</pre> マルチパスを有効化： <pre>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</pre> <div>  <div> <p>確認します etc/multipath.conf が含まれます find_multipaths no の下 defaults。</p> </div> </div> を確認します open-iscsi および multipath-tools 有効になっていて実行中：
		<pre>sudo systemctl status multipath-tools</pre>



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

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

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性を定義します。Trident がストレージシステムとの通信方法を Trident から指示し、Astra Trident がボリュームをプロビジョニングする方法も解説します。Astra Trident は、ストレージクラスが定義した要件に合わせて、バックエンドからストレージプールを自動的に提供します。お使いのストレージシステムのタイプに基づいたバックエンドの設定の詳細については、こちらを参照してください。

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

Azure NetApp Files バックエンドを設定します

Azure NetApp Files (ANF) を Astra Tridentのバックエンドとして設定できます。ANFバックエンドを使用してNASボリュームとSMBボリュームを接続できます。

- ["準備"](#)
- ["設定オプションと例"](#)

考慮事項

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

Azure NetApp Files バックエンドを設定する準備をします

ANF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。

Azure NetApp Files を初めて使用する場合や新しい場所で使用する場合は、いくつかの初期設定が必要です。

- Azure NetApp Files をセットアップしてNFSボリュームを作成する方法については、[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"](#)。
- Azure NetApp Files を設定してSMBボリュームを追加するには、以下を参照してください。 ["Azure : Azure NetApp Files 用のSMBボリュームを作成します"](#)。

要件

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が必要です。

- subscriptionID Azure NetApp Files を有効にしたAzureサブスクリプションから選択します。
- tenantID、clientID`および `clientSecret から ["アプリケーション登録"](#) Azure Active Directory で、Azure NetApp Files サービスに対する十分な権限がある。アプリケーション登録では、次のいずれかを使用します。
 - オーナーまたは寄与者のロール ["Azureで事前定義"](#)
 - A ["カスタム投稿者ロール"](#) をサブスクリプションレベルで選択します (assignableScopes)以下のアクセス許可は、Astra Tridentが必要とするものに限定されます。カスタムロールを作成したあと、["Azureポータルを使用してロールを割り当てます"](#)。

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

```

```

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

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

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

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/subvolumes/GetMetadata/action",

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

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

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

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

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

                                "Microsoft.Features/features/read",
                                "Microsoft.Features/operations/read",
                                "Microsoft.Features/providers/features/read",

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

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

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

```

- Azureがサポートされます location を1つ以上含むデータセンターを展開します **"委任されたサブネット"**。Trident 22.01の時点では location パラメータは、バックエンド構成ファイルの最上位にある必須フィールドです。仮想プールで指定された場所の値は無視されます。

SMBボリュームに関するその他の要件

- 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クラスター。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Tridentシークレットが少なくとも1つあり、ANFがActive Directoryに認証できるようになっている。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='pw'
```

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

Azure NetApp Files バックエンド構成のオプションと例

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

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



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

バックエンド構成オプション

ANFバックエンドには次の設定オプションがあります。

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

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

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



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

必要な権限とリソース

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

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

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

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

ボリュームのプロビジョニング

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

パラメータ	説明	デフォルト
exportRule	<p>新しいボリュームに対するエクスポートルール</p> <p>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。</p> <p>SMBボリュームでは無視されます。</p>	"0.0.0.0/0 "
snapshotDir	.snapshot ディレクトリの表示を制御します	いいえ
size	新しいボリュームのデフォルトサイズ	" 100G "
unixPermissions	<p>新しいボリュームのUNIX権限（8進数の4桁）。</p> <p>SMBボリュームでは無視されます。</p>	""（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）



ANFバックエンドで作成されたすべてのボリュームに対して、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ストレージボリュームにコピーします。ストレージ管理者は、ストレージプールごとにラベルを定義し、ストレージプール内に作成されたすべてのボリュームをグループ化できます。これは、バックエンド構成で提供されるカスタマイズ可能な一連のラベルに基づいてボリュームを区別するのに便利です。

構成例

例 1 : 最小限の構成

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

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

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

例 2 : 容量プールフィルタを使用した特定のサービスレベル設定

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

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

例 3：高度な設定

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

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

例 4：仮想ストレージプールの構成

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

```

{
  "version": 1,
  "storageDriverName": "azure-netapp-files",
  "subscriptionID": "9f87c765-4774-fake-ae98-a721add45451",
  "tenantID": "68e4f836-edc1-fake-bff9-b2d865ee56cf",
  "clientID": "dd043f63-bf8e-fake-8076-8de91e5713aa",
  "clientSecret": "SECRET",
  "location": "eastus",
  "resourceGroups": ["application-group-1"],
  "networkFeatures": "Basic",
  "nfsMountOptions": "vers=3,proto=tcp,timeo=600",
  "labels": {
    "cloud": "azure"
  },
  "location": "eastus",

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

```


ストレージクラスの定義

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

を使用した定義の例 `parameter.selector` フィールド

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

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

SMBボリュームの定義例

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

例1：デフォルトネームスペースの基本設定

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

例2：ネームスペースごとに異なるシークレットを使用する

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

例3：ボリュームごとに異なるシークレットを使用する

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



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

バックエンドを作成します

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs
```

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

GCP バックエンドの CVS を設定します

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

CVS for GCPのAstra Tridentサポートについてご確認ください

Astra Tridentは、デフォルトのCVSサービスタイプがonに設定されたボリュームをサポートしています **"GCP"**。Astra Tridentは、CVSサービスタイプで許可されている最小値に関係なく、100GiB未満のCVSボリュームをサポートしていません。したがって、Tridentは要求されたボリュームが最小サイズよりも小さい場合、自動的に100GiBのボリュームを作成します。

必要なもの

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

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
version		常に 1
storageDriverName	ストレージドライバの名前	"GCP-cvs"

パラメータ	説明	デフォルト
backendName	カスタム名またはストレージバックエンド	ドライバ名 + "_" + API キーの一部
storageClass	ストレージのタイプから選択します hardware（パフォーマンス最適化済み）または software（CVSサービスタイプ）	
projectNumber	Google Cloud アカウントのプロジェクト番号。この値は、Google Cloud ポータルのホームページにあります。	
apiRegion	CVS アカウント地域。バックエンドがボリュームをプロビジョニングするリージョンです。	
apiKey	を使用したGoogle CloudサービスアカウントのAPIキー netappcloudvolumes.admin ロール。このレポートには、Google Cloud サービスアカウントの秘密鍵ファイルの JSON 形式のコンテンツが含まれています（バックエンド構成ファイルにそのままコピーされます）。	
proxyURL	CVS アカウントへの接続にプロキシサーバが必要な場合は、プロキシ URL を指定します。プロキシサーバには、HTTP プロキシまたは HTTPS プロキシを使用できます。HTTPS プロキシの場合、プロキシサーバで自己署名証明書を使用するために証明書の検証はスキップされます。認証が有効になっているプロキシサーバはサポートされていません。	
nfsMountOptions	NFS マウントオプションのきめ細かな制御。	"nfsvers=3 "
limitVolumeSize	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します	""（デフォルトでは適用されません）
serviceLevel	新しいボリュームの CVS サービスレベル。「Standard」、「Premium」、「Extreme」のいずれかです。	標準
network	CVSボリュームに使用するGCPネットワーク	デフォルト

パラメータ	説明	デフォルト
debugTraceFlags	トラブルシューティング時に使用するデバッグフラグ。例： \{"api":false, "method":true}。トラブルシューティングを行って詳細なログダンプが必要な場合を除き、このオプションは使用しないでください。	null

共有VPCネットワークを使用する場合は、両方のポートを使用します `projectNumber` および `hostProjectNumber` を指定する必要があります。その場合は、 `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。

。 `apiRegion` Astra TridentがCVSボリュームを作成するGCPリージョンを表します。クロスリージョンのKubernetesクラスタを作成する場合、で作成されたCVSボリューム `apiRegion` 複数のGCPリージョンのノードでスケジュールされているワークロードで使用できます。リージョン間トラフィックは追加コストがかかることに注意してください。

- クロスリージョンアクセスを有効にするには、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
```

- `storageClass` は、必要なを選択するためのオプションのパラメータです "[CVS サービスタイプ](#)"。基本CVSサービスタイプから選択できます (`storageClass=software`) またはCVS -パフォーマンスサービスのタイプ (`storageClass=hardware`) を使用します。これは、デフォルトでTridentが使用します。必ずを指定してください `apiRegion` それぞれのCVSを提供します `storageClass` バックエンドの定義に含まれています。

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

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

構成ファイルの特別なセクションで次のオプションを指定することで、各ボリュームのデフォルトのプロビジョニング方法を制御できます。以下の設定例を参照してください。

パラメータ	説明	デフォルト
exportRule	新しいボリュームのエクスポートルール	"0.0.0.0/0 "

例 2：基本 CVS サービスタイプの設定

```
{
  "version": 1,
  "storageDriverName": "gcp-cvs",
  "projectNumber": "012345678901",
  "storageClass": "software",
  "apiRegion": "us-east4",
  "apiKey": {
    "type": "service_account",
    "project_id": "my-gcp-project",
    "private_key_id": "1234567890123456789012345678901234567890",
    "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisI
sAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSa
PIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZN
chRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzll
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
```

```

zZsrtrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrtHi
sIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrtHisIsAbOgu
SaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrtHisIsAbOguSaPIKeyA
ZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrtHisIsAbOguSaPIKeyAZNchRAGz
1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrtrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrtrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrtrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtr
HisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrtHisIsAbO
guSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrtHisIsAbOguSaPIKe
yAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrtHisIsAbOguSaPIKeyAZNchRA
Gz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
    "client_email": "cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com",
    "client_id": "123456789012345678901",
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://oauth2.googleapis.com/token",
    "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
    "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com"
  }
}

```

例 3：単一のサービスレベルの設定

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

```
{
  "version": 1,
  "storageDriverName": "gcp-cvs",
  "projectNumber": "012345678901",
  "apiRegion": "us-west2",
  "apiKey": {
    "type": "service_account",
    "project_id": "my-gcp-project",
    "private_key_id": "1234567890123456789012345678901234567890",
    "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisI
```



```

sAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSa
PIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZN
chRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1z
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
zZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHi
sIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOgu
SaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyA
ZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz
1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrth
HisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbO
guSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKe
yAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRA
Gz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
    "client_email": "cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com",
    "client_id": "123456789012345678901",
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://oauth2.googleapis.com/token",
    "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
    "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com"
  },
  "proxyURL": "http://proxy-server-hostname/",
  "nfsMountOptions": "vers=3,proto=tcp,timeo=600",
  "limitVolumeSize": "10Ti",
  "serviceLevel": "premium",
  "defaults": {
    "snapshotDir": "true",
    "snapshotReserve": "5",
    "exportRule": "10.0.0.0/24,10.0.1.0/24,10.0.2.100",
    "size": "5Ti"
  }
}

```

例 4：仮想ストレージプールの構成

この例は、仮想ストレージプールとともに設定されたバックエンド定義ファイルを示しています
StorageClasses それはそれらを再度参照する。

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 snapshotReserve 5%およびである exportRule を0.0.0.0/0に設定します。仮想ストレージプールは、で定義されます storage セクション。この例では、個々のストレージプールが独自に設定されています `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。

```
{
  "version": 1,
  "storageDriverName": "gcp-cvs",
  "projectNumber": "012345678901",
  "apiRegion": "us-west2",
  "apiKey": {
    "type": "service_account",
    "project_id": "my-gcp-project",
    "private_key_id": "1234567890123456789012345678901234567890",
    "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisI
sAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSa
PIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZN
chRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzll
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
zZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHi
sIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOgu
SaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyA
ZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz
llZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtr
HisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbO
guSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKe
yAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRA
GzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
    "client_email": "cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com",
```

```

    "client_id": "123456789012345678901",
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://oauth2.googleapis.com/token",
    "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
    "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com"
  },
  "nfsMountOptions": "vers=3,proto=tcp,timeo=600",

  "defaults": {
    "snapshotReserve": "5",
    "exportRule": "0.0.0.0/0"
  },

  "labels": {
    "cloud": "gcp"
  },
  "region": "us-west2",

  "storage": [
    {
      "labels": {
        "performance": "extreme",
        "protection": "extra"
      },
      "serviceLevel": "extreme",
      "defaults": {
        "snapshotDir": "true",
        "snapshotReserve": "10",
        "exportRule": "10.0.0.0/24"
      }
    },
    {
      "labels": {
        "performance": "extreme",
        "protection": "standard"
      },
      "serviceLevel": "extreme"
    },
    {
      "labels": {
        "performance": "premium",
        "protection": "extra"
      },

```

```

        "serviceLevel": "premium",
        "defaults": {
            "snapshotDir": "true",
            "snapshotReserve": "10"
        }
    },

    {
        "labels": {
            "performance": "premium",
            "protection": "standard"
        },
        "serviceLevel": "premium"
    },

    {
        "labels": {
            "performance": "standard"
        },
        "serviceLevel": "standard"
    }
]
}

```

次の StorageClass 定義は、上記のストレージプールを参照してください。を使用します
parameters.selector フィールドでは、ボリュームのホストに使用され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。

最初のストレージクラス (cvs-extreme-extra-protection) を最初の仮想ストレージ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、どの仮想ストレージプールを選択するかを決定し、Snapshot リザーブの要件を確実に満たします。

```

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

```

```

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

```

次の手順

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs
```

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

NetApp HCI または SolidFire バックエンドを設定します

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

必要なもの

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

知っておくべきこと

。solidfire-san ストレージドライバは、ボリュームモード（fileとblock）の両方をサポートしています。をクリックします Filesystem volumeMode、Astra Tridentがボリュームを作成し、ファイルシステムを作成ファイルシステムのタイプは StorageClass で指定されます。

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



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



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

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

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

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



使用しないでください debugTraceFlags トラブルシューティングを実行していて、詳細なログダンプが必要な場合を除きます。



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

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

次の例は、CHAP 認証を使用するバックエンドファイルと、特定の QoS 保証を適用した 3 つのボリュームタイプのモデリングを示しています。その場合は、を使用して各ストレージクラスを使用するように定義します IOPS ストレージクラスのパラメータ。

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

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

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

以下に示すバックエンド定義ファイルの例では、すべてのストレージプールに対して特定のデフォルトが設定されています。これにより、が設定されます type シルバー。仮想ストレージプールは、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で独自のタイプが設定されており、一部のプールでは上記で設定したデフォルト値が上書きされます。


```

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

  "type": "Silver",
  "labels":{"store":"solidfire", "k8scluster": "dev-1-cluster"},
  "region": "us-east-1",

  "storage": [
    {
      "labels":{"performance":"gold", "cost":"4"},
      "zone":"us-east-1a",
      "type":"Gold"
    },
    {
      "labels":{"performance":"silver", "cost":"3"},
      "zone":"us-east-1b",
      "type":"Silver"
    },
    {
      "labels":{"performance":"bronze", "cost":"2"},
      "zone":"us-east-1c",
      "type":"Bronze"
    },
    {
      "labels":{"performance":"silver", "cost":"1"},
      "zone":"us-east-1d"
    }
  ]
}

```

次の StorageClass 定義は、上記の仮想ストレージプールを参照してください。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、最初の仮想ストレージ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、どの仮想ストレージプールを選択するかを判断し、ストレージ要件を確実に満たすようにします。

```

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

```

詳細については、こちらをご覧ください

- ["ボリュームアクセスグループ"](#)

バックエンドに **ONTAP SAN** ドライバを設定します

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

- ["準備"](#)
- ["設定と例"](#)

ユーザ権限

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 クラスターユーザまたはです 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、クラスターを使用して、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、クラスター管理者ユーザを限定的に置き換えるものです。



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

ONTAP 内では、Trident ドライバが使用できるより制限的な役割を作成することもできますが、推奨しません。Trident の新リリースでは、多くの場合、考慮すべき API が追加で必要になるため、アップグレードが難しく、エラーも起こりやすくなります。

バックエンドに**ONTAP SAN**ドライバを設定する準備をします

ONTAP SAN ドライバを使用して ONTAP バックエンドを設定するための準備方法について説明します。ONTAP バックエンドすべてに対して、Astra Trident が SVM に少なくとも 1 つのアグリゲートを割り当てておく必要があります。

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

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

認証

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

- **credential based** : 必要な権限を持つ ONTAP ユーザのユーザ名とパスワード。など、事前定義されたセキュリティログインロールを使用することを推奨します admin または vsadmin ONTAP のバージョンとの互換性を最大限に高めるため。
- **証明書ベース** : Astra Trident は、バックエンドにインストールされた証明書を使用して ONTAP クラスター

と通信することもできます。この場合、バックエンド定義には、Base64 でエンコードされたクライアント証明書、キー、および信頼された CA 証明書（推奨）が含まれている必要があります。

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



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

クレデンシャルベースの認証を有効にします

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

バックエンド定義の例は次のようになります。

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

バックエンド定義は、クレデンシャルがプレーンテキストで保存される唯一の場所であることに注意してください。バックエンドが作成されると、ユーザー名とパスワードが 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。
- trustedCACertificate: 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でサポートされていることを確認する cert 認証方式。

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

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

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

```
cat cert-backend.json
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "SanBackend",
  "managementLIF": "1.2.3.4",
  "dataLIF": "1.2.3.8",
  "svm": "vserver_test",
  "clientCertificate": "Faaaakkkkeeee...Vaaalllluuueeeee",
  "clientPrivateKey": "LS0tFaKE...0VaLuES0tLS0K",
  "trustedCACertificate": "QNFinfO...SiqOyN",
  "storagePrefix": "myPrefix_"
}

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

認証方法を更新するか、クレデンシャルをローテーションして

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

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

#Update backend with tridentctl
tridentctl update backend SanBackend -f cert-backend-updated.json -n
trident

+-----+-----+-----+
+-----+-----+
|      NAME      | STORAGE DRIVER |          UUID          |
STATE | VOLUMES |
+-----+-----+-----+
+-----+-----+
| SanBackend | ontap-san      | 586b1cd5-8cf8-428d-a76c-2872713612c1 |
online |          9 |
+-----+-----+-----+
+-----+-----+
```



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

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

igroup を指定します

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

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

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

を含むバックエンドの場合 igroupName 定義されている igroupName を使用して削除できます。tridentctl backend update Astra Tridentでigroupを自動処理すでにワークロードに接続されているボリュームへのアクセスが中断されることはありません。今後作成される igroup Astra Trident を使用して接続を処理します。



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

Astra Trident が CSI Provisioner として機能するように設定されている場合、Kubernetes ノード IQN は自動的に igroup に追加 / 削除されます。ノードがKubernetesクラスターに追加されると、trident-csi DemonSetによってポッドが展開されます (trident-csi-xxxxx) を追加し、ボリュームを接続できる新しいノードを登録します。ノード IQN もバックエンドの igroup に追加されます。ノードが遮断され、削除され、Kubernetes から削除された場合も、同様の手順で IQN の削除が処理されます。

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

双方向 **CHAP** を使用して接続を認証します

Astra Tridentは、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ます。useCHAP バックエンド定義のオプション。に設定すると true、Astra Tridentは、SVMのデフォルトのイニシエータセキュリティを双方向CHAP に設定し、バックエンドファイルからのユーザ名とシークレットを設定します。接続の認証には双方向 CHAP を使用することを推奨します。次の設定例を参照してください。

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "ontap_san_chap",
  "managementLIF": "192.168.0.135",
  "svm": "ontap_iscsi_svm",
  "useCHAP": true,
  "username": "vsadmin",
  "password": "FaKePaSsWoRd",
  "igroupName": "trident",
  "chapInitiatorSecret": "cl9qxIm36DKyawxy",
  "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
  "chapTargetUsername": "iJF4heBRT0TCwxyz",
  "chapUsername": "uh2aNCLSD6cNwxyz",
}
```



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

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

動作の仕組み

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

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

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

クレデンシャルをローテーションし、バックエンドを更新

CHAPクレデンシャルを更新するには、でCHAPパラメータを更新します backend.json ファイル。CHAPシ

ークレットを更新し、を使用する必要があります `tridentctl update` 変更を反映するためのコマンドです。



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

```
cat backend-san.json
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "ontap_san_chap",
  "managementLIF": "192.168.0.135",
  "svm": "ontap_iscsi_svm",
  "useCHAP": true,
  "username": "vsadmin",
  "password": "FaKePaSsWoRd",
  "igroupName": "trident",
  "chapInitiatorSecret": "cl9qxUpDaTeD",
  "chapTargetInitiatorSecret": "rqxigXgkeUpDaTeD",
  "chapTargetUsername": "iJF4heBRT0TCwxyz",
  "chapUsername": "uh2aNCLSD6cNwxyz",
}
```

```
./tridentctl update backend ontap_san_chap -f backend-san.json -n trident
+-----+-----+-----+-----+
+-----+-----+
|  NAME          | STORAGE DRIVER |                               UUID                               |
STATE | VOLUMES |
+-----+-----+-----+-----+
+-----+-----+
| ontap_san_chap | ontap-san      | aa458f3b-ad2d-4378-8a33-1a472ffbeeb5c |
online |         7 |
+-----+-----+-----+-----+
+-----+-----+
```

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

ONTAP のSAN構成オプションと例

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

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
version		常に 1
storageDriverName	ストレージドライバの名前	「ONTAP-NAS」、「ONTAP-NAS-エコノミー」、「ONTAP-NAS-flexgroup」、「ONTAP-SAN」、「ONTAP-SAN-エコノミー」
backendName	カスタム名またはストレージバックエンド	ドライバ名 + "_" + データ LIF
managementLIF	クラスタ管理 LIF または SVM 管理 LIF の IP アドレス MetroClusterのシームレスなスイッチオーバーを実現するには、SVM 管理LIFを指定する必要があります。	「10.0.0.1」、「[2001:1234:abcd::fefe]」
dataLIF	プロトコル LIF の IP アドレス。IPv6 には角かっこを使用します。設定後に更新することはできません	特に指定がないかぎり、SVM が派生します
useCHAP	CHAP を使用して ONTAP SAN ドライバ用の iSCSI を認証する [ブーリアン]	いいえ
chapInitiatorSecret	CHAP イニシエータシークレット。の場合は必須です useCHAP=true	「」
labels	ボリュームに適用する任意の JSON 形式のラベルのセット	「」
chapTargetInitiatorSecret	CHAP ターゲットイニシエータシークレット。の場合は必須です useCHAP=true	「」
chapUsername	インバウンドユーザ名。の場合は必須です useCHAP=true	「」
chapTargetUsername	ターゲットユーザ名。の場合は必須です useCHAP=true	「」
clientCertificate	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	「」
clientPrivateKey	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	「」

パラメータ	説明	デフォルト
trustedCACertificate	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます	「」
username	クラスタ / SVM に接続するためのユーザ名。クレデンシャルベースの認証に使用されます	「」
password	クラスタ / SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます	「」
svm	使用する Storage Virtual Machine	SVMの場合に生成されます managementLIF を指定します
igroupName	SAN ボリュームで使用する igroup の名前	"trident-<backend-UUID> "
storagePrefix	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。設定後に更新することはできません	Trident
limitAggregateUsage	使用率がこの割合を超えている場合は、プロビジョニングが失敗します。* Amazon FSX for ONTAP * には適用されません	""（デフォルトでは適用されません）
limitVolumeSize	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。	""（デフォルトでは適用されません）
lunsPerFlexvol	FlexVol あたりの最大 LUN 数。有効な範囲は 50、200 です	100
debugTraceFlags	トラブルシューティング時に使用するデバッグフラグ。例：{"API" : false、"method" : true}	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 * MetroClusterではサポートされません。	いいえ

<code>useREST</code> 考慮事項



- 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。この機能を使用するには、ONTAP 9.10 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。

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



ネットアップONTAP バックエンドにAmazon FSXを使用している場合は、を指定しないでください `limitAggregateUsage` パラメータ。 `fsxadmin` および `vsadmin` Amazon FSX for NetApp ONTAP のロールには、アグリゲートの使用状況を取得し、Astra Tridentを通じて制限するために必要なアクセス権限が含まれていません。



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。

をクリックします `ontap-san` ドライバのデフォルトでは、SVMのすべてのデータLIF IPが使用され、iSCSI マルチパスが使用されます。のデータLIFのIPアドレスを指定します `ontap-san` ドライバは、マルチパスを無効にして、指定されたアドレスだけを使用します。



バックエンドを作成するときは、この点に注意してください `dataLIF` および `storagePrefix` 作成後に変更することはできません。これらのパラメータを更新するには、新しいバックエンドを作成する必要があります。

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

バックエンドは、作成後に igroup を更新することもできます。

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

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

には完全修飾ドメイン名 (FQDN) を指定できます `managementLIF` オプション

``managementLIF`` すべてのONTAP ドライバをIPv6
アドレスに設定することもできます。Tridentをに必ずインストールしてください ``---use-
ipv6`` フラグ。定義には注意が必要です ``managementLIF`` 角かっこ内のIPv6アドレス。



IPv6アドレスを使用する場合は、を確認してください `managementLIF` および `dataLIF` (バックエンド定義に含まれている場合) は、`[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`などの角括弧内に定義されます。状況 `dataLIF` が指定されていない場合、Astra TridentがSVMからIPv6 データLIFを取得します。

SANドライバでCHAPを使用できるようにするには、を設定します `useCHAP` パラメータの値 `true` バックエンドの定義に含まれています。その後、Astra Trident が、バックエンドで指定された SVM のデフォルト認証

として双方向 CHAP を設定して使用します。を参照してください ["こちらをご覧ください"](#) その仕組みについては、を参照してください。

をクリックします `ontap-san-economy` ドライバ、`limitVolumeSize` オプションを使用すると、`qtree`およびLUN用に管理するボリュームの最大サイズも制限されます。



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

ボリュームのプロビジョニング用のバックエンド構成オプション

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

パラメータ	説明	デフォルト
<code>spaceAllocation</code>	<code>space-allocation</code> for LUN のコマンドを指定します	正しいです
<code>spaceReserve</code>	スペースリザーベーションモード： 「none」（シン）または「volume」（シック）	なし
<code>snapshotPolicy</code>	使用する Snapshot ポリシー	なし
<code>qosPolicy</code>	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 <code>adaptiveQosPolicy</code> のいずれかを選択します	「」
<code>adaptiveQosPolicy</code>	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 <code>adaptiveQosPolicy</code> のいずれかを選択します	「」
<code>snapshotReserve</code>	Snapshot 「0」用にリザーブされているボリュームの割合	状況 <code>snapshotPolicy</code> は「none」、それ以外は「」です。
<code>splitOnClone</code>	作成時にクローンを親からスプリットします	いいえ
<code>splitOnClone</code>	作成時にクローンを親からスプリットします	いいえ

パラメータ	説明	デフォルト
encryption	<p>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 <code>false</code>。このオプションを使用するには、クラスターで 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。</p> <p>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。</p> <p>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"。</p>	いいえ
luksEncryption	LUKS暗号化を有効にします。を参照してください "Linux Unified Key Setup (LUKS；統合キーセットアップ) を使用" 。	""
securityStyle	新しいボリュームのセキュリティ形式	「UNIX」
tieringPolicy	「none」を使用する階層化ポリシー	ONTAP 9.5 よりも前の SVM-DR 構成の「スナップショットのみ」



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、各コンスチチュエントに個別にポリシーグループを適用することを推奨します。共有 QoS ポリシーグループにより、すべてのワークロードの合計スループットに対して上限が適用されます。

次に、デフォルトが定義されている例を示します。


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



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

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

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

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

Vserver	Volume	Aggregate	State	Type	Size	Available	Used%
		_pvc_89f1c156_3801_4de4_9f9d_034d54c395f4	online	RW	10GB	5.00GB	0%
		_pvc_e42ec6fe_3baa_4af6_996d_134adbbb8e6d	online	RW	5.79GB	5.50GB	0%
		_pvc_e8372153_9ad9_474a_951a_08ae15e1c0ba	online	RW	1GB	511.8MB	0%

3 entries were displayed.

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

最小限の設定例

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



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

ontap-san 証明書ベースの認証を使用するドライバ

これは、バックエンドの最小限の設定例です。clientCertificate、clientPrivateKey および trustedCACertificate（信頼されたCAを使用している場合はオプション）が入力されます 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",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi",
  "useCHAP": true,
  "chapInitiatorSecret": "cl9qxIm36DKyawxy",
  "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
  "chapTargetUsername": "iJF4heBRT0TCwxyz",
  "chapUsername": "uh2aNCLSD6cNwxyz",
  "igroupName": "trident",
  "clientCertificate": "ZXR0ZXJwYXB...ICMgJ3BhcGVyc2",
  "clientPrivateKey": "vciwKIyAgZG...0cnksIGRlc2NyaX",
  "trustedCACertificate": "zcyBbaG...b3Igb3duIGNsYXNz"
}
```

ontap-san 双方向CHAPを備えたドライバ

これは、バックエンドの最小限の設定例です。この基本設定では、が作成されます ontap-san バックエンドの指定 useCHAP をに設定します true。

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi",
  "labels": {"k8scluster": "test-cluster-1", "backend": "testcluster1-
sanbackend"},
  "useCHAP": true,
  "chapInitiatorSecret": "cl9qxIm36DKyawxy",
  "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
  "chapTargetUsername": "iJF4heBRT0TCwxyz",
  "chapUsername": "uh2aNCLSD6cNwxyz",
  "igroupName": "trident",
  "username": "vsadmin",
  "password": "secret"
}
```

ontap-san-economy ドライバ

```
{
  "version": 1,
  "storageDriverName": "ontap-san-economy",
  "managementLIF": "10.0.0.1",
  "svm": "svm_iscsi_eco",
  "useCHAP": true,
  "chapInitiatorSecret": "cl9qxIm36DKyawxy",
  "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
  "chapTargetUsername": "iJF4heBRT0TCwxyz",
  "chapUsername": "uh2aNCLSD6cNwxyz",
  "igroupName": "trident",
  "username": "vsadmin",
  "password": "secret"
}
```

仮想ストレージプールを使用するバックエンドの例

次のバックエンド定義ファイルの例では、などの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 spaceReserve 「なし」の場合は、spaceAllocation との誤り encryption 実行されます。仮想ストレージプールは、ストレージセクションで定義します。

この例では、一部のストレージプールが独自に設定されています spaceReserve、spaceAllocation`および `encryption 値を指定すると、一部のプールでは、上記のデフォルト値が上書きされます。

```

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

  "defaults": {
    "spaceAllocation": "false",
    "encryption": "false",
    "qosPolicy": "standard"
  },
  "labels":{"store": "san_store", "kubernetes-cluster": "prod-cluster-1"},
  "region": "us_east_1",
  "storage": [
    {
      "labels":{"protection":"gold", "creditpoints":"40000"},
      "zone":"us_east_1a",
      "defaults": {
        "spaceAllocation": "true",
        "encryption": "true",
        "adaptiveQosPolicy": "adaptive-extreme"
      }
    },
    {
      "labels":{"protection":"silver", "creditpoints":"20000"},
      "zone":"us_east_1b",
      "defaults": {
        "spaceAllocation": "false",
        "encryption": "true",
        "qosPolicy": "premium"
      }
    },
    {
      "labels":{"protection":"bronze", "creditpoints":"5000"},
      "zone":"us_east_1c",
      "defaults": {

```

```

        "spaceAllocation": "true",
        "encryption": "false"
    }
}
]
}

```

のiSCSIの例を次に示します ontap-san-economy ドライバ:

```

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

  "defaults": {
    "spaceAllocation": "false",
    "encryption": "false"
  },
  "labels": {"store": "san_economy_store"},
  "region": "us_east_1",
  "storage": [
    {
      "labels": {"app": "oracledb", "cost": "30"},
      "zone": "us_east_1a",
      "defaults": {
        "spaceAllocation": "true",
        "encryption": "true"
      }
    },
    {
      "labels": {"app": "postgresdb", "cost": "20"},
      "zone": "us_east_1b",
      "defaults": {
        "spaceAllocation": "false",
        "encryption": "true"
      }
    }
  ]
}

```

```

    },
    {
      "labels": {"app": "mysqldb", "cost": "10"},
      "zone": "us_east_1c",
      "defaults": {
        "spaceAllocation": "true",
        "encryption": "false"
      }
    }
  ]
}

```

バックエンドを **StorageClasses** にマッピングします

次の StorageClass 定義は、上記の仮想ストレージプールを参照してください。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 (protection-gold) を指定すると、内の1番目と2番目の仮想ストレージプールにマッピングされます `ontap-nas-flexgroup` 内の最初の仮想ストレージプール `ontap-san` バックエンド：ゴールドレベルの保護を提供している唯一のプールです。
- 2つ目のStorageClass (protection-not-gold) は、の3番目、4番目の仮想ストレージプールにマッピングされます `ontap-nas-flexgroup` のバックエンドと2番目の3番目の仮想ストレージプール `ontap-san` 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。
- 第3のストレージクラス (app-mysqldb) をクリックすると、で4番目の仮想ストレージプールにマッピングされます `ontap-nas` のバックエンドと3つ目の仮想ストレージプール `ontap-san-economy` バックエンド：mysqldb タイプのアプリケーション用のストレージプール設定を提供しているプールは、これらだけです。
- 第4のストレージクラス (protection-silver-creditpoints-20k) は、の3番目の仮想ストレージプールにマッピングされます `ontap-nas-flexgroup` のバックエンドと2つ目の仮想ストレージプール `ontap-san` バックエンド：ゴールドレベルの保護を提供している唯一のプールは、20000 の利用可能なクレジットポイントです。
- 第5のストレージクラス (creditpoints-5k) をクリックすると、で2つ目の仮想ストレージプールにマッピングされます `ontap-nas-economy` のバックエンドと3つ目の仮想ストレージプール `ontap-san` バックエンド：5000 ポイントの利用可能な唯一のプールは以下のとおりです。

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

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: netapp.io/trident
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-silver-creditpoints-20k
provisioner: netapp.io/trident
parameters:
  selector: "protection=silver; creditpoints=20000"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: netapp.io/trident
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"

```

ONTAP NASバックエンドを設定します

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

- "準備"
- "設定と例"



を使用する必要があります `ontap-nas` データ保護、ディザスタリカバリ、モビリティを必要とする本番ワークロード向けのドライバ。Astra Controlは、で作成されたボリュームに対して、シームレスな保護、ディザスタリカバリ、モビリティを提供します `ontap-nas` ドライバ。。 `ontap-nas-economy` ドライバは、想定されるボリューム使用量がONTAPでサポートされる量よりもはるかに高く、予想されるデータ保護、ディザスタリカバリ、モビリティ（Kubernetesクラスタ間でのボリュームの移動）の要件がないと予想される限られたユースケースでのみ使用してください。

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 `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 `vsadmin` SVMユーザ、または同じロールを持つ別の名前のユーザ。。 `fsxadmin` このユーザは、クラスタ管理者ユーザを限定的に置き換えるものです。



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

ONTAP 内では、Trident ドライバが使用できるより制限的な役割を作成することもできますが、推奨しません。Trident の新リリースでは、多くの場合、考慮すべき API が追加で必要になるため、アップグレードが難しく、エラーも起こりやすくなります。

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ドライバを使用してONTAPバックエンドを構成するための準備方法について説明します。ONTAP バックエンドすべてに対して、Astra Trident が SVM に少なくとも 1 つのアグリゲートを割り当てておく必要があります。

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

複数のドライバを実行し、1つまたは複数のドライバを参照するストレージクラスを作成することもできます。たとえば、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 には、ONTAP バックエンドを認証する 2 つのモードがあります。

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

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



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

クレデンシャルベースの認証を有効にします

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

バックエンド定義の例は次のようになります。

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

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

証明書ベースの認証を有効にします

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

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

一般的なワークフローは次の手順で構成されます。

手順

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

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

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

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

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

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

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

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

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

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

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

#Update backend with tridentctl
tridentctl update backend NasBackend -f cert-backend-updated.json -n
trident
```

NAME	STORAGE DRIVER	UUID
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 backend update`。

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

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



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

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

NFS エクスポートポリシーを管理します

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

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

- Astra Trident は、エクスポートポリシー自体を動的に管理できます。このモードでは、許容可能な IP アドレスを表す CIDR ブロックのリストをストレージ管理者が指定します。Astra Trident は、この範囲に含まれるノード IP をエクスポートポリシーに自動的に追加します。または、CIDRs が指定されていない場合は、ノード上で検出されたグローバルスコープのユニキャスト IP がエクスポートポリシーに追加されます。
- ストレージ管理者は、エクスポートポリシーを作成したり、ルールを手動で追加したりできます。構成に別のエクスポートポリシー名を指定しないと、Astra Trident はデフォルトのエクスポートポリシーを使用します。

エクスポートポリシーを動的に管理

CSI Trident の 20.04 リリースでは、ONTAP バックエンドのエクスポートポリシーを動的に管理できます。これにより、ストレージ管理者は、明示的なルールを手動で定義するのではなく、ワーカーノードの IP で許容されるアドレススペースを指定できます。エクスポートポリシーの管理が大幅に簡易化され、エクスポートポリシーを変更しても、ストレージクラスタに対する手動の操作は不要になります。さらに、ストレージクラスタへのアクセスを、指定した範囲の IP を持つワーカーノードだけに制限することで、きめ細かな管理と自動化をサポートします。



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

例

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

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap_nas_auto_export",
  "managementLIF": "192.168.0.135",
  "svm": "svm1",
  "username": "vsadmin",
  "password": "FaKePaSsWoRd",
  "autoExportCIDRs": ["192.168.0.0/24"],
  "autoExportPolicy": true
}
```



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

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

- `autoExportPolicy` がに設定されます `true`。これは、Astra Tridentがのエクスポートポリシーを作成することを示します `svm1` SVMで、を使用してルール追加と削除を処理します `autoExportCIDRs` アドレスブロック。たとえば、UUID `403b5326-842-40db-96d0-d83fb3f4daec`のバックエンドです `autoExportPolicy` をに設定します `true` という名前のエクスポートポリシーを作成します `trident-403b5326-8482-40db-96d0-d83fb3f4daec` 指定します。
- `autoExportCIDRs` アドレスブロックのリストが含まれます。このフィールドは省略可能で、デフォルト値は `["0.0.0.0/0", ":::/0"]` です。定義されていない場合は、Astra Trident が、ワーカーノードで検出されたすべてのグローバルにスコープ指定されたユニキャストアドレスを追加します。

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

更新できます `autoExportPolicy` および `autoExportCIDRs` バックエンドを作成したあとのバックエンドの場合自動的に管理されるバックエンドに新しい CIDRs を追加したり、既存の CIDRs を削除したりできます。CIDRs を削除する際は、既存の接続が切断されないように注意してください。無効にすることもできます `autoExportPolicy` をバックエンドに追加し、手動で作成したエクスポートポリシーに戻します。これにはを設定する必要があります `exportPolicy` バックエンド構成のパラメータ。

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

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

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

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

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



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

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

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、バックエンド構成の例と、バックエンドをストレージクラスにマッピングする方法を詳しく説明します。

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
version		常に 1
storageDriverName	ストレージドライバの名前	「ONTAP-NAS」、「ONTAP-NAS-エコノミー」、「ONTAP-NAS-flexgroup」、「ONTAP-SAN」、「ONTAP-SAN-エコノミー」
backendName	カスタム名またはストレージバックエンド	ドライバ名 + "_" + データ LIF
managementLIF	クラスタ管理 LIF または SVM 管理 LIF の IP アドレス MetroClusterのシームレスなスイッチオーバーを実現するには、SVM 管理LIFを指定する必要があります。	「10.0.0.1」、「[2001:1234:abcd::fefe]」
dataLIF	プロトコル LIF の IP アドレス。IPv6 には角かっこを使用します。設定後に更新することはできません	特に指定がないかぎり、SVM が派生します

パラメータ	説明	デフォルト
autoExportPolicy	エクスポートポリシーの自動作成と更新を有効にする [ブーリアン]	いいえ
autoExportCIDRs	KubernetesのノードIPをいつからフィルタリングするかを示すCIDRsのリスト autoExportPolicy が有効になります	[0.0.0.0/0]、[::/0]
labels	ボリュームに適用する任意のJSON形式のラベルのセット	「」
clientCertificate	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	「」
clientPrivateKey	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	「」
trustedCACertificate	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます	「」
username	クラスタ / SVM に接続するためのユーザ名。クレデンシャルベースの認証に使用されます	
password	クラスタ / SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます	
svm	使用する Storage Virtual Machine	SVMの場合に生成されます managementLIF を指定します
igroupName	SAN ボリュームで使用する igroup の名前	"trident-<backend-UUID> "
storagePrefix	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。設定後に更新することはできません	Trident
limitAggregateUsage	使用率がこの割合を超えている場合は、プロビジョニングが失敗します。* Amazon FSX for ONTAP * には適用されません	""（デフォルトでは適用されません）
limitVolumeSize	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。	""（デフォルトでは適用されません）
lunsPerFlexvol	FlexVol あたりの最大 LUN 数。有効な範囲は 50、200 です	100
debugTraceFlags	トラブルシューティング時に使用するデバッグフラグ。例：{"API" : false、"method" : true}	null

パラメータ	説明	デフォルト
nfsMountOptions	NFS マウントオプションをカンマで区切ったリスト	「」
qtreesPerFlexvol	FlexVol あたりの最大 qtree 数。有効な範囲は [50、300] です。	「200」
useREST	ONTAP REST API を使用するためのブーリアンパラメータ。* テクニカルプレビュー * MetroClusterではサポートされません。	いいえ

<code>useREST</code> 考慮事項



- 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。この機能を使用するには、ONTAP 9.10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。

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



ネットアップONTAP バックエンドにAmazon FSXを使用している場合は、を指定しないでください limitAggregateUsage パラメータ。 fsxadmin および vsadmin Amazon FSX for NetApp ONTAP のロールには、アグリゲートの使用状況を取得し、Astra Tridentを通じて制限するために必要なアクセス権限が含まれていません。



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。



バックエンドを作成するときは、を忘れないでください dataLIF および storagePrefix 作成後に変更することはできません。これらのパラメータを更新するには、新しいバックエンドを作成する必要があります。

には完全修飾ドメイン名 (FQDN) を指定できます managementLIF オプションにFQDNを指定することもできます dataLIF オプション。その場合は、NFSマウント処理にFQDNが使用されます。こうすることで、ラウンドロビン DNS を作成して、複数のデータ LIF 間で負荷を分散することができます。

`managementLIF` すべてのONTAP ドライバをIPv6アドレスに設定することもできます。Astra Tridentは、必ずを使用してインストールしてください `--use-ipv6` フラグ。を定義する際は注意が必要です `managementLIF` 角かっこ内のIPv6アドレス。



IPv6アドレスを使用する場合は、を確認してください managementLIF および dataLIF（バックエンド定義に含まれている場合）は、[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]などの角括弧内に定義されます。状況 dataLIF が指定されていない場合、Astra TridentがSVMからIPv6 データLIFを取得します。

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

をクリックします ontap-nas-economy ドライバ、 limitVolumeSize オプションを使用すると、qtreeおよびLUN用に管理するボリュームの最大サイズも制限されます qtreesPerFlexvol オプションを使用すると、FlexVol あたりの最大qtree数をカスタマイズできます。

。 nfsMountOptions パラメータを使用すると、マウントオプションを指定できます。Kubernetes 永続ボリュームのマウントオプションは通常ストレージクラスで指定されますが、ストレージクラスでマウントオプションが指定されていない場合、Astra Trident はストレージバックエンドの構成ファイルで指定されているマウントオプションを使用します。ストレージクラスまたは構成ファイルにマウントオプションが指定されていない場合、Astra Trident は関連付けられた永続的ボリュームにマウントオプションを設定しません。



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

ボリュームのプロビジョニング用のバックエンド構成オプション

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

パラメータ	説明	デフォルト
spaceAllocation	space-allocation for LUN のコマンドを指定します	正しいです
spaceReserve	スペースリザーベーションモード： 「none」（シン）または「volume」（シック）	なし
snapshotPolicy	使用する Snapshot ポリシー	なし
qosPolicy	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	「」

パラメータ	説明	デフォルト
adaptiveQosPolicy	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します。 経済性に影響する ONTAP - NAS ではサポートされません。	「」
snapshotReserve	Snapshot 「0」 用にリザーブされているボリュームの割合	状況 snapshotPolicy は「none」、それ以外は「」です。
splitOnClone	作成時にクローンを親からスプリットします	いいえ
encryption	新しいボリュームで NetApp Volume Encryption (NVE) を有効にします。デフォルトは false。このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。 NAE がバックエンドで有効になっている場合は、Astra Trident でプロビジョニングされたすべてのボリュームが NAE に有効になります。 詳細については、以下を参照してください。 "Astra Trident と NVE および NAE の相互運用性" 。	いいえ
securityStyle	新しいボリュームのセキュリティ形式	「UNIX」
tieringPolicy	「none」を使用する階層化ポリシー	ONTAP 9.5 よりも前の SVM-DR 構成の「スナップショットのみ」
unixPermissions	新しいボリュームのモード	「777」
Snapshot ディレクトリ	の表示/非表示を制御します .snapshot ディレクトリ	いいえ
エクスポートポリシー	使用するエクスポートポリシー	デフォルト
securityStyle の追加	新しいボリュームのセキュリティ形式	「UNIX」



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

次に、デフォルトが定義されている例を示します。

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

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

計算は次のとおりです。

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

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

Vserver	Volume	Aggregate	State	Type	Size	Available	Used%
		_pvc_89f1c156_3801_4de4_9f9d_034d54c395f4	online	RW	10GB	5.00GB	0%
		_pvc_e8372153_9ad9_474a_951a_08ae15e1c0ba	online	RW	1GB	511.8MB	0%

2 entries were displayed.

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

最小限の設定例

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



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

ontap-nas 証明書ベースの認証を使用するドライバ

これは、バックエンドの最小限の設定例です。 `clientCertificate`、 `clientPrivateKey` および `trustedCACertificate`（信頼されたCAを使用している場合はオプション）が入力されます `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": "ZXROZXJwYXB...ICMgJ3BhcGVyc2",
  "clientPrivateKey": "vciwKIyAgZG...0cnksIGRlc2NyaX",
  "trustedCACertificate": "zcyBbaG...b3Igb3duIGNsYXNz",
  "storagePrefix": "myPrefix_"
}
```

ontap-nas ドライバと自動エクスポートポリシー

この例は、動的なエクスポートポリシーを使用してエクスポートポリシーを自動的に作成および管理するように Astra Trident に指示する方法を示しています。これは、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": "secret",
  "nfsMountOptions": "nfsvers=4",
}
```

ontap-nas-flexgroup ドライバ

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

ontap-nas **IPv6**対応ドライバ

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "nas_ipv6_backend",
  "managementLIF": "[5c5d:5edf:8f:7657:bef8:109b:1b41:d491]",
  "labels": {"k8scluster": "test-cluster-east-1a", "backend": "test1-ontap-
ipv6"},
  "svm": "nas_ipv6_svm",
  "username": "vsadmin",
  "password": "netapp123"
}
```

ontap-nas-economy ドライバ

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

仮想ストレージプールを使用するバックエンドの例

次のバックエンド定義ファイルの例では、などの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。spaceReserve 「なし」の場合は、spaceAllocation との誤り encryption 実行されます。仮想ストレージプールは、ストレージセクションで定義します。

この例では、一部のストレージプールが独自に設定されています。spaceReserve、spaceAllocation および encryption 値を指定すると、一部のプールでは、上記のデフォルト値が上書きされます。

ontap-nas ドライバ

```
{
  {
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "admin",
    "password": "secret",
    "nfsMountOptions": "nfsvers=4",

    "defaults": {
      "spaceReserve": "none",
      "encryption": "false",
      "qosPolicy": "standard"
    },
    "labels": {"store": "nas_store", "k8scluster": "prod-cluster-1"},
    "region": "us_east_1",
    "storage": [
      {
        "labels": {"app": "msoffice", "cost": "100"},
        "zone": "us_east_1a",
        "defaults": {
```

```

        "spaceReserve": "volume",
        "encryption": "true",
        "unixPermissions": "0755",
        "adaptiveQosPolicy": "adaptive-premium"
    },
    {
        "labels":{"app":"slack", "cost":"75"},
        "zone":"us_east_1b",
        "defaults": {
            "spaceReserve": "none",
            "encryption": "true",
            "unixPermissions": "0755"
        }
    },
    {
        "labels":{"app":"wordpress", "cost":"50"},
        "zone":"us_east_1c",
        "defaults": {
            "spaceReserve": "none",
            "encryption": "true",
            "unixPermissions": "0775"
        }
    },
    {
        "labels":{"app":"mysqldb", "cost":"25"},
        "zone":"us_east_1d",
        "defaults": {
            "spaceReserve": "volume",
            "encryption": "false",
            "unixPermissions": "0775"
        }
    }
]
}

```

ontap-nas-flexgroup ドライバ

```

{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",

```



```

"password": "secret",

"defaults": {
    "spaceReserve": "none",
    "encryption": "false"
},
"labels":{"store":"flexgroup_store", "k8scluster": "prod-cluster-1"},
"region": "us_east_1",
"storage": [
    {
        "labels":{"protection":"gold", "creditpoints":"50000"},
        "zone":"us_east_1a",
        "defaults": {
            "spaceReserve": "volume",
            "encryption": "true",
            "unixPermissions": "0755"
        }
    },
    {
        "labels":{"protection":"gold", "creditpoints":"30000"},
        "zone":"us_east_1b",
        "defaults": {
            "spaceReserve": "none",
            "encryption": "true",
            "unixPermissions": "0755"
        }
    },
    {
        "labels":{"protection":"silver", "creditpoints":"20000"},
        "zone":"us_east_1c",
        "defaults": {
            "spaceReserve": "none",
            "encryption": "true",
            "unixPermissions": "0775"
        }
    },
    {
        "labels":{"protection":"bronze", "creditpoints":"10000"},
        "zone":"us_east_1d",
        "defaults": {
            "spaceReserve": "volume",
            "encryption": "false",
            "unixPermissions": "0775"
        }
    }
]

```

```
}
```

ontap-nas-economy ドライバ

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

  "defaults": {
    "spaceReserve": "none",
    "encryption": "false"
  },
  "labels": {"store": "nas_economy_store"},
  "region": "us_east_1",
  "storage": [
    {
      "labels": {"department": "finance", "creditpoints": "6000"},
      "zone": "us_east_1a",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "true",
        "unixPermissions": "0755"
      }
    },
    {
      "labels": {"department": "legal", "creditpoints": "5000"},
      "zone": "us_east_1b",
      "defaults": {
        "spaceReserve": "none",
        "encryption": "true",
        "unixPermissions": "0755"
      }
    },
    {
      "labels": {"department": "engineering", "creditpoints": "3000"},
      "zone": "us_east_1c",
      "defaults": {
        "spaceReserve": "none",
        "encryption": "true",
        "unixPermissions": "0775"
      }
    }
  ]
}
```

```

    }
  },
  {
    "labels":{"department":"humanresource",
"creditpoints":"2000"},
    "zone":"us_east_1d",
    "defaults": {
      "spaceReserve": "volume",
      "encryption": "false",
      "unixPermissions": "0775"
    }
  }
]
}

```

バックエンドを **StorageClasses** にマッピングします

次の StorageClass 定義は、上記の仮想ストレージプールを参照してください。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 (protection-gold) を指定すると、内の1番目と2番目の仮想ストレージプールにマッピングされます `ontap-nas-flexgroup` 内の最初の仮想ストレージプール `ontap-san` バックエンド：ゴールドレベルの保護を提供している唯一のプールです。
- 2つ目のStorageClass (protection-not-gold) は、の3番目、4番目の仮想ストレージプールにマッピングされます `ontap-nas-flexgroup` のバックエンドと2番目の3番目の仮想ストレージプール `ontap-san` 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。
- 第3のストレージクラス (app-mysqldb) をクリックすると、で4番目の仮想ストレージプールにマッピングされます `ontap-nas` のバックエンドと3つ目の仮想ストレージプール `ontap-san-economy` バックエンド：mysqldb タイプのアプリケーション用のストレージプール設定を提供しているプールは、これらだけです。
- 第4のストレージクラス (protection-silver-creditpoints-20k) は、の3番目の仮想ストレージプールにマッピングされます `ontap-nas-flexgroup` のバックエンドと2つ目の仮想ストレージプール `ontap-san` バックエンド：ゴールドレベルの保護を提供している唯一のプールは、20000 の利用可能なクレジットポイントです。
- 第5のストレージクラス (creditpoints-5k) をクリックすると、で2つ目の仮想ストレージプールにマッピングされます `ontap-nas-economy` のバックエンドと3つ目の仮想ストレージプール `ontap-san` バックエンド：5000 ポイントの利用可能な唯一のプールは以下のとおりです。

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

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: netapp.io/trident
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-silver-creditpoints-20k
provisioner: netapp.io/trident
parameters:
  selector: "protection=silver; creditpoints=20000"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: netapp.io/trident
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"

```

Amazon FSX for NetApp ONTAP で Astra Trident を使用

"NetApp ONTAP 対応の Amazon FSX"は、NetApp ONTAP ストレージ・オペレーティング・システムを搭載したファイル・システムの起動と実行を可能にする、フルマネージドの AWS サービスです。Amazon FSX for NetApp ONTAP を使用すると、使い慣れたネットアップの機能、パフォーマンス、管理機能を活用しながら、AWS にデータを格納する際の簡易性、即応性、セキュリティ、拡張性を活用できます。FSX は、ONTAP のファイルシステム機能と管理 API の多くをサポートしています。

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

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

ONTAP ファイルシステム用に Amazon FSX を作成します

自動バックアップが有効になっている Amazon FSX ファイルシステムで作成されたボリュームは Trident で削除できません。PVC を削除するには、PV と ONTAP ボリュームの FSX を手動で削除する必要があります。

この問題 を回避するには、次の手順

- ONTAP ファイル・システム用の FSX を作成する場合は '**Quick create**' を使用しないでください。クイック作成ワークフローでは、自動バックアップが有効になり、オプトアウトオプションはありません。
- **Standard create** を使用する場合は、自動バックアップを無効にしてください。自動バックアップを無効にすると、Trident は手動操作なしでボリュームを正常に削除できます。



▼ Backup and maintenance - optional

Daily automatic backup [Info](#)

Amazon FSx can protect your data through daily backups

☐ Enabled

☒ Disabled

Astra Trident の詳細をご確認ください

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

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

ラクティス"

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

ドライバー機能の詳細をご覧ください ["こちらをご覧ください"](#)。

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

Astra Tridentは vsadmin 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 クラスターユーザ：を使用することは推奨されません fsxadmin Tridentを使用したユーザ vsadmin SVMユーザは、より多くのAstra Trident機能にアクセスできます。

ドライバ

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

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

認証

Astra Trident には、次の 2 つの認証モードがあります。

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



を使用することを強く推奨します vsadmin ユーザがではなく fsxadmin バックエンドを設定します。Astra Trident は、このユーザ名とパスワードを使用して FSX ファイルシステムと通信します。

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、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。

認証の詳細については、次のリンクを参照してください。

- ["ONTAP NAS"](#)
- ["ONTAP SAN"](#)

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

必要なもの

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



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

Astra Trident のその他の要件については、を参照してください ["こちらをご覧ください"](#)。

手順

1. のいずれかを使用してAstra Tridentを導入 ["導入方法"](#)。
2. Trident を設定する手順は次のとおりです。
 - a. SVM の管理 LIF の DNS 名を収集します。たとえば、AWS CLIを使用してを検索します `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. 認証用の証明書を作成してインストールします。を使用する場合 `ontap-san` バックエンド。を参照してください ["こちらをご覧ください"](#)。を使用する場合 `ontap-nas` バックエンド。を参照してください ["こちらをご覧ください"](#)。



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

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

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

バックエンドの作成については、次のリンクを参照してください。

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



指定しないでください dataLIF をクリックします ontap-san および ontap-san-economy Astra Tridentがマルチパスを使用できるようにするためのドライバ。



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

導入後、次の手順を実行してを作成します ["ストレージクラスを定義してボリュームをプロビジョニングし、ポッドでボリュームをマウント"](#)。

詳細については、こちらをご覧ください

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

kubectl を使用してバックエンドを作成します

バックエンドは、Astra Trident とストレージシステムの関係性を定義します。Trident がストレージシステムとの通信方法を Trident から指示し、Astra Trident がボリュームをプロビジョニングする方法も解説します。Astra Trident のインストールが完了したら、次の手順でバックエンドを作成します。。

TridentBackendConfig Custom Resource Definition (CRD) を使用すると、TridentバックエンドをKubernetesインターフェイスから直接作成および管理できます。これは、を使用して実行できます kubectl または、Kubernetesディストリビューションと同等のCLIツールを使用します。

TridentBackendConfig

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

うになりました (tridentctl)。

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

- バックエンドは、指定した構成に基づいて Astra Trident によって自動的に作成されます。これは、内部的にはとして表されます TridentBackend (tbe、tridentbackend) CR。
- TridentBackendConfig は一意にバインドされます TridentBackend Astra Tridentによって作成されたのです。

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



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

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

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

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

。spec バックエンド固有の設定パラメータを使用します。この例では、バックエンドはを使用します ontap-san storage driverおよびでは、に示す構成パラメータを使用します。使用するストレージドライバの設定オプションの一覧については、を参照してください ["ストレージドライバのバックエンド設定情報"](#)。

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

- credentials：このパラメータは必須フィールドで、ストレージシステム/サービスとの認証に使用されるクレデンシャルが含まれています。ユーザが作成した Kubernetes Secret に設定されます。クレデンシャルをプレーンテキストで渡すことはできないため、エラーになります。
- deletionPolicy:このフィールドは、がどうなるかを定義します TridentBackendConfig が削除されました。次の2つの値のいずれかを指定できます。

- delete:この結果、両方が削除されます TridentBackendConfig CRとそれに関連付けられたバックエンド。これがデフォルト値です。
- retain:時 TridentBackendConfig CRが削除され、バックエンド定義は引き続き存在し、で管理できます tridentctl。削除ポリシーをに設定しています retain 以前のリリース（21.04より前）にダウングレードし、作成されたバックエンドを保持することができます。このフィールドの値は、のあとに更新できます TridentBackendConfig が作成されます。



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



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

手順の概要

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

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

バックエンドを作成したら、を使用してそのステータスを確認できます `kubectl get tbc <tbc-name> -n <trident-namespace>` 追加の詳細情報を収集します。

手順 1：Kubernetes Secret を作成します

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

```
kubectl -n trident create -f backend-tbc-ontap-san-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-san-secret
type: Opaque
stringData:
  username: cluster-admin
  password: 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シー クレットフィールドの説明	秘密	Fields概要
Azure NetApp Files の特長	ClientID	アプリケーション登録からのクラ イアント ID
Cloud Volumes Service for GCP	private_key_id です	秘密鍵の ID 。CVS 管理者ロール を持つ GCP サービスアカウントの API キーの一部
Cloud Volumes Service for GCP	private_key を使用します	秘密鍵CVS 管理者ロールを持つ GCP サービスアカウントの API キー の一部
Element （ NetApp HCI / SolidFire ）	エンドポイント	テナントのクレデンシャルを使用 する SolidFire クラスタの MVIP
ONTAP	ユーザ名	クラスタ / SVM に接続するための ユーザ名。クレデンシャルベース の認証に使用されます
ONTAP	パスワード	クラスタ / SVM に接続するための パスワード。クレデンシャルベー スの認証に使用されます
ONTAP	clientPrivateKey	クライアント秘密鍵の Base64 エ ンコード値。証明書ベースの認証 に使用されます
ONTAP	chapUsername のコマンド	インバウンドユーザ名。useCHAP = true の場合は必須。の場合 ontap-san および ontap-san- economy
ONTAP	chapInitiatorSecret	CHAP イニシエータシークレッ ト。useCHAP = true の場合は必 須。の場合 ontap-san および ontap-san-economy
ONTAP	chapTargetUsername のコマンド	ターゲットユーザ名。useCHAP = true の場合は必須。の場合 ontap-san および ontap-san- economy
ONTAP	chapTargetInitiatorSecret	CHAP ターゲットイニシエータシ ーケット。useCHAP = true の場 合は必須。の場合 ontap-san お よび ontap-san-economy

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

手順2：を作成します `TridentBackendConfig` CR

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

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

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

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

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

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

NAME	BACKEND NAME	BACKEND UUID
backend-tbc-ontap-san	ontap-san-backend	8d24fce7-6f60-4d4a-8ef6-bab2699e6ab8
Bound	Success	

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

フェーズには次のいずれかの値を指定できます。

- `Bound`: `TridentBackendConfig` CRはバックエンドに関連付けられており、そのバックエンドにはが含まれています `configRef` をに設定します `TridentBackendConfig` CRの`uid`。
- `Unbound`:を使用して表されます `""`。。 `TridentBackendConfig` オブジェクトがバックエンドにバインドされていません。新しく作成されたすべてのファイル `TridentBackendConfig` CRSはデフォルトでこのフェーズになっています。フェーズが変更された後、再度 `Unbound` に戻すことはできません。

- **Deleting:** TridentBackendConfig CR deletionPolicy が削除対象に設定されました。をクリックします TridentBackendConfig CRが削除され、削除状態に移行します。
 - バックエンドに永続ボリューム要求 (PVC) が存在しない場合は、を削除します TridentBackendConfig その結果、Astra Tridentによってバックエンドとが削除されます TridentBackendConfig CR。
 - バックエンドに 1 つ以上の PVC が存在する場合は、削除状態になります。。 TridentBackendConfig CRはその後、削除フェーズにも入ります。バックエンドと TridentBackendConfig は、すべてのPVCが削除されたあとにのみ削除されます。
- **Lost:**に関連付けられているバックエンド TridentBackendConfig CRが誤って削除されたか、故意に削除された TridentBackendConfig CRには削除されたバックエンドへの参照があります。。 TridentBackendConfig CRは、に関係なく削除できます deletionPolicy 価値。
- **Unknown:** Astra Tridentは、に関連付けられているバックエンドの状態または存在を特定できません TridentBackendConfig CR。たとえば、APIサーバが応答していない場合や、が応答していない場合などです tridentbackends.trident.netapp.io CRDがありません。これには、ユーザの介入が必要な場合があります。

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

(オプション) 手順 4：詳細を確認します

バックエンドに関する詳細情報を確認するには、次のコマンドを実行します。

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

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

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

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

```

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

```

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

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



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

kubectl を使用してバックエンド管理を実行します

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

バックエンドを削除します

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

次のコマンドを実行します。

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

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

既存のバックエンドを表示します

次のコマンドを実行します。

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

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

バックエンドを更新します

バックエンドを更新する理由はいくつかあります。

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

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

- パラメータ（使用する ONTAP SVM の名前など）を更新する必要があります。
この場合、`TridentBackendConfig` オブジェクトは Kubernetes を使用して直接更新できます。

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

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

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

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

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

tridentctl を使用してバックエンド管理を実行します

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

バックエンドを作成します

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

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

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

```
tridentctl logs -n trident
```

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

バックエンドを削除します

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

1. バックエンド名を取得します。

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

2. バックエンドを削除します。

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




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

既存のバックエンドを表示します

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

- 概要を取得するには、次のコマンドを実行します。

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

- すべての詳細を確認するには、次のコマンドを実行します。

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

バックエンドを更新します

新しいバックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs -n trident
```

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

バックエンドを使用するストレージクラスを特定します

以下は、回答 でできるJSON形式の質問の例です tridentctl バックエンドオブジェクトの出力。これにはを使用します jq ユーティリティをインストールする必要があります。

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

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

バックエンド管理オプション間を移動します

Astra Trident でバックエンドを管理するさまざまな方法をご確認ください。を導入しました `TridentBackendConfig` 管理者は現在、バックエンドを2つの方法で管理できるようになっています。これには、次のような質問があります。

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

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

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

これは、次のシナリオに該当します。

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

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

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

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

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

手順 0：バックエンドを特定します

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

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

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

```
cat ontap-nas-backend.json
```

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

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

```
]
}
```

手順 1 : Kubernetes Secret を作成します

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

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

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

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

手順2：を作成します TridentBackendConfig CR

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

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

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

```

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

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

```

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

のあとに入力します TridentBackendConfig が作成されている必要があります Bound。また、既存のバックエンドと同じバックエンド名と UUID が反映されている必要があります。

```
kubectl -n trident get tbc tbc-ontap-nas-backend -n trident
```

NAME	BACKEND NAME	BACKEND UUID
tbc-ontap-nas-backend	ontap-nas-backend	52f2eb10-e4c6-4160-99fc-96b3be5ab5d7

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

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

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

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

`tridentctl` を使用して、を使用して作成されたバックエンドを表示できます `TridentBackendConfig`。また、管理者は、を使用してこのようなバックエンドを完全に管理することもできます `tridentctl` 削除します `TridentBackendConfig` そして確かめなさい `spec.deletionPolicy` がに設定されます `retain`。

手順 0 : バックエンドを特定します

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

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

NAME	BACKEND NAME	BACKEND UUID
backend-tbc-ontap-san	ontap-san-backend	81abcb27-ea63-49bb-b606-0a5315ac5f82

```
tridentctl get backend ontap-san-backend -n trident
```

NAME	STORAGE DRIVER	UUID
ontap-san-backend	ontap-san	81abcb27-ea63-49bb-b606-0a5315ac5f82

出力からはそのことがわかります TridentBackendConfig は正常に作成され、バックエンドにバインドされています[バックエンドのUUIDを確認してください]。

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

では、の価値を見てみましょう deletionPolicy。これはに設定する必要があります retain。これにより、が確実に実行されます TridentBackendConfig CRが削除され、バックエンド定義は引き続き存在し、で管理できます tridentctl。

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

NAME	BACKEND NAME	BACKEND UUID
backend-tbc-ontap-san	ontap-san-backend	81abcb27-ea63-49bb-b606-0a5315ac5f82

```
# 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
backend-tbc-ontap-san	ontap-san-backend	81abcb27-ea63-49bb-b606-0a5315ac5f82



それ以外の場合は、次の手順に進まないでください `deletionPolicy` がに設定されます `retain`。

手順2：を削除します `TridentBackendConfig CR`

最後の手順は、を削除することです `TridentBackendConfig CR`。確認が完了したら `deletionPolicy` がに設定されます `retain` をクリックすると、次のように削除されます。

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

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

が削除されたとき `TridentBackendConfig Astra Trident`は、実際にバックエンド自体を削除することなく、単にオブジェクトを削除します。

ストレージクラスを管理する

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

ストレージクラスを設計する

を参照してください ["こちらをご覧ください"](#) ストレージクラスとその設定方法の詳細については、を参照してください。

ストレージクラスを作成する。

ストレージクラスファイルが作成されたら、次のコマンドを実行します。

```
kubectl create -f <storage-class-file>
```

`<storage-class-file>` は、ストレージクラスのファイル名に置き換えてください。

ストレージクラスを削除する

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

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

<storage-class> は、ストレージクラスで置き換える必要があります。

このストレージクラスで作成された永続ボリュームには変更はなく、Astra Trident によって引き続き管理されます。



Astra Tridentでは空白が強制される fsType を作成します。iSCSIバックエンドの場合は、適用することを推奨します parameters.fsType ストレージクラス。esixting StorageClassesを削除して、を使用して再作成する必要があります parameters.fsType 指定された。

既存のストレージクラスを表示します

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

```
kubectl get storageclass
```

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

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

- Astra Trident の同期されたストレージクラスを表示するには、次のコマンドを実行します。

```
tridentctl get storageclass
```

- Astra Trident の同期されたストレージクラスの詳細を表示するには、次のコマンドを実行します。

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

デフォルトのストレージクラスを設定する

Kubernetes 1.6 では、デフォルトのストレージクラスを設定する機能が追加されています。永続ボリューム要求（PVC）に永続ボリュームが指定されていない場合に、永続ボリュームのプロビジョニングに使用するストレージクラスです。

- アノテーションを設定してデフォルトのストレージクラスを定義します
storageclass.kubernetes.io/is-default-class をtrueに設定してストレージクラスの定義に追加します。仕様に応じて、それ以外の値やアノテーションがない場合は false と解釈されます。

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

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

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

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

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



クラスタには、常に 1 つのデフォルトストレージクラスだけを設定してください。Kubernetes では、技術的に複数のストレージを使用することはできますが、デフォルトのストレージクラスがまったくない場合と同様に動作します。

ストレージクラスのバックエンドを特定します

以下は、回答 でできる JSON 形式の質問の例です `tridentctl Astra Trident` バックエンドオブジェクトの出力これにははを使用します `jq` ユーティリティ。先にインストールする必要がある場合があります。

```
tridentctl get storageclass -o json | jq '[.items[] | {storageClass:  
.Config.name, backends: [.storage]|unique}]'
```

ボリューム操作を実行する

Trident がボリュームを管理するための各種機能をご紹介します。

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

CSI トポロジを使用します

Astra Trident では、を使用して、Kubernetes クラスタ内にあるノードにボリュームを選択的に作成して接続できます ["CSI トポロジ機能"](#)。CSI トポロジ機能を使用すると、領域およびアベイラビリティゾーンに基づいて、ボリュームへのアクセスをノードのサブセットに制限できます。現在、クラウドプロバイダは、Kubernetes 管理者がゾーンベースのノードを生成できるようになっています。ノードは、リージョンによって異なるアベイラビリティゾーンに配置することも、リージョンによって配置することもできます。マルチゾーンアーキテクチャでワークロード用のボリュームをプロビジョニングするために、Astra Trident は CSI トポロジを使用します。



CSI トポロジ機能の詳細については、を参照してください["こちらをご覧ください"](#)。

Kubernetes には、2 つの固有のボリュームバインドモードがあります。

- を使用 VolumeBindingMode をに設定します Immediate トポロジを認識することなくボリュームを作成できます。ボリュームバインディングと動的プロビジョニングは、PVC が作成されるときに処理されます。これがデフォルトです `VolumeBindingMode` また、トポロジの制約を適用しないクラスタにも適しています。永続ボリュームは、要求側ポッドのスケジュール要件に依存せずに作成されます。
- を使用 VolumeBindingMode をに設定します `WaitForFirstConsumer` PVCの永続的ボリュームの作成とバインディングは、PVCを使用するポッドがスケジュールされて作成されるまで遅延されます。これにより、トポロジの要件に応じたスケジュールの制約を満たすようにボリュームが作成されます。



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

必要なもの

CSI トポロジを使用するには、次のものがが必要です。

- 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:"1e11e4a2108024935ecfcb2912226cedeaafd99df",
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:"1e11e4a2108024935ecfcb2912226cedeaafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- クラスタ内のノードには、トポロジを認識するためのラベルが必要です (topology.kubernetes.io/region および topology.kubernetes.io/zone) 。このラベル * は、Astra Trident をトポロジ対応としてインストールする前に、クラスタ内のノードに存在する必要があります。

```
kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},
{.metadata.labels}]{ "\n"}{end}' | grep --color "topology.kubernetes.io"
[node1,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kubernetes.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","kubernetes.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","kubernetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/os":"linux","node-role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-east1","topology.kubernetes.io/zone":"us-east1-c"}]
```

手順 1：トポロジ対応バックエンドを作成する

Astra Trident ストレージバックエンドは、アベイラビリティゾーンに基づいてボリュームを選択的にプロビジョニングするように設計できます。各バックエンドはオプションで伝送できます `supportedTopologies` サポートする必要があるゾーンおよび領域のリストを表すブロック。ストレージクラスがそのようなバックエンドを使用する場合、ボリュームは、サポートされているリージョン/ゾーンでスケジューリングされているアプリケーションから要求された場合にのみ作成されます。

バックエンド定義の例を次に示します。

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "san-backend-us-east1",
  "managementLIF": "192.168.27.5",
  "svm": "iscsi_svm",
  "username": "admin",
  "password": "xxxxxxxxxxxx",
  "supportedTopologies": [
    {"topology.kubernetes.io/region": "us-east1",
     "topology.kubernetes.io/zone": "us-east1-a"},
    {"topology.kubernetes.io/region": "us-east1",
     "topology.kubernetes.io/zone": "us-east1-b"}
  ]
}
```



supportedTopologies は、バックエンドごとのリージョンとゾーンのリストを提供するために使用されます。これらのリージョンとゾーンは、StorageClass で指定できる許容値のリストを表します。バックエンドで提供されるリージョンとゾーンのサブセットを含む StorageClasses の場合、Astra Trident がバックエンドにボリュームを作成します。

を定義できます supportedTopologies ストレージプールごとに作成することもできます。次の例を参照してください。

```

{"version": 1,
"storageDriverName": "ontap-nas",
"backendName": "nas-backend-us-central1",
"managementLIF": "172.16.238.5",
"svm": "nfs_svm",
"username": "admin",
"password": "Netapp123",
"supportedTopologies": [
  {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-a"},
  {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-b"}
]
"storage": [
  {
    "labels": {"workload":"production"},
    "region": "Iowa-DC",
    "zone": "Iowa-DC-A",
    "supportedTopologies": [
      {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-a"}
    ]
  },
  {
    "labels": {"workload":"dev"},
    "region": "Iowa-DC",
    "zone": "Iowa-DC-B",
    "supportedTopologies": [
      {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-b"}
    ]
  }
]
}

```

この例では、を使用しています region および zone ラベルはストレージプールの場所を表します。 topology.kubernetes.io/region および topology.kubernetes.io/zone ストレージプールの使用場所を指定します。

手順 2：トポロジを認識するストレージクラスを定義する

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

次の例を参照してください。

```

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

```

上記のStorageClass定義で、volumeBindingMode がに設定されます WaitForFirstConsumer。このStorageClass で要求された PVC は、ポッドで参照されるまで処理されません。および、allowedTopologies 使用するゾーンとリージョンを提供します。。netapp-san-us-east1 StorageClassがにPVCを作成します san-backend-us-east1 上で定義したバックエンド。

ステップ 3 : PVC を作成して使用する

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

例を参照 spec 下記：

```

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

```

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

```

kubect1 create -f pvc.yaml
persistentvolumeclaim/pvc-san created
kubect1 get pvc
NAME          STATUS      VOLUME      CAPACITY    ACCESS MODES    STORAGECLASS
AGE
pvc-san      Pending                                netapp-san-us-east1
2s
kubect1 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
  ----      -
  Normal    WaitForFirstConsumer  6s     persistentvolume-controller
waiting
for first consumer to be created before binding
  Message
  -----

```

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


```

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

```

このpodSpecにより、Kubernetesは、にあるノードにPODをスケジュールするように指示されます us-east1 リージョンを選択し、にある任意のノードから選択します us-east1-a または us-east1-b ゾーン。

次の出力を参照してください。

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

バックエンドを更新して追加 supportedTopologies

既存のバックエンドを更新して、のリストを追加することができます supportedTopologies を使用します tridentctl backend update。これは、すでにプロビジョニングされているボリュームには影響せず、以降の PVC にのみ使用されます。

詳細については、こちらをご覧ください

- ["コンテナのリソースを管理"](#)
- ["ノードセクタ"](#)
- ["アフィニティと非アフィニティ"](#)
- ["塗料および耐性"](#)

スナップショットを操作します

永続ボリューム（PVS）のKubernetesボリュームSnapshot（ボリュームSnapshot）を作成して、Astra Tridentボリュームのポイントインタイムコピーを保持できます。また、既存のボリュームSnapshotから、`_clone__`という名前の新しいボリュームを作成することもできます。ボリュームSnapshotは、でサポートされます ontap-nas、ontap-san、ontap-san-economy、solidfire-san、gcp-cvs`および`azure-netapp-files ドライバ。

作業を開始する前に

外部スナップショットコントローラとカスタムリソース定義（CRD）が必要です。Kubernetesオーケストレーションツール（例：Kubeadm、GKE、OpenShift）の役割を担っています。

KubernetesディストリビューションにスナップショットコントローラとCRDが含まれていない場合は、を参照してください [ボリュームSnapshotコントローラを導入する](#)。



GKE環境でオンデマンドボリュームスナップショットを作成する場合は、スナップショットコントローラを作成しないでください。GKEでは、内蔵の非表示のスナップショットコントローラを使用します。

ステップ1: VolumeSnapshotClass

次の例は、ボリュームSnapshotクラスを作成します。

```
cat snap-sc.yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

。 driver Astra TridentのCSIドライバをポイントします。 deletionPolicy は、です Delete または Retain。 に設定すると Retain`を使用すると、ストレージクラスタの基盤となる物理Snapshotが、の場合でも保持されます `VolumeSnapshot オブジェクトが削除された。

詳細については、link: [./trident-reference/objects.html#Kubernetes -volumesnapshotclass-objects](#)を参照してください[VolumeSnapshotClass]。

手順 2 : 既存の PVC のスナップショットを作成します

次に、既存のPVCのスナップショットを作成する例を示します。

```
cat snap.yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: pvc1-snap
spec:
  volumeSnapshotClassName: csi-snapclass
  source:
    persistentVolumeClaimName: pvc1
```

この例では、という名前のPVCに対してスナップショットが作成されます pvc1 Snapshotの名前はに設定されます pvc1-snap。

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

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

これでが作成されました VolumeSnapshot オブジェクト。ボリュームSnapshotはPVCに似ており、に関連付けられています VolumeSnapshotContent 実際のスナップショットを表すオブジェクト。

を識別できます VolumeSnapshotContent のオブジェクト pvc1-snap ボリュームSnapshot。ボリュームSnapshotの詳細を定義します。

```
kubectl describe volumesnapshots pvc1-snap
Name:          pvc1-snap
Namespace:     default
.
.
.
Spec:
  Snapshot Class Name:    pvc1-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-525400f3f660
  Source:
    API Group:
    Kind:      PersistentVolumeClaim
    Name:      pvc1
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
.
.
```

。 Snapshot Content Name このSnapshotを提供するVolumeSnapshotContentオブジェクトを特定します。。 Ready To Use パラメータは、Snapshotを使用して新しいPVCを作成できることを示します。

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

次に、Snapshotを使用してPVCを作成する例を示します。

```
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: pvcl-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

dataSource に、という名前のボリュームSnapshotを使用してPVCを作成する必要があることを示します pvcl-snap データのソースとして。このコマンドを実行すると、Astra Trident が Snapshot から PVC を作成するように指示します。作成された PVC は、ポッドに接続して、他の PVC と同様に使用できます。



スナップショットが関連付けられている永続ボリュームを削除すると、対応する Trident ボリュームが「削除状態」に更新されます。Astra Trident ボリュームを削除するには、ボリュームの Snapshot を削除する必要があります。

ボリューム**Snapshot**コントローラを導入する

KubernetesディストリビューションにスナップショットコントローラとCRDが含まれていない場合は、次のように導入できます。

手順

1. ボリュームのSnapshot作成

```
cat snapshot-setup.sh
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

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

```
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
```

関連リンク

- ["ボリューム Snapshot"](#)
- ["ボリュームSnapshotクラス"](#)

ボリュームを展開します

Astra Trident により、Kubernetes ユーザは作成後にボリュームを拡張できます。ここでは、iSCSI ボリュームと NFS ボリュームの拡張に必要な設定について説明します。

iSCSI ボリュームを展開します

CSI プロビジョニングを使用して、iSCSI Persistent Volume（PV）を拡張できます。



iSCSIボリューム拡張は、でサポートされます ontap-san、ontap-san-economy、solidfire-san ドライバとにはKubernetes 1.16以降が必要です。

概要

iSCSI PV の拡張には、次の手順が含まれます。

- StorageClass定義を編集してを設定します allowVolumeExpansion フィールドからに移動します true。
- PVC定義を編集してを更新します spec.resources.requests.storage 新たに必要となったサイズを反映するには、元のサイズよりも大きくする必要があります。
- サイズを変更するには、PV をポッドに接続する必要があります。iSCSI PV のサイズ変更には、次の 2 つのシナリオがあります。
 - PV がポッドに接続されている場合、Astra Trident はストレージバックエンドのボリュームを拡張し、デバイスを再スキャンし、ファイルシステムのサイズを変更します。
 - 未接続の PV のサイズを変更しようとする、Astra Trident がストレージバックエンドのボリュームを拡張します。PVC がポッドにバインドされると、Trident はデバイスを再スキャンし、ファイルシステムのサイズを変更します。展開操作が正常に完了すると、Kubernetes は PVC サイズを更新します。

次の例は、iSCSI PVS の仕組みを示しています。

手順 1：ボリュームの拡張をサポートするようにストレージクラスを設定する

```
cat storageclass-ontapsan.yaml
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

既存のストレージクラスの場合は、編集してを追加します allowVolumeExpansion パラメータ

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

```
cat pvc-ontapsan.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Astra Trident が、永続的ボリューム（PV）を作成し、この永続的ボリューム要求（PVC）に関連付けます。

```
kubectl get pvc
```

NAME	STATUS	VOLUME	CAPACITY
ACCESS MODES	STORAGECLASS	AGE	
san-pvc	Bound	pvc-8a814d62-bd58-4253-b0d1-82f2885db671	1Gi
RWO	ontap-san	8s	


```
kubectl get pv
```

NAME	CAPACITY	ACCESS MODES
RECLAIM POLICY	STATUS	CLAIM
STORAGECLASS	REASON	AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671	1Gi	RWO
Delete	Bound	default/san-pvc
ontap-san		10s

手順 3：PVC を接続するポッドを定義します

この例では、を使用するポッドが作成されます san-pvc。

```
kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
centos-pod	1/1	Running	0	65s


```
kubectl describe pvc san-pvc
```

Name: san-pvc
Namespace: default
StorageClass: ontap-san
Status: Bound
Volume: pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels: <none>
Annotations: pv.kubernetes.io/bind-completed: yes
pv.kubernetes.io/bound-by-controller: yes
volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers: [kubernetes.io/pvc-protection]
Capacity: 1Gi
Access Modes: RWO
VolumeMode: Filesystem
Mounted By: centos-pod

ステップ 4：PV を展開します

1Giから2Giに作成されたPVのサイズを変更するには、PVCの定義を編集してを更新します spec.resources.requests.storage 2Giへ。


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

手順 5：拡張を検証する

PVC、PV、Astra Trident のボリュームのサイズを確認することで、拡張が正しく機能しているかどうかを検証できます。

```
kubectl get pvc san-pvc
NAME          STATUS    VOLUME                                     CAPACITY
ACCESS MODES  STORAGECLASS  AGE
san-pvc      Bound      pvc-8a814d62-bd58-4253-b0d1-82f2885db671  2Gi
RWO           ontap-san    11m

kubectl get pv
NAME          CAPACITY  ACCESS MODES
RECLAIM POLICY STATUS    CLAIM          STORAGECLASS  REASON    AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671  2Gi        RWO
Delete              Bound      default/san-pvc  ontap-san    12m

tridentctl get volumes -n trident
+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
|          NAME          | SIZE | STORAGE CLASS |
+-----+-----+-----+-----+-----+-----+
|          BACKEND UUID  | STATE | MANAGED |
+-----+-----+-----+-----+-----+-----+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san |
| block      | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true      |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
```

NFS ボリュームを拡張します

Astra Tridentは、でプロビジョニングしたNFS PVSのボリューム拡張をサポートしています ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、gcp-cvs`および `azure-netapp-files バックエンド

手順 1：ボリュームの拡張をサポートするようにストレージクラスを設定する

NFS PVのサイズを変更するには、管理者はまず、を設定してボリュームを拡張できるようにストレージクラスを構成する必要があります allowVolumeExpansion フィールドからに移動します true：

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

このオプションを指定せずにストレージクラスを作成済みの場合は、を使用して既存のストレージクラスを編集するだけです kubectl edit storageclass ボリュームを拡張できるようにするため。

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

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

Astra Trident が、この PVC に対して 20MiB の NFS PV を作成する必要があります。

```
kubectl get pvc
NAME                STATUS      VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
ontapnas20mb        Bound       pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7  20Mi       RWO             ontapnas       9s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                STORAGECLASS   REASON   AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7  20Mi       RWO             Delete           Bound    default/ontapnas20mb  ontapnas   2m42s
```

ステップ3 : **PV**を拡張する

新しく作成した20MiBのPVのサイズを1GiBに変更するには、そのPVCを編集してを設定します
spec.resources.requests.storage 1 GBに設定する場合 :

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

手順4：拡張を検証する

PVC、PV、Astra Trident のボリュームのサイズを確認することで、サイズ変更が正しく機能しているかどうかを検証できます。

```
kubectl get pvc ontapnas20mb
NAME          STATUS    VOLUME
CAPACITY      ACCESS MODES  STORAGECLASS  AGE
ontapnas20mb  Bound      pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7  1Gi
RWO           ontapnas      4m44s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME          CAPACITY  ACCESS MODES
RECLAIM POLICY STATUS    CLAIM          STORAGECLASS  REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7  1Gi      RWO
Delete          Bound      default/ontapnas20mb  ontapnas
5m35s

tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n trident
+-----+-----+-----+
+-----+-----+-----+-----+
|          NAME          | SIZE  | STORAGE CLASS |
+-----+-----+-----+-----+
| PROTOCOL | BACKEND UUID          | STATE | MANAGED |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file      | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true     |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

ボリュームをインポート

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

ボリュームインポートをサポートするドライバ

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

ドライバ	リリース。
ontap-nas	19.04
ontap-nas-flexgroup	19.04
solidfire-san	19.04
azure-netapp-files	19.04

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

ボリュームをインポートする理由

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

- アプリケーションのコンテナ化と既存のデータセットの再利用
- エフェメラルアプリケーション用のデータセットのクローンを使用する
- 障害が発生した Kubernetes クラスタの再構築
- ディザスタリカバリ時にアプリケーションデータを移行する

インポートはどのように機能しますか。

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

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

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

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

ストレージボリュームをインポートするには、ボリュームが含まれている Astra Trident バックエンドの名前と、ストレージ上のボリュームを一意に識別する名前（ONTAP FlexVol、Element Volume、CVS ボリュームパスなど）を指定します。ストレージボリュームは、読み取り / 書き込みアクセスを許可し、指定された Astra Trident バックエンドからアクセスできる必要があります。。-f string 引数は必須で、YAML または JSON PVC ファイルへのパスを指定します。

Astra Trident がインポートボリューム要求を受信すると、既存のボリュームサイズが決定され、PVC で設定されます。ストレージドライバによってボリュームがインポートされると、PV は ClaimRef を使用して PVC

に作成されます。再利用ポリシーは、最初には設定されています `retain` PVにあります。Kubernetes が PVC と PV を正常にバインドすると、再利用ポリシーがストレージクラスの再利用ポリシーに合わせて更新されます。ストレージクラスの再利用ポリシーが `delete` の場合、PVが削除されるとストレージボリュームが削除されます。

を使用してボリュームがインポートされる場合 `--no-manage` 引数として、Tridentはオブジェクトのライフサイクルに関してPVCまたはPVに対する追加の操作を実行しません。TridentはのPVイベントとPVCイベントを無視するため `--no-manage` オブジェクト。PVを削除してもストレージボリュームは削除されません。ボリュームのクローンやサイズ変更などの他の処理も無視されます。このオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、Kubernetes 以外でストレージボリュームのライフサイクルを管理する場合に便利です。

PVC と PV にアノテーションが追加されます。このアノテーションは、ボリュームがインポートされたこと、および PVC と PV が管理されていることを示す二重の目的を果たします。このアノテーションは変更または削除しないでください。

Trident 19.07 以降では、PVS の添付ファイルを処理し、ボリュームをインポートの一環としてマウントします。以前のバージョンの Astra Trident を使用しているインポートの場合、データパスに処理は存在しないため、ボリュームをマウントできるかどうかはボリュームインポートで検証されません。ストレージクラスが正しくない場合など、ボリュームのインポートでミスが発生した場合は、PVの再利用ポリシーを変更することでリカバリできます `retain` をクリックしてPVCとPVを削除し、`volume import` コマンドを再実行します。

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

を使用して作成した各ボリューム `ontap-nas` driverはONTAP クラスタ上のFlexVol です。を使用してFlexVol をインポートする `ontap-nas` ドライバも同じように動作します。ONTAP クラスタにすでに存在するFlexVol は、としてインポートできます `ontap-nas` PVC。同様に、FlexGroup ボリュームはとしてインポートできます `ontap-nas-flexgroup` PVC



Trident がインポートする ONTAP のタイプは RW である必要があります。DP タイプのボリュームは SnapMirror デスティネーションボリュームです。Trident にボリュームをインポートする前に、ミラー関係を解除する必要があります。



。 `ontap-nas` ドライバで `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。

たとえば、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` では、次のコマンドを使用します。

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

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

という名前のボリュームをインポートします unmanaged_volume（上 ontap_nas backend）を使用します。Tridentは管理しません。次のコマンドを使用します。

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

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

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



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

ontap-san インポート

Astra Trident は、1つの LUN を含む ONTAP SAN FlexVol をインポートすることもできます。これはと同じです ontap-san ドライバ。FlexVol 内の各PVCおよびLUNにFlexVol を作成します。を使用できます tridentctl import 他の場合と同様にコマンドを実行します。

- の名前を含めます ontap-san バックエンド：

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



管理対象外のをインポートする場合 `ontap-san` ボリューム：FlexVol 内の LUN の名前がになっていることを確認します `lun0` とは、目的のイニシエータを含む `igroup` にマッピングされている。Trident が管理対象のインポートに対して自動的に処理します。

次に、Astra Trident が FlexVol をインポートし、PVC 定義に関連付けます。Astra Trident は、FlexVol の名前もに変更します `pvc-<uuid>` および FlexVol 内の LUN をからにフォーマットします `lun0`。



既存のアクティブな接続がないボリュームをインポートすることを推奨します。アクティブに使用されているボリュームをインポートする場合は、最初にボリュームをクローニングしてからインポートを実行します。

例

をインポートします `ontap-san-managed` にある FlexVol `ontap_san_default` バックエンドでを実行します `tridentctl import` コマンドの形式：

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

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



ONTAP ボリュームのタイプが RW であることが Astra Trident でインポートされる必要があります。DP タイプのボリュームは SnapMirror デスティネーションボリュームです。ボリュームを Astra Trident にインポートする前に、ミラー関係を解除する必要があります。

element インポート

Trident を使用して、NetApp Element ソフトウェア / NetApp HCI ボリュームを Kubernetes クラスタにインポートできます。必要に応じて、Astra Trident バックエンドの名前、ボリュームと PVC ファイルの一意の名前をの引数として指定します `tridentctl import` コマンドを実行します

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

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



Element ドライバではボリューム名の重複がサポートされます。ボリューム名が重複している場合、Trident のボリュームインポートプロセスはエラーを返します。回避策として、ボリュームをクローニングし、一意のボリューム名を指定します。次に、クローンボリュームをインポートします。

gcp-cvs インポート



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

をインポートします gcp-cvs バックエンドのボリュームの名前はです gcpcvs_YEppr を指定します `adroit-jolly-swift` では、次のコマンドを使用します。

```
tridentctl import volume gcpcvs_YEppr adroit-jolly-swift -f <path-to-pvc-file> -n trident
```

```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
|          NAME          | SIZE | STORAGE CLASS |
+-----+-----+-----+-----+
|          BACKEND UUID  | STATE | MANAGED |
+-----+-----+-----+-----+
| pvc-a46ccab7-44aa-4433-94b1-e47fc8c0fa55 | 93 GiB | gcp-storage | file
+-----+-----+-----+-----+
| e1a6e65b-299e-4568-ad05-4f0a105c888f | online | true      |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```



ボリュームパスは、/ のあとのボリュームのエクスポートパスの部分です。たとえば、エクスポートパスがの場合などです 10.0.0.1:/adroit-jolly-swift、ボリュームのパスはです adroit-jolly-swift。

azure-netapp-files インポート

をインポートします azure-netapp-files バックエンドのボリュームの名前はです
azurenetaappfiles_40517 を指定します `importvol1` を使用して、次のコマンドを実行します。

```
tridentctl import volume azurenetaappfiles_40517 importvol1 -f <path-to-pvc-file> -n trident
```

NAME	SIZE	STORAGE CLASS
pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab	100 GiB	anf-storage
file	1c01274f-d94b-44a3-98a3-04c953c9a51e	online



ANF ボリュームのボリュームパスは、/ のあとのマウントパスにあります。たとえば、マウントパスがの場合などです 10.0.0.2:/importvol1、ボリュームのパスはです importvol1。

ネームスペース間でNFSボリュームを共有します

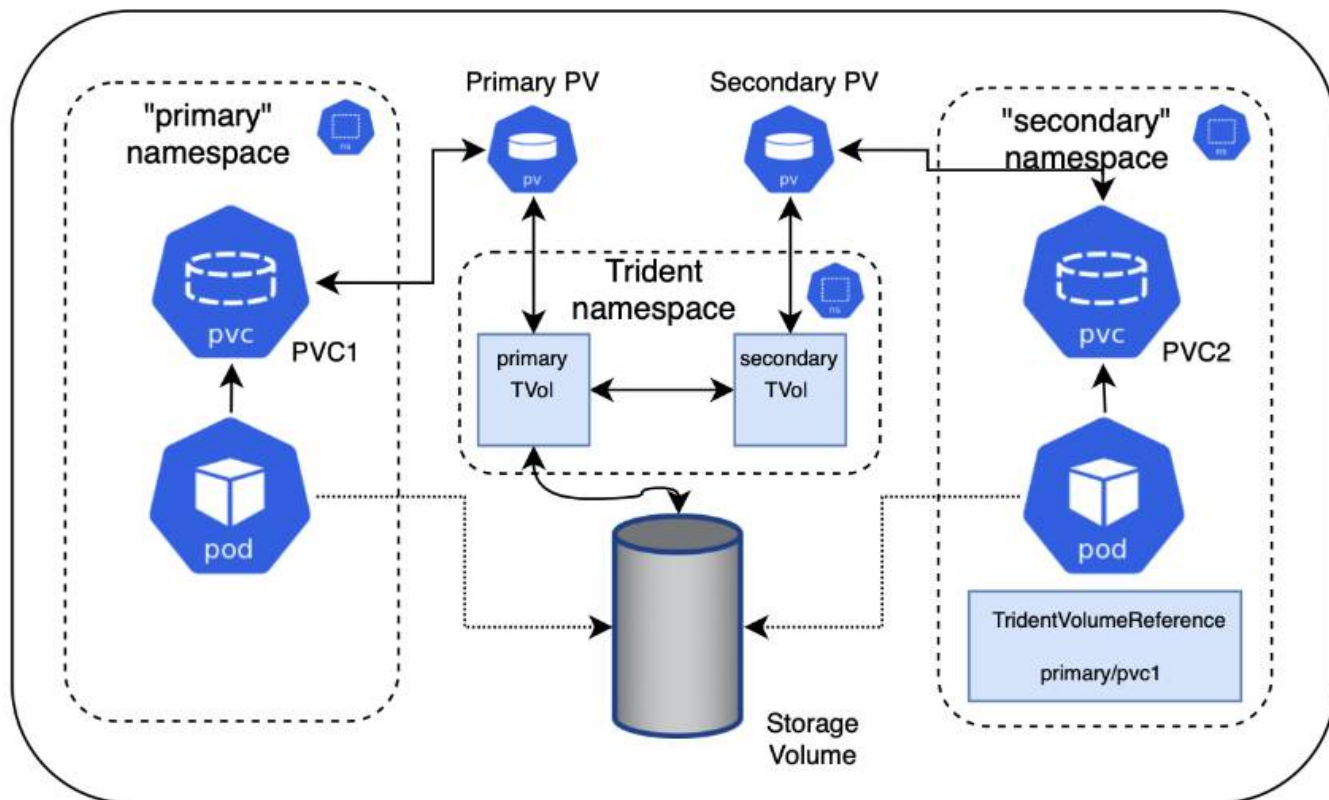
Tridentを使用すると、プライマリネームスペースにボリュームを作成し、1つ以上のセカンダリネームスペースで共有できます。

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kubecfgを作成して、デスティネーションネームスペースの所有者にTridentVolumeReference CRを作成する権限を付与します。
3. *デスティネーションネームスペース所有者：*ソースネームスペースを参照するデスティネーションネームスペースにTridentVolumeReference CRを作成します pvc1。

```

apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1

```

4. 宛先名前空間の所有者：PVCを作成します (pvc2) をデスティネーションネームスペースに展開します (namespace2)を使用します shareFromPVC 送信元PVCを指定する注釈。

```

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi

```



宛先PVCのサイズは、送信元PVCのサイズ以下である必要があります。

結果

Astra Tridentがを読み取り shareFromPVC デスティネーションPVCにアノテーションを設定し、ソースPVを参照するストレージリソースを持たない下位のボリュームとしてデスティネーションPVを作成し、ソースPVストレージリソースを共有します。宛先PVCとPVは、通常どおりバインドされているように見えます。

共有ボリュームを削除

複数のネームスペースで共有されているボリュームは削除できます。Tridentが、ソースネームスペースのボリュームへのアクセスを削除し、ボリュームを共有する他のネームスペースへのアクセスを維持します。ボリュームを参照するすべてのネームスペースが削除されると、Astra Tridentによってボリュームが削除されます。

使用 tridentctl get 下位のボリュームを照会する

を使用する[tridentctl ユーティリティを使用すると、を実行できます get コマンドを使用して下位のボリュームを取得します。詳細については、リンク:./trident-reference/tridentctl.htmlを参照してください

[tridentctl コマンドとオプション]。

```
Usage:
  tridentctl get [option]
```

フラグ：

- `-h, --help`：ボリュームのヘルプ。
- `--parentOfSubordinate string`：クエリを下位のソースボリュームに制限します。
- `--subordinateOf string`：クエリをボリュームの下位に制限します。

制限

- Astra Tridentでは、デスティネーションネームスペースが共有ボリュームに書き込まれるのを防ぐことはできません。共有ボリュームのデータの上書きを防止するには、ファイルロックなどのプロセスを使用する必要があります。
- を削除しても、送信元PVCへのアクセスを取り消すことはできません `shareToNamespace` または `shareFromNamespace` 注釈またはを削除します `TridentVolumeReference` CR。アクセスを取り消すには、下位PVCを削除する必要があります。
- Snapshot、クローン、およびミラーリングは下位のボリュームでは実行できません。

を参照してください。

ネームスペース間のボリュームアクセスの詳細については、次の資料を参照してください。

- にアクセスします ["ネームスペース間でのボリュームの共有：ネームスペース間のボリュームアクセスを許可する場合は「Hello」と入力します"](#)。
- のデモをご覧ください ["ネットアップTV"](#)。

Astra Trident を監視

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

Astra Trident が提供する指標を使用すると、次のことが可能になります。

- Astra Trident の健全性と設定を保持処理が成功した方法と、想定どおりにバックエンドと通信できるかどうかを調べることができます。
- バックエンドの使用状況の情報を調べて、バックエンドでプロビジョニングされているボリュームの数や消費されているスペースなどを確認します。
- 利用可能なバックエンドにプロビジョニングされたボリュームの量のマッピングを維持します。
- パフォーマンスを追跡する。Astra Trident がバックエンドと通信して処理を実行するのにどれくらいの時間がかかるかを調べることができます。



デフォルトでは、Tridentの指標はターゲットポートで公開されています 8001 で /metrics エンドポイント。これらの指標は、Trident のインストール時にデフォルトで * 有効になります。

必要なもの

- Astra Trident がインストールされた Kubernetes クラスタ
- Prometheus インスタンス。これは a である場合もある ["コンテナ化された Prometheus 環境"](#) または、Prometheus をとして実行することもできます ["ネイティブアプリケーション"](#)。

手順 1 : Prometheus ターゲットを定義する

Prometheus ターゲットを定義して指標を収集し、Astra Trident が管理するバックエンド、作成するボリュームなどの情報を取得する必要があります。これ ["ブログ"](#) Prometheus と Grafana を Astra Trident とともに使用して指標を取得する方法について説明します。ブログでは、Kubernetes クラスタでオペレータとして Prometheus を実行する方法と、Astra Trident のメトリックを取得する ServiceMonitor を作成する方法について説明しています。

手順 2 : Prometheus ServiceMonitor を作成します

Tridentの指標を利用するには、を監視するPrometheus ServiceMonitorを作成する必要があります trident-csi サービスおよびリッスン metrics ポート：ServiceMonitor のサンプルは次のようになります。

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

このServiceMonitor定義は、から返されたメトリックを取得します trident-csi サービスとは、を特に探します metrics サービスのエンドポイント。その結果、PrometheusはAstra Tridentの指標：

Astra Tridentから直接取得できる指標に加えて、kubeletは多くの指標を公開しています kubelet_volume_* 独自の指標エンドポイントを使用した指標。Kubelet では、接続されているボリュームに関する情報、および

ポッドと、それが処理するその他の内部処理を確認できます。を参照してください ["こちらをご覧ください"](#)。

ステップ 3 : **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 の健全性情報を取得

- **Astra Trident** からの **HTTP 2XX** 応答の割合

```
(sum (trident_rest_ops_seconds_total_count{status_code=~"2.."} OR on()  
vector(0)) / sum (trident_rest_ops_seconds_total_count)) * 100
```

- **Astra Trident** からのステータスコードによる **REST** 応答の割合

```
(sum (trident_rest_ops_seconds_total_count) by (status_code) / scalar  
(sum (trident_rest_ops_seconds_total_count))) * 100
```

- **Astra Trident** によって実行された処理の平均時間 (ミリ秒)

```
sum by (operation)  
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by  
(operation)  
(trident_operation_duration_milliseconds_count{success="true"})
```

Astra Trident の使用状況に関する情報を入手

- 平均体積サイズ

```
trident_volume_allocated_bytes/trident_volume_count
```

- 各バックエンドによってプロビジョニングされた合計ボリューム容量

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

個々のボリュームの使用状況を取得する



これは、kubelet 指標も収集された場合にのみ有効になります。

- 各ボリュームの使用済みスペースの割合

```
kubelet_volume_stats_used_bytes / kubelet_volume_stats_capacity_bytes * 100
```

Astra Trident AutoSupport の計測データ

デフォルトでは、Astra Trident は Prometheus 指標と基本バックエンド情報を毎日定期的にネットアップに送信します。

- Astra TridentからPrometheus指標や基本バックエンド情報がネットアップに送信されないようにするには、を渡します `--silence-autosupport` Astra Tridentのインストール中にフラグを付ける。
- Tridentからネットアップサポートにコンテナログをオンデマンドで送信することもできます `tridentctl send autosupport`。Astra Trident をトリガーしてログをアップロードする必要があります。ログを送信する前に、ネットアップのに同意する必要があります ["プライバシーポリシー"](#)。
- 指定しないと、Astra Trident は過去 24 時間からログを取得します。
- ログの保持期間はで指定できます `--since` フラグ。例： `tridentctl send autosupport --since=1h`。この情報は、を介して収集および送信されます `trident-autosupport` コンテナこれはAstra Tridentと一緒にインストールされます。コンテナイメージは、で取得できます ["Trident AutoSupport の略"](#)。
- Trident AutoSupport は、個人情報（PII）や個人情報を収集または送信しません。それにはが付いています ["EULA"](#) これは Trident コンテナイメージ自体には該当しません。ネットアップのデータセキュリティと信頼に対する取り組みの詳細を確認できます ["こちらをご覧ください"](#)。

Astra Trident から送信されるペイロードの例を次に示します。

```
{
  "items": [
    {
      "backendUUID": "ff3852e1-18a5-4df4-b2d3-f59f829627ed",
      "protocol": "file",
      "config": {
        "version": 1,
        "storageDriverName": "ontap-nas",
        "debug": false,
        "debugTraceFlags": null,
        "disableDelete": false,
        "serialNumbers": [
          "nwkvzfanek_SN"
        ],
        "limitVolumeSize": ""
      },
      "state": "online",
      "online": true
    }
  ]
}
```

- AutoSupport メッセージは、ネットアップの AutoSupport エンドポイントに送信されます。コンテナイメージの格納にプライベートレジストリを使用している場合は、`--image-registry` フラグ。
- インストール YAML ファイルを生成してプロキシ URL を設定することもできます。これは、`--generate-custom-yaml` を使用して実行できます `tridentctl install --generate-custom-yaml` YAML ファイルを作成し、`--proxy-url` の引数 `trident-autosupport` にコンテナがあります `trident-deployment.yaml`。

Astra Trident の指標を無効化

****メトリックがレポートされないようにするには、を使用してカスタムYAMLを生成する必要があります**
`--generate-custom-yaml` フラグを付けて編集し、`--metrics` に対する呼び出し元からのフラグ `trident-main`
 コンテナ：

Trident for Docker が必要です

導入の前提条件

Trident を導入するには、必要なプロトコルをホストにインストールして設定しておく必要があります。

要件を確認します

- の導入がすべてを満たしていることを確認します "要件"。
- サポートされているバージョンの Docker がインストールされていることを確認します。Docker のバージョンが最新でない場合は、"[インストールまたは更新します](#)"。

```
docker --version
```

- プロトコルの前提条件がホストにインストールされ、設定されていることを確認します。

プロトコル	オペレーティングシステム	コマンド
NFS	RHEL / CentOS 7.	<code>sudo yum install -y nfs-utils</code>
NFS	Ubuntu	<code>sudo apt-get install -y nfs-common</code>

プロトコル	オペレーティングシステム	コマンド
iSCSI	RHEL / CentOS 7.	<ol style="list-style-type: none"> 次のシステムパッケージをインストールします。 <pre>sudo yum install -y lsscsi iscsi-initiator- utils sg3_utils device- mapper-multipath</pre> 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。 <pre>rpm -q iscsi-initiator- utils</pre> スキャンを手動に設定： <pre>sudo sed -i 's/^\(node.session.scan \).*\/\1 = manual/' /etc/iscsi/iscsid.conf</pre> マルチパスを有効化： <pre>sudo mpathconf --enable --with_multipathd y --find_multipaths n</pre> <div data-bbox="1115 1150 1455 1415">  <p>確認します etc/multipath.conf が含まれます find_multipaths no の下 defaults。</p> </div> を確認します iscsid および multipathd 実行中： <pre>sudo systemctl enable --now iscsid multipathd</pre> を有効にして開始します iscsi： <pre>sudo systemctl enable --now iscsi</pre>

プロトコル	オペレーティングシステム	コマンド
iSCSI	Ubuntu	<ol style="list-style-type: none"> 1. 次のシステムパッケージをインストールします。 <pre>sudo apt-get install -y open-iscsi lsscsi sg3- utils multipath-tools scsitools</pre> 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。 <pre>dpkg -l open-iscsi</pre> 3. スキャンを手動に設定： <pre>sudo sed -i 's/^\(node.session.scan \).*\/\1 = manual/' /etc/iscsi/iscsid.conf</pre> 4. マルチパスを有効化： <pre>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</pre> <div>  <p>確認します etc/multipath.conf が含まれます find_multipaths no の下 defaults。</p> </div> 5. を確認します open-iscsi および multipath-tools 有効になっていて実行中： <pre>sudo systemctl status multipath-tools</pre>

Astra Trident を導入

Astra Trident for Docker は、ネットアップのストレージプラットフォーム向けの Docker エコシステムとの直接統合を実現します。ストレージプラットフォームから Docker ホストまで、ストレージリソースのプロビジョニングと管理をサポートします。また、将来プラットフォームを追加するためのフレームワークもサポートします。

Astra Trident の複数のインスタンスを同じホストで同時に実行できます。これにより、複数のストレージシステムとストレージタイプへの同時接続が可能になり、Docker ボリュームに使用するストレージをカスタマイズできます。

必要なもの

を参照してください ["導入の前提条件"](#)。前提条件を満たしていることを確認したら、Astra Trident を導入する準備ができました。

Docker Managed Plugin メソッド（バージョン 1.13 / 17.03 以降）



作業を開始する前に

従来のデーモン方式で Astra Trident 以前の Docker 1.13 / 17.03 を使用していた場合は、マネージドプラグイン方式を使用する前に Astra Trident プロセスを停止し、Docker デーモンを再起動してください。

1. 実行中のインスタンスをすべて停止します。

```
killall /usr/local/bin/netappdvp
killall /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、を指定すると、選択した任意の名前を使用できます。`config` オプションを指定してファイル名を指定します。構成ファイルはに格納されている必要があります。`etc/netappdvp` ホストシステム上のディレクトリ。

- log-level:ログレベルを指定します (debug、info、warn、error、fatal) 。デフォルトはです info。
- debug:デバッグロギングを有効にするかどうかを指定します。デフォルトは false です。true の場合、ログレベルを上書きします。
 - i. 構成ファイルの場所を作成します。

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

- ii. 構成ファイルを作成します

```
cat << EOF > /etc/netappdvp/config.json
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "secret",
    "aggregate": "aggr1"
}
EOF
```

2. マネージドプラグインシステムを使用して Astra Trident を起動交換してください <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/v22.10.0/trident-  
installer-22.10.0.tar.gz  
tar xzf trident-installer-22.10.0.tar.gz
```

- b. ビンパス内の場所に移動します。

```
sudo mv trident-installer/extras/bin/trident /usr/local/bin/  
sudo chown root:root /usr/local/bin/trident  
sudo chmod 755 /usr/local/bin/trident
```

- c. 構成ファイルの場所を作成します。

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

- d. 構成ファイルを作成します

```
cat << EOF > /etc/netappdvp/ontap-nas.json
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "secret",
  "aggregate": "aggr1"
}
EOF
```

- バイナリを配置して構成ファイルを作成したら、必要な構成ファイルを使用して Trident デーモンを開始します。

```
sudo trident --config=/etc/netappdvp/ontap-nas.json
```



指定しないかぎり、ボリュームドライバのデフォルト名は「netapp」です。

デーモンが開始されたら、Docker CLI インターフェイスを使用してボリュームを作成および管理できます

- ボリュームを作成します

```
docker volume create -d netapp --name trident_1
```

- 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
```

- 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。このファイルをCentOS 7 / RHELで使

用するには、次の手順を実行します。

1. ファイルを正しい場所にコピーします。

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

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

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

```
systemctl daemon-reload
```

4. サービスを有効にします。

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

```
systemctl enable trident
```

5. サービスを開始します。

```
systemctl start trident
```

6. ステータスを確認します。

```
systemctl status trident
```



単位ファイルを変更する場合は、を実行します `systemctl daemon-reload` 変更を認識するためのコマンド。

Astra Trident をアップグレードまたはアンインストールする

使用中のボリュームに影響を与えることなく、Astra Trident for Docker を安全にアップグレードできます。アップグレードプロセスでは、が短時間実行されます `docker volume` プラグインで指示されたコマンドは正常に実行されず、プラグインが再度実行されるまでアプリケーションはボリュームをマウントできません。ほとんどの場合、これは秒の問題です。

アップグレード

Astra Trident for Docker をアップグレードするには、次の手順を実行します。

手順

1. 既存のボリュームを表示します。

```
docker volume ls
DRIVER          VOLUME NAME
netapp:latest   my_volume
```

2. プラグインを無効にします。

```
docker plugin disable -f netapp:latest
docker plugin ls
ID                NAME          DESCRIPTION
ENABLED
7067f39a5df5     netapp:latest nDVP - NetApp Docker Volume
Plugin    false
```

3. プラグインをアップグレードします。

```
docker plugin upgrade --skip-remote-check --grant-all-permissions
netapp:latest netapp/trident-plugin:21.07
```



nDVP に代わる、Astra Trident の 18.01 リリース。から直接アップグレードする必要があります netapp/ndvp-plugin への画像 netapp/trident-plugin イメージ (Image) :

4. プラグインを有効にします。

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

5. プラグインが有効になっていることを確認します。

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

6. ボリュームが表示されることを確認します。

```
docker volume ls
DRIVER          VOLUME NAME
netapp:latest   my_volume
```



古いバージョンの Astra Trident（20.10 より前）から Astra Trident 20.10 以降にアップグレードすると、エラーが発生する場合があります。詳細については、[を参照してください "既知の問題"](#)。このエラーが発生した場合は、まずプラグインを無効にしてからプラグインを削除し、次に追加のconfigパラメータを渡して、必要なAstra Tridentバージョンをインストールします。 `docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant-all-permissions config=config.json`

をアンインストールします

Astra Trident for Docker をアンインストールするには、次の手順を実行します。

手順

1. プラグインで作成されたボリュームをすべて削除します。
2. プラグインを無効にします。

```
docker plugin disable netapp:latest
docker plugin ls
ID                NAME                DESCRIPTION
ENABLED
7067f39a5df5     netapp:latest       nDVP - NetApp Docker Volume
Plugin    false
```

3. プラグインを削除します。

```
docker plugin rm netapp:latest
```

ボリュームを操作します

標準のを使用すると、ボリュームを簡単に作成、クローニング、および削除できます `docker volume` 必要に応じてAstra Tridentドライバ名を指定したコマンド。

ボリュームを作成します

- デフォルトの名前を使用して、ドライバでボリュームを作成します。

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

- 特定の Astra Trident インスタンスを使用してボリュームを作成します。

```
docker volume create -d ntap_bronze --name bronzeVolume
```



何も指定しない場合 "オプション (Options)"、ドライバのデフォルトが使用されます。

- デフォルトのボリュームサイズを上書きします。次の例を参照して、ドライバで 20GiB ボリュームを作成してください。

```
docker volume create -d netapp --name my_vol --opt size=20G
```



ボリュームサイズは、オプションの単位 (10G、20GB、3TiB など) を含む整数値で指定します。単位を指定しない場合、デフォルトは g です。サイズの単位は、2 の累乗 (B、KiB、MiB、GiB、TiB) または 10 の累乗 (B、KB、MB、GB、TB) のいずれかです。略記単位では、2 の累乗が使用されます (G=GiB、T=TiB、...)。

ボリュームを削除します

- 他の Docker ボリュームと同様にボリュームを削除します。

```
docker volume rm firstVolume
```



を使用する場合 solidfire-san driver、上記の例では、ボリュームを削除およびパージします。

Astra Trident for Docker をアップグレードするには、次の手順を実行します。

ボリュームのクローンを作成します

を使用する場合 ontap-nas、ontap-san、solidfire-san および `gcp-cvs storage drivers` Trident がボリュームをクローニングできます。を使用する場合 `ontap-nas-flexgroup` または ontap-nas-economy ドライバ、クローニングはサポートされていません。既存のボリュームから新しいボリュームを作成すると、新しい Snapshot が作成されます。

- ボリュームを調べて Snapshot を列挙します。

```
docker volume inspect <volume_name>
```

- 既存のボリュームから新しいボリュームを作成します。その結果、新しい Snapshot が作成されます。

```
docker volume create -d <driver_name> --name <new_name> -o
from=<source_docker_volume>
```

- ボリューム上の既存の Snapshot から新しいボリュームを作成します。新しい Snapshot は作成されません。

```
docker volume create -d <driver_name> --name <new_name> -o
from=<source_docker_volume> -o fromSnapshot=<source_snap_name>
```

例

```
docker volume inspect firstVolume
```

```
[
  {
    "Driver": "ontap-nas",
    "Labels": null,
    "Mountpoint": "/var/lib/docker-volumes/ontap-
nas/netappdvp_firstVolume",
    "Name": "firstVolume",
    "Options": {},
    "Scope": "global",
    "Status": {
      "Snapshots": [
        {
          "Created": "2017-02-10T19:05:00Z",
          "Name": "hourly.2017-02-10_1505"
        }
      ]
    }
  }
]
```

```
docker volume create -d ontap-nas --name clonedVolume -o from=firstVolume
clonedVolume
```

```
docker volume rm clonedVolume
```

```
docker volume create -d ontap-nas --name volFromSnap -o from=firstVolume
-o fromSnapshot=hourly.2017-02-10_1505
volFromSnap
```

```
docker volume rm volFromSnap
```

外部で作成されたボリュームにアクセス

Tridentを使用すると、外部で作成されたブロックデバイス（またはそのクローン）にTrident *からアクセスできます。Tridentは、パーティションがなく、Astra Tridentでサポートされているファイルシステム（など）の場合にのみ利用できます ext4-フォーマット済み /dev/sdc1 Astra Trident経由ではアクセスできません）。

ドライバ固有のボリュームオプション

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

ボリューム作成処理では、これらのオプションを簡単に使用できます。を使用して、オプションと値を指定します -o CLI処理中の演算子。これらは、JSON 構成ファイルの同等の値よりも優先されます。

ONTAP ボリュームのオプション

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

オプション	説明
size	ボリュームのサイズ。デフォルトは 1GiB です。
spaceReserve	ボリュームをシンプロビジョニングまたはシックプロビジョニングします。デフォルトはシンです。有効な値はです none（シンプロビジョニング）および volume（シックプロビジョニング）。
snapshotPolicy	Snapshot ポリシーが目的の値に設定されます。デフォルトはです none`つまり、ボリュームに対してSnapshotが自動的に作成されることはありません。ストレージ管理者によって変更されていない限り、「 default 」という名前のポリシーがすべての ONTAP システムに存在し、 6 個の時間単位 Snapshot 、 2 個の日単位 Snapshot 、および 2 個の週単位 Snapshot を作成して保持します。Snapshotに保存されているデータは、にアクセスしてリカバリできます ` .snapshot ボリューム内の任意のディレクトリ内のディレクトリ。

オプション	説明
snapshotReserve	これにより、Snapshot リザーブの割合が希望する値に設定されます。デフォルト値は no で、Snapshot ポリシーを選択した場合は ONTAP によって snapshotReserve が選択されます（通常は 5%）。Snapshot ポリシーがない場合は 0% が選択されます。構成ファイルのすべての ONTAP バックエンドに対して snapshotReserve のデフォルト値を設定できます。また、この値は、ONTAP-NAS-エコノミーを除くすべての ONTAP バックエンドでボリューム作成オプションとして使用できます。
splitOnClone	ボリュームをクローニングすると、そのクローンが原因 ONTAP によって親から即座にスプリットされます。デフォルトは false。クローンボリュームのクローニングは、作成直後に親からクローンをスプリットする方法を推奨します。これは、ストレージ効率化の効果がまったくないためです。たとえば、空のデータベースをクローニングすると、時間を大幅に節約できますが、ストレージの節約はほとんどできないため、クローンをすぐに分割することをお勧めします。
encryption	<p>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。このオプションを使用するには、クラスターで 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。</p> <p>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。</p> <p>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"。</p>
tieringPolicy	ボリュームに使用する階層化ポリシーを設定します。これにより、アクセス頻度の低いコールドデータをクラウド階層に移動するかどうかが決まります。

以下は、NFS * のみ * 用の追加オプションです。

オプション	説明
unixPermissions	これにより、ボリューム自体の権限セットを制御できます。デフォルトでは、権限はに設定されます `---rwxr-xr-x` または数値表記 0755、およびです `root` が所有者になります。テキスト形式または数値形式のどちらかを使用できます。

オプション	説明
snapshotDir	これをに設定します true がを作成します .snapshot ボリュームにアクセスしているクライアントから認識できるディレクトリ。デフォルト値は false`の可視性を意味します ` .snapshot ディレクトリはデフォルトで無効になっています。公式のMySQLイメージなどの一部のイメージは、の場合、期待どおりに機能しません .snapshot ディレクトリが表示されます。
exportPolicy	ボリュームで使用するエクスポートポリシーを設定します。デフォルトはです default。
securityStyle	ボリュームへのアクセスに使用するセキュリティ形式を設定します。デフォルトはです unix。有効な値はです unix および mixed。

以下の追加オプションは、iSCSI * のみ * 用です。

オプション	説明
fileSystemType	iSCSI ボリュームのフォーマットに使用するファイルシステムを設定します。デフォルトはです ext4。有効な値はです ext3、 ext4`および ` xfs。
spaceAllocation	これをに設定します false LUNのスペース割り当て機能をオフにします。デフォルト値はです `true`つまり、ボリュームのスペースが不足し、ボリューム内のLUNに書き込みを受け付けられなくなったときに、ONTAP からホストに通知されます。また、このオプションで ONTAP、ホストでデータが削除された時点での自動スペース再生も有効になります。

例

以下の例を参照してください。

- 10GiB ボリュームを作成します。

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

- Snapshot を使用して 100GiB のボリュームを作成します。

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

- setuid ビットが有効になっているボリュームを作成します。

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

最小ボリュームサイズは 20MiB です。

Snapshot リザーブが指定されていない場合、Snapshot ポリシーはです `none` Trident は 0% の Snapshot リザーブを使用します。

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

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

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

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

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

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

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

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

Element ソフトウェアのボリュームオプション

Element ソフトウェアのオプションでは、ボリュームに関連付けられているサービス品質（QoS）ポリシーのサイズと QoS を指定できます。ボリュームの作成時に、関連付けられている QoS ポリシーを使用して指定します `-o type=service_level 名称`。

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

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

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

例

QoS 定義を含む次のサンプル構成ファイルを参照してください。

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

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

- 10GiB の Gold ボリュームを作成します。

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

- 100GiB Bronze ボリュームを作成します。

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

CVS on GCP ボリュームのオプション

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

オプション	説明
size	ボリュームのサイズ。デフォルトは100GiBです。
serviceLevel	ボリュームの CVS サービスレベル。デフォルトは「Standard」です。有効な値は、standard、premium、extreme です。
snapshotReserve	これにより、Snapshot リザーブの割合が希望する値に設定されます。デフォルト値は no で、CVS によって Snapshot リザーブが選択されます（通常は 0%）。

例

- 2TiB のボリュームを作成します。

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

- 5TiB の Premium ボリュームを作成します。

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

最小ボリュームサイズは100GiBです。

Azure NetApp Files ボリュームのオプション

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

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

例

- 200GiBのボリュームを作成します。

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

最小ボリュームサイズは 100GB です。

ログを収集します

トラブルシューティングに役立つログを収集できます。ログの収集方法は、Docker プラグインの実行方法によって異なります。

トラブルシューティング用にログを収集する

手順

1. 推奨される管理プラグイン方法（を使用）でAstra Tridentを実行している場合 docker plugin コマンド）で表示される情報は次のとおりです。

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

標準的なロギングレベルでは、ほとんどの問題を診断できます。十分でない場合は、デバッグロギングをイネーブルにできます。

2. デバッグロギングをイネーブルにするには、デバッグロギングをイネーブルにしてプラグインをインストールします。

```
docker plugin install netapp/trident-plugin:<version> --alias <alias>
debug=true
```

または、プラグインがすでにインストールされている場合にデバッグログを有効にします。

```
docker plugin disable <plugin>
docker plugin set <plugin> debug=true
docker plugin enable <plugin>
```

3. ホスト上でバイナリ自体を実行している場合、ログはホストので使用できます `/var/log/netappdvp` ディレクトリ。デバッグロギングを有効にするには、を指定します `-debug` プラグインを実行すると、

一般的なトラブルシューティングのヒント

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

```
Error response from daemon: dial unix /run/docker/plugins/<id>/netapp.sock:
connect: no such file or directory
```

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

- PVをコンテナにマウントする際に問題が発生する場合は、を確認してください `rpcbind` をインストールして実行しておきます。ホストOSに必要なパッケージマネージャを使用して、かどうかを確認します `rpcbind` を実行しています。 `rpcbind` サービスのステータスを確認するには、を実行します `systemctl status rpcbind` またはそれと同等のものです。

複数の Astra Trident インスタンスを管理

複数のストレージ構成を同時に使用する必要がある場合は、Trident の複数のインスタンスが必要です。複数のインスタンスを作成するには、を使用して異なる名前を付けます `--alias` オプションにコンテナ化プラグインを指定するか、を指定します `--volume-driver` ホストでTridentをインスタンス化する際のオプション。

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

1. エイリアスと構成ファイルを指定して、最初のインスタンスを起動します。

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

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

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

3. ドライバ名としてエイリアスを指定するボリュームを作成します。

たとえば、gold ボリュームの場合：

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

たとえば、Silver ボリュームの場合：

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

従来の（バージョン **1.12** 以前）の場合の手順

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

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

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

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

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

たとえば、NFS の場合：

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

たとえば、iSCSI の場合：

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

ストレージ構成オプション

Astra Trident 構成で利用できる設定オプションを確認してください。

グローバル構成オプション

以下の設定オプションは、使用するストレージプラットフォームに関係なく、すべての Astra Trident 構成に適用されます。

オプション	説明	例
version	構成ファイルのバージョン番号	1.
storageDriverName	ストレージドライバの名前	ontap-nas、ontap-san、ontap-nas-economy、ontap-nas-flexgroup、solidfire-san、azure-netapp-files`または`gcp-cvs
storagePrefix	ボリューム名のオプションのプレフィックス。デフォルト：「netappdvp_」。	ステージング _
limitVolumeSize	ボリュームサイズに関するオプションの制限。デフォルト：「」（適用されていない）	10G



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

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

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

ONTAP構成

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

オプション	説明	例
managementLIF	ONTAP 管理 LIF の IP アドレス。Fully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定できます。	10.0.0.1

オプション	説明	例
dataLIF	プロトコル LIF の IP アドレス。指定しない場合に生成されます。をクリックします <code>ontap-nas</code> ドライバ*のみ*、FQDNを指定できます。この場合、FQDNがNFSマウント処理に使用されます。をクリックします <code>ontap-san</code> ドライバのデフォルトでは、SVMのすべてのデータLIF IPが使用され、iSCSI マルチパスが使用されます。IP アドレスの指定 <code>dataLIF</code> をクリックします <code>ontap-san</code> ドライバは、マルチパスを無効にして、指定されたアドレスだけを使用します。	10.0.0.2
svm	使用する Storage Virtual Machine (管理 LIF がクラスタ LIF である場合は必須)	SVM_NFS の場合
username	ストレージデバイスに接続するユーザ名	vsadmin
password	ストレージ・デバイスに接続するためのパスワード	秘密
aggregate	プロビジョニング用のアグリゲート (オプション。設定する場合は SVM に割り当てる必要があります)。をクリックします <code>ontap-nas-flexgroup</code> ドライバ。このオプションは無視されます。SVM に割り当てられたすべてのアグリゲートを使用して FlexGroup ボリュームがプロビジョニングされます。	アグリゲート1
limitAggregateUsage	オプション。使用率がこの割合を超えている場合は、プロビジョニングを失敗させます	75%
nfsMountOptions	NFS マウントオプションのきめ細かな制御。デフォルトは「 <code>-o nfsvers=3</code> 」です。でのみ使用できます <code>ontap-nas</code> および <code>ontap-nas-economy</code> ドライバ。 "ここでは、NFS ホストの設定情報を参照してください" 。	<code>-o nfsvers=4</code>

オプション	説明	例
igroupName	プラグインで使用する igroup 。デフォルトは「netappdvp」です。* 「ONTAP-SAN'd river」のみ利用可能です。	myigroup と入力します
limitVolumeSize	最大要求可能ボリュームサイズと qtree 親ボリュームサイズ。*のため ontap-nas-economy また、このオプションを使用すると、作成するFlexVol *のサイズも制限されます。	300 g
qtreesPerFlexvol	FlexVol あたりの最大 qtree 数は [50 、 300] の範囲で指定する必要があります。デフォルトは 200 です。 *のため ontap-nas-economy ドライバ。このオプションを使用すると、FlexVol あたりの最大qtree数をカスタマイズできます。	300

作成するすべてのボリュームでデフォルトのオプションを指定しなくても済むようになっています。

オプション	説明	例
spaceReserve	スペースリザーベーションモード : 「none」 (シンプロビジョニング) または「volume」 (シック)	なし
snapshotPolicy	使用する Snapshot ポリシー。デフォルトは「none」です。	なし
snapshotReserve	Snapshot リザーブの割合。ONTAP のデフォルトを受け入れる場合は、デフォルトで「」になります	10
splitOnClone	作成時にクローンを親からスプリットします。デフォルトは「false」です。	いいえ

オプション	説明	例
encryption	<p>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 <code>false</code>。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。</p> <p>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。</p> <p>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"。</p>	正しいです
unixPermissions	プロビジョニングされた NFS ボリューム用の NAS オプション。デフォルトは「777」	777
snapshotDir	にアクセスするためのNASオプション <code>.snapshot</code> ディレクトリ、デフォルトは「false」	正しいです
exportPolicy	NFS エクスポートポリシーで使用する NAS オプション。デフォルトは「default」	デフォルト
securityStyle	プロビジョニングされた NFS ボリュームにアクセスするための NAS オプション（デフォルトは「UNIX」）	混在
fileSystemType	SAN オプション：ファイルシステムタイプを選択します。デフォルトは「ext4」です。	XFS
tieringPolicy	使用する階層化ポリシー。デフォルトは「none」です。ONTAP 9.5 より前の SVM-DR 構成では「snapshot-only」です	なし

スケーリングオプション

。 `ontap-nas` および `ontap-san` ドライバによって、DockerボリュームごとにONTAP FlexVol が作成されます。ONTAP では、クラスタノードあたり最大 1、000 個の FlexVol がサポートされます。クラスタの最大 FlexVol 数は 12、000 です。この制限内にDockerボリュームの要件が収まる場合は、を参照してください

ontap-nas FlexVolで提供されるDockerボリューム単位のSnapshotやクローニングなどの機能が追加されているため、NAS解決策 がドライバとして推奨されます。

FlexVol の制限で対応できない数のDockerボリュームが必要な場合は、を選択します ontap-nas-economy または ontap-san-economy ドライバ。

。 ontap-nas-economy ドライバによって、自動管理されるFlexVolのプール内に、DockerボリュームがONTAP qtreeとして作成される。qtree の拡張性は、クラスタノードあたり最大 10、000、クラスタあたり最大 2、40、000 で、一部の機能を犠牲にすることで大幅に向上しています。。 ontap-nas-economy ドライバは、Dockerボリューム単位のスナップショットやクローニングをサポートしていません。



。 ontap-nas-economy ドライバは現在Docker Swarmではサポートされていません。Swarm は複数のノード間でのボリューム作成のオーケストレーションを行わないためです。

。 ontap-san-economy ドライバによって、自動で管理されるFlexVolの共有プール内にDockerボリュームがONTAP LUNとして作成される。この方法により、各 FlexVol が 1 つの LUN に制限されることはなく、SAN ワークロードのスケラビリティが向上します。ストレージレイに依拠して、ONTAP はクラスタあたり最大 16384 個の LUN をサポートします。このドライバは、ボリュームが下位の LUN であるため、Docker ボリューム単位の Snapshot とクローニングをサポートします。

を選択します ontap-nas-flexgroup 数十億個のファイルを含むペタバイト規模に拡張可能な1つのボリュームへの並列処理能力を高めるドライバ。FlexGroup のユースケースとしては、AI / ML / DL、ビッグデータと分析、ソフトウェアのビルド、ストリーミング、ファイルリポジトリなどが考えられます。Trident は、FlexGroup ボリュームのプロビジョニング時に SVM に割り当てられたすべてのアグリゲートを使用します。Trident での FlexGroup のサポートでは、次の点も考慮する必要があります。

- ONTAP バージョン 9.2 以降が必要です。
- 本ドキュメントの執筆時点では、FlexGroup は NFS v3 のみをサポートしています。
- SVM で 64 ビットの NFSv3 ID を有効にすることを推奨します。
- 推奨される最小 FlexGroup サイズは 100GB です。
- FlexGroup Volume ではクローニングはサポートされていません。

FlexGroup と FlexGroup に適したワークロードの詳細については、を参照してください "[NetApp FlexGroup Volume Best Practices and Implementation Guide](#)"。

同じ環境で高度な機能と大規模な拡張性を実現するために、を使用して、Docker Volume Pluginの複数のインスタンスを実行できます ontap-nas を使用しています ontap-nas-economy。

ONTAP 構成ファイルの例

- NFSの例 ontap-nas ドライバ*

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "secret",
  "aggregate": "aggr1",
  "defaults": {
    "size": "10G",
    "spaceReserve": "none",
    "exportPolicy": "default"
  }
}
```

• NFSの例 ontap-nas-flexgroup ドライバ*

```
{
  "version": 1,
  "storageDriverName": "ontap-nas-flexgroup",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "secret",
  "defaults": {
    "size": "100G",
    "spaceReserve": "none",
    "exportPolicy": "default"
  }
}
```

• NFSの例 ontap-nas-economy ドライバ*

```
{
  "version": 1,
  "storageDriverName": "ontap-nas-economy",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "secret",
  "aggregate": "aggr1"
}
```

• iSCSIの例 ontap-san ドライバ*

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi",
  "username": "vsadmin",
  "password": "secret",
  "aggregate": "aggr1",
  "igroupName": "myigroup"
}
```

• NFSの例 ontap-san-economy ドライバ*

```
{
  "version": 1,
  "storageDriverName": "ontap-san-economy",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi_eco",
  "username": "vsadmin",
  "password": "secret",
  "aggregate": "aggr1",
  "igroupName": "myigroup"
}
```

Element ソフトウェアの設定

Element ソフトウェア（NetApp HCI / SolidFire）を使用する場合は、グローバルな設定値のほかに、以下のオプションも使用できます。

オプション	説明	例
Endpoint	\https : //<login> : <password>@<mvip>/ JSON -RPC /<element-version>	https://admin:admin@192.168.160.3/json-rpc/8.0
SVIP	iSCSI の IP アドレスとポート	10.0.0.7 : 3260
TenantName	使用する SolidFire テナント（見つからない場合に作成）	Docker です
InitiatorIFace	iSCSI トラフィックをデフォルト以外のインターフェイスに制限する場合は、インターフェイスを指定します	デフォルト
Types	QoS の仕様	以下の例を参照してください
LegacyNamePrefix	アップグレードされた Trident インストールのプレフィックス。1.3.2 より前のバージョンの Trident を使用していて、既存のボリュームをアップグレードする場合は、この値を設定して、ボリューム名メソッドを使用してマッピングされた古いボリュームにアクセスする必要があります。	「netappdvp -」

。 solidfire-san ドライバは Docker Swarm をサポートしていません。

Element ソフトウェア構成ファイルの例


```
{
  "version": 1,
  "storageDriverName": "solidfire-san",
  "Endpoint": "https://admin:admin@192.168.160.3/json-rpc/8.0",
  "SVIP": "10.0.0.7:3260",
  "TenantName": "docker",
  "InitiatorIFace": "default",
  "Types": [
    {
      "Type": "Bronze",
      "Qos": {
        "minIOPS": 1000,
        "maxIOPS": 2000,
        "burstIOPS": 4000
      }
    },
    {
      "Type": "Silver",
      "Qos": {
        "minIOPS": 4000,
        "maxIOPS": 6000,
        "burstIOPS": 8000
      }
    },
    {
      "Type": "Gold",
      "Qos": {
        "minIOPS": 6000,
        "maxIOPS": 8000,
        "burstIOPS": 10000
      }
    }
  ]
}
```

GCP 上の Cloud Volumes Service (CVS) 構成

Astra Tridentは、デフォルトのCVSサービスタイプがonに設定されたボリュームをサポートしています **"GCP"**。Astra Tridentは、CVSサービスタイプで許可されている最小値に関係なく、100GiB未満のCVSボリュームをサポートしていません。したがって、Tridentは要求されたボリュームが最小サイズよりも小さい場合、自動的に100GiBのボリュームを作成します。

GCP で CVS を使用する場合は、グローバル構成の値に加えて、次のオプションも使用できます。

オプション	説明	例
apiRegion	CVS アカ운트리ージョン（必須）。は、このバックエンドがボリュームをプロビジョニングする GCP リージョンです。	「 us-west2 」
projectNumber	GCP プロジェクト番号（必須）。GCP Web ポータルのホームページにあります。	“ 123456789012 ”
hostProjectNumber	GCP 共有 VPC ホストプロジェクト番号（共有 VPC を使用する場合は必須）	「 098765432109 」
apiKey	CVS admin ロールを持つ GCP サービスアカウントの API キー（必須）。は、GCP サービスアカウントの秘密鍵ファイルの JSON 形式のコンテンツです（バックエンド構成ファイルにそのままコピーされます）。サービスアカウントには netappcloudvolumes .admin ロールが必要です。	（秘密鍵ファイルの内容）
secretKey	CVS アカountのシークレットキー（必須）。CVS Web ポータルの「アカウント設定」>「API アクセス」で確認できます。	デフォルト
proxyURL	CVS アカountへの接続にプロキシサーバが必要な場合は、プロキシ URL を指定します。プロキシサーバには、HTTP プロキシまたは HTTPS プロキシを使用できます。HTTPS プロキシの場合、証明書の検証は省略され、プロキシサーバで自己署名証明書が使用されるようになります。* 認証が有効になっているプロキシサーバはサポートされていません*。	「http://proxy-server-hostname/”
nfsMountOptions	NFS マウントオプション。デフォルトは「-o nfsvers=3」です。	「 nfsvers=3 、 proto=tcp 、 timeo=600 」
serviceLevel	パフォーマンスレベル（標準、プレミアム、エクストリーム）、デフォルトは「標準」	Premium サービス


```
/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kw  
s8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY  
9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc  
zZsrtrHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHi  
sIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOgu  
SaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyA  
ZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz  
lzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3  
bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4  
Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5o  
jY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznH  
czZsrtrHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtr  
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbO  
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKe  
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRA  
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzlzZE4j  
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlWgLwGa==\n-----END PRIVATE  
KEY-----\n",  
    "client_email": "cloudvolumes-admin-sa@my-gcp-  
project.iam.gserviceaccount.com",  
    "client_id": "123456789012345678901",  
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",  
    "token_uri": "https://oauth2.googleapis.com/token",  
    "auth_provider_x509_cert_url":  
"https://www.googleapis.com/oauth2/v1/certs",  
    "client_x509_cert_url":  
"https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-  
sa%40my-gcp-project.iam.gserviceaccount.com"  
},  
    "proxyURL": "http://proxy-server-hostname/"  
}
```

Azure NetApp Files 構成

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

- subscriptionID Azure NetApp Files を有効にしたAzureサブスクリプションから選択します
- tenantID、clientID、および clientSecret から "アプリケーション登録" Azure Active Directory で、 Azure NetApp Files サービスに対する十分な権限がある
- Azure ロケーションに少なくとも 1 つ以上が含まれている "委任されたサブネット"



初めて Azure NetApp Files を使用している場合や、新しい場所で使用している場合は、そのような初期設定が必要になります ["クイックスタートガイド"](#) ご案内します。



Astra Trident 21.04.0 以前では、手動 QoS 容量プールはサポートされていません。

オプション	説明	デフォルト
version	常に 1	
storageDriverName	「 azure-NetApp-files 」	
backendName	ストレージバックエンドのカスタム名	ドライバ名 + "_" + ランダムな文字
subscriptionID	Azure サブスクリプションのサブスクリプション ID	
tenantID	アプリケーション登録からのテナント ID	
clientID	アプリケーション登録からのクライアント ID	
clientSecret	アプリケーション登録からのクライアントシークレット	
serviceLevel	「 Standard 」、「 Premium 」、「 Ultra 」のいずれか	「 (ランダム) 」
location	新しいボリュームを作成する Azure の場所の名前を指定します	「 (ランダム) 」
virtualNetwork	委任されたサブネットを持つ仮想ネットワークの名前	「 (ランダム) 」
subnet	に委任されたサブネットの名前 Microsoft.Netapp/volumes	「 (ランダム) 」
nfsMountOptions	NFS マウントオプションのきめ細かな制御	「 -o nfsvers=3 」
limitVolumeSize	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します	"" (デフォルトでは適用されません)



Azure NetApp Files サービスでは、サイズが 100GB 未満のボリュームはサポートされません。Trident では、アプリケーションの導入を容易にするために、より小さいボリュームが要求された場合に、100GB のボリュームが自動的に作成されます。

これらのオプションを使用して、構成の特別なセクションで各ボリュームをデフォルトでプロビジョニングす

る方法を制御できます。

オプション	説明	デフォルト
exportRule	新しいボリュームのエクスポートルール。CIDR 表記の IPv4 アドレスまたは IPv4 サブネットの任意の組み合わせをカンマで区切って指定する必要があります。	「0.0.0.0/0」
snapshotDir	の表示/非表示を制御します .snapshot ディレクトリ	いいえ
size	新しいボリュームのデフォルトサイズ	「100G」

Azure NetApp Files 構成の例

- 例 1：azure-NetApp-files* のバックエンドの最小構成

これは、バックエンドの絶対的な最小構成です。この構成では、Trident がお客様のネットアップアカウント、容量プール、および ANF に委譲されたサブネットをすべて検出し、新しいボリュームをいずれかの場所にランダムに配置します。

この構成は、ANF の利用を開始して問題を解決するのに役立ちます。しかし実際には、プロビジョニングするボリュームの範囲を追加して、必要な特性を確実に持ち、それを使用しているコンピューティングに近いネットワーク上で終了するようにします。詳細については、以降の例を参照してください。

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

- 例 2：Azure NetApp Files の単一の場所と特定のサービスレベル *

このバックエンド構成では、Azure の「eastus」ロケーションにボリュームを「Premium」容量プールに配置します。Trident は、ANF に委任されているすべてのサブネットを自動的に検出し、いずれかのサブネットに新しいボリュームをランダムに配置します。

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

• 例 3 : azure-NetApp-files* の高度な設定

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

```
{
  "version": 1,
  "storageDriverName": "azure-netapp-files",
  "subscriptionID": "9f87c765-4774-fake-ae98-a721add45451",
  "tenantID": "68e4f836-edc1-fake-bff9-b2d865ee56cf",
  "clientID": "dd043f63-bf8e-fake-8076-8de91e5713aa",
  "clientSecret": "SECRET",
  "location": "eastus",
  "serviceLevel": "Premium",
  "virtualNetwork": "my-virtual-network",
  "subnet": "my-subnet",
  "nfsMountOptions": "nfsvers=3,proto=tcp,timeo=600",
  "limitVolumeSize": "500Gi",
  "defaults": {
    "exportRule": "10.0.0.0/24,10.0.1.0/24,10.0.2.100",
    "size": "200Gi"
  }
}
```

• 例 4 : azure-NetApp-files* を使用する仮想ストレージプール

このバックエンド構成では、複数の定義され **ストレージのプール** 1 つのファイルに格納できます。これは、異なるサービスレベルをサポートする複数の容量プールがあり、それらを表すストレージクラスを Kubernetes で作成する場合に便利です。

仮想ストレージプールの機能の表面に、ラベルが貼られています。

```
{
  "version": 1,
  "storageDriverName": "azure-netapp-files",
  "subscriptionID": "9f87c765-4774-fake-ae98-a721add45451",
  "tenantID": "68e4f836-edc1-fake-bff9-b2d865ee56cf",
  "clientID": "dd043f63-bf8e-fake-8076-8de91e5713aa",
  "clientSecret": "SECRET",
  "nfsMountOptions": "nfsvers=3,proto=tcp,timeo=600",
  "labels": {
    "cloud": "azure"
  },
  "location": "eastus",

  "storage": [
    {
      "labels": {
        "performance": "gold"
      },
      "serviceLevel": "Ultra"
    },
    {
      "labels": {
        "performance": "silver"
      },
      "serviceLevel": "Premium"
    },
    {
      "labels": {
        "performance": "bronze"
      },
      "serviceLevel": "Standard",
    }
  ]
}
```

既知の問題および制限事項

Astra Trident と Docker を使用する際の既知の問題と制限事項について説明しています。

Trident Docker Volume Plugin を旧バージョンから **20.10** 以降にアップグレードすると、該当するファイルエラーまたはディレクトリエラーなしでアップグレードが失敗します。

回避策

1. プラグインを無効にします。

```
docker plugin disable -f netapp:latest
```

2. プラグインを削除します。

```
docker plugin rm -f netapp:latest
```

3. 追加を指定してプラグインを再インストールします config パラメータ

```
docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant  
-all-permissions config=config.json
```

ボリューム名は **2 文字以上**にする必要があります。



これは Docker クライアントの制限事項です。クライアントは、1 文字の名前を Windows パスと解釈します。"[バグ 25773](#) を参照"。

Docker Swarm には、**Astra Trident** がストレージやドライバのあらゆる組み合わせでサポートしないようにする一定の動作があります。

- Docker Swarm は現在、ボリューム ID ではなくボリューム名を一意的ボリューム識別子として使用します。
- ボリューム要求は、Swarm クラスタ内の各ノードに同時に送信されます。
- ボリュームプラグイン（Astra Trident を含む）は、Swarm クラスタ内の各ノードで個別に実行する必要があります。
これは、ONTAP の仕組みとの仕組みによるものです ontap-nas および ontap-san ドライバ機能は、これらの制限内で動作できるようになる唯一の機能です。

その他のドライバには、競合状態などの問題があります。このような問題が発生すると、ボリュームを同じ名前で異なる ID にする機能が Element に備わっているため、「勝者」を明確にせずに 1 回の要求で大量のボリュームを作成できるようになります。

ネットアップは Docker チームにフィードバックを提供しましたが、今後の変更の兆候はありません。

FlexGroup をプロビジョニングする場合、プロビジョニングする **FlexGroup** と共通のアグリゲートが **2 つ目の FlexGroup** に **1 つ以上**あると、**ONTAP** は **2 つ目の FlexGroup** をプロビジョニングしません。

よくある質問

Trident が提供する Astra のインストール、設定、アップグレード、トラブルシューティングに関する FAQ を掲載しています。

一般的な質問

Trident がリリースされる頻度を教えてください。

Trident は、1 月、4 月、7 月、10 月の 3 カ月ごとにリリースされます。Kubernetes のリリースから 1 カ月後です。

Astra Trident は、特定のバージョンの **Kubernetes** でリリースされたすべての機能をサポートしていますか。

Astra Trident は、通常、Kubernetes でアルファ機能をサポートしていません。Trident は、Kubernetes ベータリリースに続く 2 つの Trident リリースでベータ機能をサポートしています。

Astra Trident には、他のネットアップ製品との依存関係はありますか。

Astra Trident は、他のネットアップソフトウェア製品に依存しないため、スタンドアロンアプリケーションとして機能します。ただし、ネットアップのバックエンドストレージデバイスが必要です。

Astra Trident の設定の詳細をすべて取得するにはどうすればよいですか。

を使用します `tridentctl get` コマンドを使用して、Astra Trident 構成に関する詳細を確認できます。

Astra Trident を使用してストレージをプロビジョニングする方法に関するメトリクスを取得できますか。

はい。Trident 20.01 には、管理対象のバックエンドの数、プロビジョニングされたボリュームの数、消費されたバイト数など、Astra Trident の動作に関する情報を収集するために使用できる Prometheus エンドポイントが導入されています。また、Cloud Insights を使用して監視と分析を行うこともできます。

Astra Trident を **CSI** プロビジョニング担当者として使用すると、ユーザエクスペリエンスは変化しますか。

いいえユーザエクスペリエンスと機能に関する変更はありません。使用されるプロビジョニングツール名は `csi.trident.netapp.io`。現在および将来のリリースで提供される新しい機能をすべて使用する場合は、Astra Trident をインストールする方法を推奨します。

Kubernetes クラスタに **Astra Trident** をインストールして使用

のサポート対象のバージョンを指定します `etcd`？

Astra Trident はもう必要ありません `etcd`。状態を維持するために CRD を使用します。

Astra Trident はプライベートレジストリからのオフラインインストールをサポートしていますか。

はい、Astra Trident はオフラインでインストールできます。を参照してください ["こちらをご覧ください"](#)。

Astra Trident はリモートからインストールできますか。

はい。Astra Trident 18.10以降では、を搭載した任意のマシンからリモートインストール機能がサポートされます `kubectl` クラスタへのアクセス。実行後 `kubectl` アクセスが検証されます（「開始」など） `kubectl get nodes` リモートマシンからコマンドを実行して確認）、インストール手順に従います。

Astra Trident でハイアベイラビリティを構成できますか。

Astra Trident は、1つのインスタンスで Kubernetes Deployment（ReplicaSet）としてインストールされるため、HA が組み込まれています。導入環境内のレプリカの数が増やすべきではありません。Astra Trident がインストールされているノードが失われた場合や、ポッドにアクセスできない場合は、Kubernetes によって、クラスタ内の正常なノードにポッドが自動的に再導入されます。Astra Trident はコントロールプレーンのみであるため、Astra Trident を再導入しても、現在マウントされているポッドには影響しません。

Astra Trident は `kube-system` ネームスペースにアクセスする必要がありますか。

Astra Trident は Kubernetes API Server からデータを読み取り、アプリケーションが新しい PVC を要求するタイミングを判断して、`kube-system` へのアクセスを必要とします。

Astra Trident で使用されるロールと権限を教えてください。

Trident インストーラが Kubernetes ClusterRole を作成します。このロールには、Kubernetes クラスタの PersistentVolume、PersistentVolumeClaim、StorageClass、Secret の各リソースへのアクセス権があります。を参照してください ["こちらをご覧ください"](#)。

Astra Trident がインストールに使用するマニフェストファイルをローカルで生成できますか。

必要に応じて、マニフェストファイルである Astra Trident のインストールに使用するものをローカルで生成して変更できます。を参照してください ["こちらをご覧ください"](#)。

2 つの別々の Kubernetes クラスタに対して、同じ ONTAP バックエンド SVM を 2 つの別々の Astra Trident インスタンスに対して共有できますか。

推奨されませんが、同じバックエンド SVM を 2 つの Astra Trident インスタンスに使用できます。インストール時に各インスタンスに一意的なボリューム名を指定するか、一意のボリューム名を指定します `StoragePrefix` のパラメータを指定します `setup/backend.json` ファイル。これは、両方のインスタンスで同じ FlexVol を使用しないためです。

ContainerLinux（旧 CoreOS）に Astra Trident をインストールすることはできますか。

Astra Trident は Kubernetes ポッドとして機能し、Kubernetes が実行されている場所に導入できます。

ネットアップの **Cloud Volumes ONTAP** で **Astra Trident** を使用できますか。

はい、Astra Trident は AWS 、 Google Cloud 、 Azure でサポートされています。

Astra Trident は **Cloud Volume** サービスと連携していますか。

はい。Astra Trident は、Azure の Azure NetApp Files サービスと GCP の Cloud Volumes Service をサポートしています。

トラブルシューティングとサポート

ネットアップは **Astra Trident** をサポートしていますか。

Astra Trident はオープンソースであり、無償で提供されますが、ネットアップのバックエンドがサポートされていれば、完全にサポートされています。

サポートケースを作成するにはどうすればよいですか？

サポートケースを作成するには、次のいずれかを実行します。

1. サポートアカウントマネージャーに連絡して、チケットの発行に関するサポートを受けてください。
2. 連絡してサポートケースを作成します ["ネットアップサポート"](#)。

サポートログバンドルを生成するにはどうすればよいですか？

を実行すると、サポートバンドルを作成できます `tridentctl logs -a`。バンドルでキャプチャされたログに加えて、kubelet ログをキャプチャして、Kubernetes 側のマウントの問題を診断します。kubelet ログの取得手順は、Kubernetes のインストール方法によって異なります。

新しい機能のリクエストを発行する必要がある場合は、どうすればよいですか。

に問題を作成します ["Astra Trident Github"](#) そして、概要の件名と問題に「* RFE *」と明記してください。

不具合を発生させる場所

に問題を作成します ["Astra Trident Github"](#)。問題に関連する必要なすべての情報とログを記録しておいてください。

ネットアップが **Trident** の **Astra** について簡単に質問できたらどうなりますか。コミュニティやフォーラムはありますか？

ご質問、ご質問、ご要望がございましたら、ネットアップのアストラからお問い合わせください ["チャンネルを外します"](#) またはGitHub。

ストレージシステムのパスワードが変更され、**Astra Trident** が機能しなくなった場合、どのように回復すればよいですか。

バックエンドのパスワードをで更新します `tridentctl update backend myBackend -f`

</path/to_new_backend.json> -n trident。交換してください myBackend この例では、バックエンド名にとを指定しています ` /path/to_new_backend.json` と入力します backend.json ファイル。

Astra Trident が Kubernetes ノードを検出できない。この問題を解決するにはどうすればよいですか

Trident が Kubernetes ノードを検出できない場合、次の 2 つのケースが考えられます。Kubernetes または DNS 問題内のネットワーク問題が原因の場合もあります。各 Kubernetes ノードで実行される Trident ノードのデモモンが Trident コントローラと通信し、Trident にノードを登録する必要があります。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のメジャーリリースへのアップグレードをサポートしています。バージョン 18.xx から 19.xx、19.xx から 20.xx にアップグレードできます。本番環境の導入前に、ラボでアップグレードをテストする必要があります。

Trident を以前のリリースにダウングレードできますか。

ダウングレードする場合は、いくつかの要因を評価する必要があります。を参照してください ["ダウングレードに関するセクション"](#)。

バックエンドとボリュームを管理

ONTAP バックエンド定義ファイルに管理 LIF とデータ LIF の両方を定義する必要がありますか。

バックエンド定義ファイルには両方を指定することを推奨します。必須の管理 LIF は 1 つだけです。

Astra Trident が ONTAP バックエンドに CHAP を設定できるか。

はい。20.04 以降、Astra Trident は ONTAP バックエンドに対して双方向 CHAP をサポートします。これには設定が必要です `useCHAP=true` バックエンド構成

Astra Trident を使用してエクスポートポリシーを管理するにはどうすればよいですか。

Astra Trident では、バージョン 20.04 以降からエクスポートポリシーを動的に作成、管理できます。これにより、ストレージ管理者はバックエンド構成に 1 つ以上の CIDR ブロックを指定でき、Trident では、その範囲に含まれるノード IP を作成したエクスポートポリシーに追加できます。このようにして、Astra Trident は特

定の CIDR 内に IP アドレスが割り当てられたノードのルールの追加と削除を自動的に管理します。この機能には CSI Trident が必要です。

データ LIF にポートを指定できるか。

Astra Trident 19.01 以降では、DataLIF にポートを指定できます。で設定します backend.json ファイルの形式 "managementLIF": <ip address>:<port>"。たとえば、管理LIFのIPアドレスが192.0.2.1で、ポートが1000の場合、を設定します "managementLIF": "192.0.2.1:1000"。

管理 LIF とデータ LIF に IPv6 アドレスを使用できますか。

はい。Astra Trident 20.01 は、ONTAP バックエンドの管理 LIF パラメータとデータ LIF パラメータに対して IPv6 アドレスを定義できます。アドレスがIPv6のセマンティクスに従い、管理LIFが角かっこで囲まれて定義されていることを確認します（例：[ec0d:6504:a9c1:ae67:53d1:4bdf:ab32:e233]）。また、を使用して Astra Trident をインストールしておく必要があります --use-ipv6 IPv6 で動作するためのフラグ。

バックエンドの管理 LIF を更新できますか。

はい、を使用してバックエンドの管理LIFを更新できます tridentctl update backend コマンドを実行します

バックエンドのデータ LIF を更新できるか。

いいえ、バックエンドのデータ LIF を更新できません。

Kubernetes 向け Astra Trident で複数のバックエンドを作成できますか。

Astra Trident では、同じドライバまたは別々のドライバを使用して、多数のバックエンドを同時にサポートできます。

Astra Trident はバックエンドクレデンシャルをどのように保存しますか。

Astra Trident では、バックエンドのクレデンシャルを Kubernetes のシークレットとして格納します。

Astra Trident ではどのようにして特定のバックエンドを選択しますか。

バックエンド属性を使用してクラスに適したプールを自動的に選択できない場合は、を参照してください storagePools および additionalStoragePools パラメータは、特定のプールセットを選択するために使用します。

Astra Trident が特定のバックエンドからプロビジョニングされないようにするにはどうすればよいですか。

。excludeStoragePools パラメータを使用して、一連のプールをフィルタします。一連のプールがTrident からプロビジョニングに使用され、一致するプールは削除されます。

同じ種類のバックエンドが複数ある場合、**Astra Trident** はどのバックエンドを使用するかをどのように選択しますか。

同じタイプのバックエンドが複数設定されている場合、Astra Tridentはにあるパラメータに基づいて適切なバックエンドを選択します StorageClass および PersistentVolumeClaim。たとえば、ONTAPとNASのドライババックエンドが複数ある場合、Astra Tridentは内のパラメータを照合しようとします StorageClass および PersistentVolumeClaim に記載された要件を提供できるバックエンドを組み合わせることができます StorageClass および PersistentVolumeClaim。この要求に一致するバックエンドが複数ある場合、Astra Trident はいずれかのバックエンドからランダムに選択します。

Astra Trident は、**Element / SolidFire** で双方向 **CHAP** をサポートしていますか。

はい。

Trident が **ONTAP** ボリュームに **qtree** を導入する方法を教えてください。1 つのボリュームに配置できる **qtree** の数はいくつですか。

。ontap-nas-economy ドライバは、同じFlexVol に最大200個のqtreeを作成し（50~300で設定可能）、クラスタノードあたり100、000個のqtreeを、クラスタあたり240万個まで作成します。をクリックします PersistentVolumeClaim これは、エコノミードライバが対応しているため、ドライバは新しいqtreeを処理できるFlexVol がすでに存在するかどうかを調べます。qtree を提供できる FlexVol が存在しない場合は、新しい FlexVol が作成されます。

ONTAP NAS でプロビジョニングされたボリュームに **UNIX** アクセス権を設定するにはどうすればよいですか。

Astra Trident でプロビジョニングしたボリュームに対して UNIX 権限を設定するには、バックエンド定義ファイルにパラメータを設定します。

ボリュームをプロビジョニングする際に、明示的な **ONTAP NFS** マウントオプションを設定するにはどうすればよいですか。

Trident では、デフォルトでマウントオプションが Kubernetes でどの値にも設定されています。Kubernetes ストレージクラスでマウントオプションを指定するには、次の例を実行します ["こちらをご覧ください"](#)。

プロビジョニングしたボリュームを特定のエクスポートポリシーに設定するにはどうすればよいですか？

適切なホストにボリュームへのアクセスを許可するには、を使用します exportPolicy バックエンド定義ファイルで設定されたパラメータ。

ONTAP を使用して **Astra Trident** 経由でボリューム暗号化を設定する方法を教えてください。

Trident によってプロビジョニングされたボリュームで暗号化を設定するには、バックエンド定義ファイルの暗号化パラメータを使用します。詳細については、以下を参照してください。 ["Astra TridentとNVEおよびNAEの相互運用性"](#)

Trident 経由で ONTAP に QoS を実装するには、どのような方法が最適ですか。

使用 StorageClasses ONTAP にQoSを実装するには、次の手順を

Trident 経由でシンプロビジョニングやシックプロビジョニングを指定するにはどうすればよいですか。

ONTAP ドライバは、シンプロビジョニングまたはシックプロビジョニングをサポートします。ONTAP ドライバはデフォルトでシンプロビジョニングに設定されています。シックプロビジョニングが必要な場合は、バックエンド定義ファイルまたはを設定する必要があります StorageClass。両方が設定されている場合は、StorageClass 優先されます。ONTAP で次の項目を設定します。

1. オン StorageClass を設定します `provisioningType` シックとしての属性。
2. バックエンド定義ファイルで、を設定してシックボリュームを有効にします backend spaceReserve parameter ボリュームとして。

誤って PVC を削除した場合でも、使用中のボリュームが削除されないようにするにはどうすればよいですか。

Kubernetes では、バージョン 1.10 以降、PVC 保護が自動的に有効になります。

Astra Trident によって作成された NFS PVC を拡張できますか。

はい。Astra Trident によって作成された PVC を拡張できます。ボリュームの自動拡張は ONTAP の機能であり、Trident には適用されません。

Astra Trident の外部で作成したボリュームを Astra Trident にインポートできますか。

19.04 以降では、ボリュームインポート機能を使用してボリュームを Kubernetes に移行できます。

ボリュームが SnapMirror データ保護（DP）モードまたはオフラインモードの間にインポートできますか。

外部ボリュームが DP モードになっているかオフラインになっている場合、ボリュームのインポートは失敗します。次のエラーメッセージが表示されます。

```
Error: could not import volume: volume import failed to get size of
volume: volume <name> was not found (400 Bad Request) command terminated
with exit code 1.
Make sure to remove the DP mode or put the volume online before importing
the volume.
```

Astra Trident によって作成された iSCSI PVC を拡張できますか。

Trident 19.10 は CSI プロビジョニング担当者を使用した iSCSI PVS の拡張をサポートしています。

リソースクォータをネットアップクラスタに変換する方法

Kubernetes ストレージリソースクォータは、ネットアップストレージの容量があるかぎり機能します。容量不足が原因でネットアップストレージが Kubernetes のクォータ設定を受け入れられない場合、Astra Trident はプロビジョニングを試みますがエラーになります。

Trident を使用してボリューム **Snapshot** を作成できますか。

はい。Trident が、Snapshot からオンデマンドのボリューム Snapshot と永続的ボリュームを作成できるようになりました。スナップショットからPVSを作成するには、を確認してください
VolumeSnapshotDataSource フィーチャーゲートが有効になりました。

Astra Trident のボリュームスナップショットをサポートするドライバを教えてください。

現在のところ、オンデマンドスナップショットがサポートされています `ontap-nas`、`ontap-nas-flexgroup`、`ontap-san`、`ontap-san-economy`、`solidfire-san`、`gcp-cvs` および `azure-netapp-files` バックエンドドライバ

ONTAP を使用して **Astra Trident** でプロビジョニングしたボリュームの **Snapshot** バックアップを作成する方法を教えてください。

これはで入手できます `ontap-nas`、`ontap-san` および `ontap-nas-flexgroup` ドライバ。を指定することもできます `snapshotPolicy` をクリックします `ontap-san-economy` ドライバーは `FlexVol` レベルです。

この機能は、でも使用できます `ontap-nas-economy` ドライバの詳細は、`FlexVol` レベルではなく、`qtree` レベルで表示されます。Astra Tridentによってプロビジョニングされたボリュームのスナップショットを作成できるようにするには、`backend` パラメータオプションを設定します `snapshotPolicy` `ONTAP` バックエンドで定義されている `Snapshot` ポリシーにコピーします。ストレージコントローラで作成された `Snapshot` は Astra Trident で認識されません。

Trident 経由でプロビジョニングしたボリュームの **Snapshot** リザーブの割合を設定できますか。

はい。を設定することで、Astra Tridentを使用して、Snapshotコピーを格納するためのディスクスペースの特定の割合を予約できます `snapshotReserve` バックエンド定義ファイルの属性。を設定している場合は `snapshotPolicy` および `snapshotReserve` バックエンド定義ファイルでは、に従って `Snapshot` リザーブの割合が設定されます `snapshotReserve` バックエンドファイルに指定されている割合。状況に応じて `snapshotReserve` この割合は省略しています。ONTAP ではデフォルトで `Snapshot` リザーブの割合が5に設定されます。状況に応じて `snapshotPolicy` オプションが `none` に設定されている場合、`Snapshot` リザーブの割合は0に設定されます。

ボリュームの **Snapshot** ディレクトリに直接アクセスしてファイルをコピーできますか。

はい。Tridentがプロビジョニングしたボリュームの `Snapshot` ディレクトリにアクセスするには、を設定します `snapshotDir` バックエンド定義ファイルのパラメータ。

Astra Trident を使用して、ボリューム用の **SnapMirror** をセットアップできますか。

現時点では、SnapMirror は ONTAP CLI または OnCommand System Manager を使用して外部に設定する必要があります。

永続ボリュームを特定の **ONTAP Snapshot** にリストアするにはどうすればよいですか？

ボリュームを ONTAP Snapshot にリストアするには、次の手順を実行します。

1. 永続ボリュームを使用しているアプリケーションポッドを休止します。
2. ONTAP CLI または OnCommand システムマネージャを使用して、必要な Snapshot にリバートします。
3. アプリケーションポッドを再起動します。

Tridentは、負荷共有ミラーが設定されている**SVM**でボリュームをプロビジョニングできますか。

負荷共有ミラーは、NFS経由でデータを提供するSVMのルートボリューム用に作成できます。ONTAP は、Tridentによって作成されたボリュームの負荷共有ミラーを自動的に更新します。ボリュームのマウントが遅延する可能性があります。Tridentを使用して複数のボリュームを作成する場合、ボリュームをプロビジョニングする方法は、負荷共有ミラーを更新するONTAP によって異なります。

お客様 / テナントごとにストレージクラスの使用状況を分離するにはどうすればよいですか。

Kubernetes では、ネームスペース内のストレージクラスは使用できません。ただし、Kubernetes を使用すると、ネームスペースごとにストレージリソースクォータを使用することで、ネームスペースごとに特定のストレージクラスの使用量を制限できます。特定のストレージへのネームスペースアクセスを拒否するには、そのストレージクラスのリソースクォータを 0 に設定します。

サポート

Astra Trident は、正式にサポートされているネットアップのプロジェクトです。任意の標準メカニズムを使用してネットアップに連絡し、必要なエンタープライズクラスのサポートを受けることができます。

また、ネットアップのアストラには、コンテナユーザ（Astra Trident開発者を含む）の活気あるパブリックコミュニティもあります ["チャネルを外します"](#)。プロジェクトに関する一般的な質問をしたり、同じような気のある同僚と関連するトピックについて話し合うのには、この場所が最適です。

トラブルシューティング

Astra Trident のインストール中および使用中に発生する可能性のある問題のトラブルシューティングには、ここに記載されているポインタを使用してください。



Astra Tridentのサポートを受けるには、を使用してサポートバンドルを作成してください
`tridentctl logs -a -n trident` に送信します NetApp Support <Getting Help>。



トラブルシューティングに関する記事の包括的なリストについては、を参照してください "[ネットアップナレッジベース \(ログインが必要\)](#)"。また、Astra に関連する問題のトラブルシューティングに関する情報も参照できます "[こちらをご覧ください](#)"。

全般的なトラブルシューティング

- Tridentポッドが正常に起動しないと（Tridentポッドがで停止した場合など） ContainerCreating 準備が完了したコンテナが2つ未満のフェーズ）を実行中であること `kubectl -n trident describe deployment trident` および `kubectl -n trident describe pod trident--**` 詳細な分析情報を提供できます。kubeletログの取得（例：Via `journalctl -xeu kubelet`）また有用である場合もある。
- Tridentのログに十分な情報がない場合は、にアクセスしてTridentのデバッグモードを有効にすることができます `-d` インストールオプションに基づいてインストールパラメータにフラグを設定します。

次に、を使用してデバッグが設定されていることを `./tridentctl logs -n trident` を検索して `level=debug msg` ログに記録されます。

オペレータとともにインストールされます

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

すべての Trident ポッドが再起動されます。これには数秒かかることがあります。これを確認するには、の出力の「経過時間」列を確認します `kubectl get pod -n trident`。

Astra Trident 20.07および20.10では `tprov` の代わりに `torc`。

Helm とともにインストールされます

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

tridentctl を使用してインストールされます

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

- を含めて、各バックエンドのデバッグログを取得することもできます `debugTraceFlags` バックエンドの定義に含まれています。たとえば、と指定します `debugTraceFlags: {"api":true, "method":true,}` TridentログでAPI呼び出しとメソッドの逆数を取得する。既存のバックエンドにはは追加できます `debugTraceFlags` を使用して設定します `tridentctl backend update`。
- Red Hat CoreOSを使用する場合は、次の点を確認してください `iscsid` はワーカーノードで有効になり、デフォルトで開始されます。この設定には、`OpenShift MachineConfig` を使用するか、イグニッションテンプレートを変更します。
- Trident をで使用する際によく発生する問題です ["Azure NetApp Files の特長"](#) テナントとクライアントのシークレットが、必要な権限がないアプリケーションの登録から取得された場合です。Trident の要件の詳細については、を参照してください ["Azure NetApp Files の特長"](#) 設定
- PVをコンテナにマウントする際に問題が発生する場合は、を確認してください `rpcbind` をインストールして実行しておきます。ホストOSに必要なパッケージマネージャを使用して、かどうかを確認します `rpcbind` を実行しています。のステータスを確認できます `rpcbind` を実行してサービスを提供します `systemctl status rpcbind` またはそれと同等のものです。
- Tridentバックエンドがにあると報告した場合 `failed` 以前は対処したことがあるにもかかわらず、状況はバックエンドに関連付けられたSVM /管理者クレデンシャルの変更が原因であると考えられます。を使用したバックエンド情報の更新 `tridentctl update backend` Tridentポッドをバウンスすると、この問題 が修正されます。
- Kubernetes クラスタや Trident をアップグレードしてベータ版のボリューム Snapshot を使用する場合は、既存の alpha スナップショット CRS がすべて削除されていることを確認してください。その後、を使用できます `tridentctl obliviante alpha-snapshot-crd` アルファスナップショット作成の作成を削除するコマンド。を参照してください ["この blog"](#) アルファスナップショットの移行手順を理解する。
- TridentをDockerでコンテナランタイムとしてインストールする際に権限の問題が発生した場合は、Trident のインストールをで試してください `--in cluster=false` フラグ。これはインストーラポッドを使用せず、に起因する許可の問題を回避する `trident-installer` ユーザ：
- を使用します `uninstall parameter <Uninstalling Trident>` 実行に失敗したあとにクリーンアップに使用します。デフォルトでは、スクリプトは Trident によって作成された CRD を削除しないため、実行中の導入環境でも安全にアンインストールしてインストールできます。
- Tridentの以前のバージョンにダウングレードする場合は、最初にを実行します `tridentctl uninstall` Tridentを削除するコマンド。必要なダウンロードします ["Trident のバージョン"](#) を使用してをインストールします `tridentctl install` コマンドを実行します新しい PVS が作成されておらず、既存の PVS / バックエンド / ストレージクラスに変更がない場合にのみ、ダウングレードを検討してください。Tridentは現在、状態を維持するためにSSDを使用しているため、作成されたすべてのストレージエンティティ（バックエンド、ストレージクラス、PVS、ボリュームSnapshot）にはが含まれています `associated CRD objects <Kubernetes CustomResourceDefinition Objects>` 以前にインストールされたTridentのバージョンで使用されていたPVに書き込まれるデータではありません。* 以前のバージョンに戻すと、新しく作成した PVS は使用できなくなります。* バックエンド、PVS、ストレージクラス、ボリュームスナップショット（作成 / 更新 / 削除）などのオブジェクトに加えた変更は、ダウングレード時に Trident に表示されません *。以前のバージョンの Trident で使用されていた PV は、Trident から見ることはできません。以前のバージョンに戻しても、アップグレードされていないかぎり、以前のリリースを使用してすでに作成された PVS へのアクセスは中断されません。
- Tridentを完全に削除するには、を実行します `tridentctl obliviante crd` コマンドを実行しますこれにより、すべての CRD オブジェクトが削除され、CRD が定義解除されます。Trident は、すでにプロビジョニングされている PVS を管理しなくなります。



Trident はその後、最初から再構成する必要があります。

- インストールが成功した後、PVCがにスタックしている場合 Pending 実行中のフェーズ `kubectl describe pvc` TridentがこのPVCのPVのプロビジョニングに失敗した理由について追加情報 に説明できる。

オペレータを使用して失敗した **Trident** の導入をトラブルシューティングします

オペレータを使用してTridentを導入している場合は、ステータスがになります `TridentOrchestrator` からの変更 `Installing` 終了: `Installed`。を確認した場合は `Failed` ステータスが表示され、オペレータが単独でリカバリできない場合は、次のコマンドを実行してオペレータのログを確認する必要があります。

```
tridentctl logs -l trident-operator
```

`trident-operator` コンテナのログの末尾には、問題のある場所を示すことができます。たとえば、このような問題の1つは、エアーギャップ環境のアップストリームレジストリから必要なコンテナイメージをプルできないことです。

Tridentのインストールが失敗した理由を理解するには、次の項目を参照してください: `TridentOrchestrator` ステータス。

```
kubectl describe torc trident-2
Name:          trident-2
Namespace:
Labels:        <none>
Annotations:   <none>
API Version:   trident.netapp.io/v1
Kind:          TridentOrchestrator
...
Status:
  Current Installation Params:
    IPv6:
    Autosupport Hostname:
    Autosupport Image:
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:
    Image Pull Secrets:      <nil>
    Image Registry:
    k8sTimeout:
    Kubelet Dir:
    Log Format:
    Silence Autosupport:
    Trident Image:
  Message:                  Trident is bound to another CR 'trident'
  Namespace:                trident-2
  Status:                   Error
  Version:
Events:
  Type      Reason  Age                From              Message
  ----      -
Warning    Error   16s (x2 over 16s)  trident-operator.netapp.io  Trident
is bound to another CR 'trident'
```

このエラーは、がすでに存在することを示します `TridentOrchestrator`
 これはTridentのインストールに使用された機能です。各Kubernetesクラスタで実行できるのは
 Tridentのインスタンスを1つ使用することで、
 アクティブな状態が1つだけ存在する時間 `TridentOrchestrator` それができること
 を作成します。

また、Trident ポッドのステータスを確認することで、適切でないものがあるかどうかを確認できます。

```
kubectl get pods -n trident
```

NAME	READY	STATUS	RESTARTS
trident-csi-4p5kq 5m18s	1/2	ImagePullBackOff	0
trident-csi-6f45bfd8b6-vfrkw 5m19s	4/5	ImagePullBackOff	0
trident-csi-9q5xc 5m18s	1/2	ImagePullBackOff	0
trident-csi-9v95z 5m18s	1/2	ImagePullBackOff	0
trident-operator-766f7b8658-ldzsv 8m17s	1/1	Running	0

ポッドを完全に初期化できないことがわかります。
1つ以上のコンテナイメージが取得されなかったためです。

問題に対処するには、を編集する必要があります `TridentOrchestrator` CR。
または、を削除することもできます `TridentOrchestrator` をクリックし、新しい
修正された正確な定義を持つもの。

を使用した **Trident** の導入に失敗した場合のトラブルシューティング `tridentctl`

何が問題になったかを確認するには、を使用してインストーラを再実行します `-d` 引数。デバッグモードをオンにして、問題の内容を理解するのに役立ちます。

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

問題に対処したら、次のようにインストールをクリーンアップし、を実行します `tridentctl install` コマンドの再実行：

```
./tridentctl uninstall -n trident
INFO Deleted Trident deployment.
INFO Deleted cluster role binding.
INFO Deleted cluster role.
INFO Deleted service account.
INFO Removed Trident user from security context constraint.
INFO Trident uninstallation succeeded.
```


ベストプラクティスと推奨事項

導入

Astra Trident の導入時には、ここに示す推奨事項を使用してください。

専用のネームスペースに導入します

"[ネームスペース](#)" 異なるアプリケーション間で管理を分離できるため、リソース共有の障壁となります。たとえば、あるネームスペースの PVC を別のネームスペースから使用することはできません。Astra Trident は、Kubernetes クラスタ内のすべてのネームスペースに PV リソースを提供するため、権限が昇格されたサービスアカウントを利用します。

また、Trident ポッドにアクセスすると、ユーザがストレージシステムのクレデンシャルやその他の機密情報にアクセスできるようになります。アプリケーションユーザと管理アプリケーションが Trident オブジェクト定義またはポッド自体にアクセスできないようにすることが重要です。

クォータと範囲制限を使用してストレージ消費を制御します

Kubernetes には、2 つの機能があります。これらの機能を組み合わせることで、アプリケーションによるリソース消費を制限する強力なメカニズムが提供されます。。"[ストレージクォータメカニズム](#)" 管理者は、グローバルおよびストレージクラス固有の、容量とオブジェクト数の使用制限をネームスペース単位で実装できます。さらに、を使用します "[範囲制限](#)" 要求がプロビジョニングツールに転送される前に、PVC 要求が最小値と最大値の両方の範囲内にあることを確認します。

これらの値はネームスペース単位で定義されます。つまり、各ネームスペースに、リソースの要件に応じた値を定義する必要があります。の詳細については、こちらを参照してください "[クォータの活用方法](#)"。

ストレージ構成

ネットアップポートフォリオの各ストレージプラットフォームには、コンテナ化されたアプリケーションやそうでないアプリケーションに役立つ独自の機能があります。

プラットフォームの概要

Trident は ONTAP や Element と連携1つのプラットフォームが他のプラットフォームよりもすべてのアプリケーションとシナリオに適しているわけではありませんが、プラットフォームを選択する際には、アプリケーションのニーズとデバイスを管理するチームを考慮する必要があります。

使用するプロトコルに対応したホストオペレーティングシステムのベースラインベストプラクティスに従う必要があります。必要に応じて、アプリケーションのベストプラクティスを適用する際に、バックエンド、ストレージクラス、PVC の設定を利用して、特定のアプリケーションのストレージを最適化することもできます。

ONTAP と Cloud Volumes ONTAP のベストプラクティス

Trident 向けに ONTAP と Cloud Volumes ONTAP を設定するためのベストプラクティスをご確認ください。

次に示す推奨事項は、Trident によって動的にプロビジョニングされたボリュームを消費するコンテナ化されたワークロード用に ONTAP を設定する際のガイドラインです。それぞれの要件を考慮し、環境内で適切かどうかを評価する必要があります。

Trident 専用の SVM を使用

Storage Virtual Machine (SVM) を使用すると、ONTAP システムのテナントを分離し、管理者が分離できます。SVM をアプリケーション専用にしておくと、権限の委譲が可能になり、リソース消費を制限するためのベストプラクティスを適用できます。

SVM の管理には、いくつかのオプションを使用できます。

- バックエンド構成でクラスタ管理インターフェイスを適切なクレデンシャルとともに指定し、SVM 名を指定します。
- ONTAP System Manager または CLI を使用して、SVM 専用の管理インターフェイスを作成します。
- NFS データインターフェイスで管理ロールを共有します。

いずれの場合も、インターフェイスは DNS にあり、Trident の設定時には DNS 名を使用する必要があります。これにより、ネットワーク ID を保持しなくても SVM-DR などの一部の DR シナリオが簡単になります。

専用の管理 LIF または共有の管理 LIF を SVM に使用する方法は推奨されませんが、ネットワークセキュリティポリシーを選択した方法と一致させる必要があります。最大の柔軟性を確保するには、どのような場合でも DNS 経由で管理 LIF にアクセスできるようにします **"SVM-DR"** Trident と組み合わせて使用できます。

最大ボリューム数を制限します

ONTAP ストレージシステムの最大ボリューム数は、ソフトウェアのバージョンとハードウェアプラットフォームによって異なります。を参照してください **"NetApp Hardware Universe の略"** 具体的な制限については、使用しているプラットフォームと ONTAP のバージョンに対応しています。ボリューム数を使い果たした場合、Trident のプロビジョニング処理だけでなく、すべてのストレージ要求に対してプロビジョニング処理が失敗します。

Trident `ontap-nas` および `ontap-san` ドライバによって、作成された各 Kubernetes Persistent Volume (PV ; 永続ボリューム) 用の FlexVol がプロビジョニングされます。。`ontap-nas-economy` ドライバは、200 PVSごとに約1つのFlexVolを作成します (50~300で構成可能)。。`ontap-san-economy` ドライバは、PVS 100個につきFlexVolを約1つ作成します (50~200の間で設定可能)。Trident がストレージシステム上の使用可能なボリュームをすべて消費しないようにするには、SVM に制限を設定する必要があります。コマンドラインから実行できます。

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

の値 `max-volumes` 環境に固有のいくつかの条件によって異なります。

- ONTAP クラスタ内の既存のボリュームの数
- 他のアプリケーション用に Trident 外部でプロビジョニングするボリュームの数
- Kubernetes アプリケーションで消費されると予想される永続ボリュームの数

。 `max-volumes` 値は、ONTAP クラスタ内のすべてのノードでプロビジョニングされているボリュームの合計であり、個々の ONTAP ノードではプロビジョニングされていません。その結果、ONTAP クラスタノード

の Trident でプロビジョニングされたボリュームの数が、別のノードよりもはるかに多い、または少ない場合があります。

たとえば、2 ノードの ONTAP クラスタでは、最大 2、000 個の FlexVol をホストできます。最大ボリューム数を 1250 に設定していると、非常に妥当な結果が得られます。ただし、のみの場合 **"アグリゲート"** あるノードから SVM に割り当てられている場合や、あるノードから割り当てられたアグリゲートをプロビジョニングできない場合（容量など）は、他のノードが Trident でプロビジョニングされたすべてのボリュームのターゲットになります。つまり、そのノードがボリューム数の上限に達するまでの可能性があります `max-volumes` の値に達したため、そのノードを使用する Trident と他のボリューム処理の両方に影響が生じます。* クラスタ内の各ノードのアグリゲートを、Trident が使用する SVM に同じ番号で確実に割り当てることで、この状況を回避できます。*

Trident で作成できるボリュームの最大サイズを制限

Trident で作成できるボリュームの最大サイズを設定するには、を使用します `limitVolumeSize` のパラメータ `backend.json` 定義（Definition）：

ストレージレイでボリュームサイズを制御するだけでなく、Kubernetes の機能も利用する必要があります。

双方向 CHAP を使用するように Trident を設定します

バックエンド定義で CHAP イニシエータとターゲットのユーザ名とパスワードを指定し、Trident を使用して SVM で CHAP を有効にすることができます。を使用する `useCHAP` バックエンド構成のパラメータである Trident は、CHAP を使用して ONTAP バックエンドの iSCSI 接続を認証します。双方向 CHAP のサポートは Trident 20.04 以降で利用できます。

SVM QoS ポリシーを作成して使用します

SVM に適用された ONTAP QoS ポリシーを使用すると、Trident でプロビジョニングされたボリュームが使用できる IOPS の数が制限されます。これはに役立ちます **"Bully を防止します"** Trident SVM 外のワークロードに影響を及ぼす、制御不能なコンテナ。

SVM の QoS ポリシーはいくつかの手順で作成します。正確な情報については、ご使用の ONTAP バージョンのマニュアルを参照してください。次の例は、SVM で使用可能な合計 IOPS を 5000 に制限する QoS ポリシーを作成します。

```
# create the policy group for the SVM
qos policy-group create -policy-group <policy_name> -vserver <svm_name>
-max-throughput 5000iops

# assign the policy group to the SVM, note this will not work
# if volumes or files in the SVM have existing QoS policies
vserver modify -vserver <svm_name> -qos-policy-group <policy_name>
```

また、使用しているバージョンの ONTAP でサポートされている場合は、最小 QoS を使用してコンテナ化されたワークロードへのスループットを保証することもできます。アダプティブ QoS は SVM レベルのポリシーには対応していません。

コンテナ化されたワークロード専用の IOPS は、さまざまな要素によって異なります。その中には、次のよう

なものがあります。

- ストレージ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 つまたは少数のみに高い IOPS が必要な場合、コンテナ化された他のワークロードに対する Bully になる可能性があります。その場合は、外部の自動化を使用したボリュームごとの QoS ポリシーの割り当てを検討してください。



ONTAP バージョン 9.8 より前の場合は、QoS ポリシーグループを SVM * only * に割り当ててください。

Trident の QoS ポリシーグループを作成

Quality of Service (QoS ; サービス品質) は、競合するワークロードによって重要なワークロードのパフォーマンスが低下しないようにします。ONTAP の QoS ポリシーグループには、ボリュームに対する QoS オプションが用意されており、ユーザは 1 つ以上のワークロードに対するスループットの上限を定義できます。QoS の詳細については、を参照してください ["QoS によるスループットの保証"](#)。

QoS ポリシーグループはバックエンドまたはストレージプールに指定でき、そのプールまたはバックエンドに作成された各ボリュームに適用されます。

ONTAP には、従来型とアダプティブ型の 2 種類の QoS ポリシーグループがあります。従来のポリシーグループは、最大スループット（以降のバージョンでは最小スループット）がフラットに表示されます。アダプティブ QoS では、ワークロードのサイズの変更に合わせてスループットが自動的に調整され、TB または GB あたりの IOPS が一定に維持されます。これにより、何百何千という数のワークロードを管理する大規模な環境では大きなメリットが得られます。

QoS ポリシーグループを作成するときは、次の点に注意してください。

- を設定する必要があります qosPolicy キーを押します defaults バックエンド構成のブロック。次のバックエンド設定例を参照してください。

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "0.0.0.0",
  "dataLIF": "0.0.0.0",
  "svm": "svm0",
  "username": "user",
  "password": "pass",
  "defaults": {
    "qosPolicy": "standard-pg"
  },
  "storage": [
    {
      "labels": {"performance": "extreme"},
      "defaults": {
        "adaptiveQosPolicy": "extremely-adaptive-pg"
      }
    },
    {
      "labels": {"performance": "premium"},
      "defaults": {
        "qosPolicy": "premium-pg"
      }
    }
  ]
}
```

- ボリュームごとにポリシーグループを適用して、各ボリュームがポリシーグループの指定に従ってスループット全体を取得するようにします。共有ポリシーグループはサポートされません。

QoS ポリシーグループの詳細については、を参照してください ["ONTAP 9.8 QoS コマンド"](#)。

ストレージリソースへのアクセスを **Kubernetes** クラスタメンバーに制限する

Trident によって作成される NFS ボリュームと iSCSI LUN へのアクセスを制限することは、Kubernetes 環境のセキュリティ体制に欠かせない要素です。これにより、Kubernetes クラスタに属していないホストがボリュームにアクセスしたり、データが予期せず変更されたりすることを防止できます。

ネームスペースは Kubernetes のリソースの論理的な境界であることを理解することが重要です。ただし、同じネームスペース内のリソースは共有可能であることが前提です。重要なのは、ネームスペース間に機能がなことです。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。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 クラスターノードおよび Kubernetes クラスターの外部にあるターゲットサーバからのアクセスを可能にすることもできます。

専用のインフラノード（OpenShift など）や、ユーザアプリケーションにスケジュールできない他のノードを導入する場合は、別々のエクスポートポリシーを使用してストレージリソースへのアクセスをさらに制限する必要があります。これには、これらのインフラノードに導入されているサービス（OpenShift Metrics サービスや Logging サービスなど）のエクスポートポリシーの作成と、非インフラノードに導入されている標準アプリケーションの作成が含まれます。

専用のエクスポートポリシーを使用します

Kubernetes クラスター内のノードへのアクセスのみを許可するエクスポートポリシーが各バックエンドに存在することを確認する必要があります。Trident では、20.04 リリース以降、エクスポートポリシーを自動的に作成、管理できます。これにより、Trident はプロビジョニング対象のボリュームへのアクセスを Kubernetes クラスター内のノードに制限し、ノードの追加や削除を簡易化します。

また、エクスポートポリシーを手動で作成し、各ノードのアクセス要求を処理する 1 つ以上のエクスポートルールを設定することもできます。

- を使用します `vserver export-policy create` ONTAP の CLI コマンドを使用してエクスポートポリシーを作成します。
- を使用して、エクスポートポリシーにルールを追加します `vserver export-policy rule create` ONTAP CLI コマンド。

これらのコマンドを実行すると、データにアクセスできる Kubernetes ノードを制限できます。

無効にします `showmount` アプリケーション **SVM** 用

。 `showmount` 機能を使用すると、NFS クライアントが SVM を照会して、使用可能な NFS エクスポートのリストを表示できます。Kubernetes クラスターに導入されたポッドは、問題に対応しています `showmount -e` コマンドをデータ LIF に対して実行し、アクセス権のないマウントも含めて使用可能なマウントのリストを取得します。これだけではセキュリティ上の妥協ではありませんが、権限のないユーザが NFS エクスポートに接続するのを阻止する可能性のある不要な情報が提供されます。

を無効にする必要があります `showmount` SVM レベルの ONTAP CLI コマンドを使用して、次の作業を行います。

```
vserver nfs modify -vserver <svm_name> -showmount disabled
```

SolidFire のベストプラクティス

Trident に SolidFire ストレージを設定するためのベストプラクティスをご確認ください。

SolidFire アカウントを作成します

各 SolidFire アカウントは固有のボリューム所有者で、Challenge Handshake Authentication Protocol（CHAP；チャレンジハンドシェイク認証プロトコル）クレデンシャルのセットを受け取ります。アカウントに割り当てられたボリュームには、アカウント名とその CHAP クレデンシャルを使用してアクセスするか、ボリュームアクセスグループを通じてアクセスできます。アカウントには最大 2、000 個のボリュームを関連付けることができますが、1 つのボリュームが属することのできるアカウントは 1 つだけです。

QoS ポリシーを作成する

標準的なサービス品質設定を作成して保存し、複数のボリュームに適用する場合は、SolidFire のサービス品質（QoS）ポリシーを使用します。

QoS パラメータはボリューム単位で設定できます。QoS を定義する 3 つの設定可能なパラメータである Min IOPS、Max IOPS、Burst IOPS を設定することで、各ボリュームのパフォーマンスが保証されます。

4KB のブロックサイズの最小 IOPS、最大 IOPS、バースト IOPS の値を次に示します。

IOPS パラメータ	定義（Definition）	最小価値	デフォルト値	最大値（4KB）
最小 IOPS	ボリュームに対して保証されたレベルのパフォーマンス。	50 です	50 です	15、000
最大 IOPS	パフォーマンスはこの制限を超えません。	50 です	15、000	20 万
バースト IOPS	短時間のバースト時に許容される最大 IOPS。	50 です	15、000	20 万



Max IOPS と Burst IOPS は最大 200、000 に設定できますが、実際のボリュームの最大パフォーマンスは、クラスタの使用量とノードごとのパフォーマンスによって制限されます。

ブロックサイズと帯域幅は、IOPS に直接影響します。ブロックサイズが大きくなると、システムはそのブロックサイズを処理するために必要なレベルまで帯域幅を増やします。帯域幅が増えると、システムが処理可能な IOPS は減少します。を参照してください ["SolidFire のサービス品質"](#) QoS およびパフォーマンスの詳細については、を参照してください。

SolidFire 認証

Element では、認証方法として CHAP とボリュームアクセスグループ（VAG）の 2 つがサポートされています。CHAP は CHAP プロトコルを使用して、バックエンドへのホストの認証を行います。ボリュームアクセスグループは、プロビジョニングするボリュームへのアクセスを制御します。CHAP はシンプルで拡張性に制限がないため、認証に使用することを推奨します。



Trident と強化された CSI プロビジョニングツールは、CHAP 認証の使用をサポートします。VAG は、従来の CSI 以外の動作モードでのみ使用する必要があります。

CHAP 認証（イニシエータが対象のボリュームユーザであることの確認）は、アカウントベースのアクセス制御でのみサポートされます。認証に CHAP を使用している場合は、単方向 CHAP と双方向 CHAP の 2 つのオプションがあります。単方向 CHAP は、SolidFire アカウント名とイニシエータシークレットを使用してボリュームアクセスを認証します。双方向の CHAP オプションを使用すると、ボリュームがアカウント名とイニシエータシークレットを使用してホストを認証し、ホストがアカウント名とターゲットシークレットを使用してボリュームを認証するため、ボリュームを最も安全に認証できます。

ただし、CHAP を有効にできず VAG が必要な場合は、アクセスグループを作成し、ホストのイニシエータとボリュームをアクセスグループに追加します。アクセスグループに追加した各 IQN は、CHAP 認証の有無に関係なく、グループ内の各ボリュームにアクセスできます。iSCSI イニシエータが CHAP 認証を使用するように設定されている場合は、アカウントベースのアクセス制御が使用されます。iSCSI イニシエータが CHAP 認証を使用するように設定されていない場合は、ボリュームアクセスグループのアクセス制御が使用されます。

詳細情報の入手方法

ベストプラクティスのドキュメントの一部を以下に示します。を検索します ["NetApp ライブラリ"](#) 最新バージョンの場合。

- ONTAP *
- ["NFS Best Practice and Implementation Guide"](#)
- ["SAN アドミニストレーションガイド"](#)（iSCSI の場合）
- ["RHEL 向けの iSCSI のクイック構成"](#)
- Element ソフトウェア *
- ["SolidFire for Linux を設定しています"](#)
- NetApp HCI *
- ["NetApp HCI 導入の前提条件"](#)
- ["NetApp Deployment Engine にアクセスします"](#)
- アプリケーションのベストプラクティス情報 *
- ["ONTAP での MySQL に関するベストプラクティスです"](#)
- ["SolidFire での MySQL に関するベストプラクティスです"](#)
- ["NetApp SolidFire および Cassandra"](#)
- ["SolidFire での Oracle のベストプラクティス"](#)
- ["SolidFire での PostgreSQL のベストプラクティスです"](#)

すべてのアプリケーションに具体的なガイドラインがあるわけではありません。そのためには、ネットアップのチームと協力し、を使用することが重要です ["NetApp ライブラリ"](#) 最新のドキュメントを検索できます。

Astra Trident を統合

Astra Trident を統合するには、設計とアーキテクチャに関する次の要素を統合する必要があります。ドライバの選択と導入、ストレージクラス的设计、仮想ストレージプールの設計、永続的ボリューム要求（PVC）は、Astra Trident を使用したストレージプロビジョニング、ボリューム運用、OpenShift サービスの導入に影響を及ぼします。

ドライバの選択と展開

ストレージシステム用のバックエンドドライバを選択して導入します。

ONTAP バックエンドドライバ

ONTAP バックエンドドライバは、使用されるプロトコルと、ストレージシステムでのボリュームのプロビジョニング方法によって異なります。そのため、どのドライバを展開するかを決定する際には、慎重に検討する必要があります。

アプリケーションに共有ストレージを必要とするコンポーネント（同じ PVC にアクセスする複数のポッド）がある場合、NAS ベースのドライバがデフォルトで選択されますが、ブロックベースの iSCSI ドライバは非共有ストレージのニーズを満たします。アプリケーションの要件と、ストレージチームとインフラチームの快適さレベルに基づいてプロトコルを選択してください。一般的に、ほとんどのアプリケーションでは両者の違いはほとんどないため、共有ストレージ（複数のポッドで同時にアクセスする必要がある場合）が必要かどうかに基づいて判断することがよくあります。

使用可能なONTAP バックエンドドライバは次のとおりです。

- `ontap-nas`：プロビジョニングされた各PVは、ONTAP のフルFlexVolです。
- `ontap-nas-economy`：PVがプロビジョニングされた各ボリュームはqtreeであり、FlexVolあたりのqtree数は設定可能です（デフォルトは200）。
- `ontap-nas-flexgroup`：すべてのONTAP FlexGroup としてプロビジョニングされたPVごとに、SVM に割り当てられたすべてのアグリゲートが使用されます。
- `ontap-san`：プロビジョニングされた各PVは、固有のFlexVol内のLUNです。
- `ontap-san-economy`：プロビジョニングされた各PVはLUNで、FlexVolあたりのLUN数は設定可能です（デフォルトは100）。

3 つの NAS ドライバの間で選択すると、アプリケーションで使用できる機能にいくつかの影響があります。

次の表では、Astra Trident からすべての機能が提供されるわけではありません。一部の機能は、プロビジョニング後にストレージ管理者が適用する必要があります。上付き文字の脚注は、機能やドライバごとに機能を区別します。

ONTAP NAS ドライバ	Snapshot	クローン	動的なエクスポートポリシー	マルチアタッチ	QoS	サイズ変更	レプリケーション
<code>ontap-nas</code>	はい。	はい。	○脚注：5[]	はい。	Yesfootnote: 1[]	はい。	Yesfootnote: 1[]
<code>ontap-nas-economy</code>	Yesfootnote: 3[]	Yesfootnote: 3[]	○脚注：5[]	はい。	Yesfootnote: 3[]	はい。	Yesfootnote: 3[]
<code>ontap-nas-flexgroup</code>	Yesfootnote: 1[]	いいえ	○脚注：5[]	はい。	Yesfootnote: 1[]	はい。	Yesfootnote: 1[]

Astra Trident は、ONTAP 向けに 2 つの SAN ドライバを提供しています。このドライバの機能は次のとおりです。

ONTAP SANドライバ	Snapshot	クローン	マルチアタッチ	双方向CHAP	QoS	サイズ変更	レプリケーション
ontap-san	はい。	はい。	Yesfootnote: 4[]	はい。	Yesfootnote: 1[]	はい。	Yesfootnote: 1[]
ontap-san-economy	はい。	はい。	Yesfootnote: 4[]	はい。	Yesfootnote: 3[]	はい。	Yesfootnote: 3[]

上記の表の脚注：

Yes [1]：Astra Tridentで管理されない

Yesfootnote: 2[]：Astra Tridentが管理しますが、PV Granularは管理しません

Yesfootnote: 3[]：Astra Tridentで管理されず、PV Granularでは管理されない

Yes [4]:raw-blockボリュームでサポート

Yesfootnote: 5[]：CSI Tridentによるサポート

PV に細分化されていない機能は FlexVol 全体に適用され、PVS（共有 FlexVol 内の qtree または LUN）にはすべて共通のスケジュールが適用されます。

上の表に示すように、の機能の多くはです ontap-nas および ontap-nas-economy は同じです。しかし、だからです ontap-nas-economy ドライバは、PV単位でスケジュールを制御する機能を制限します。これは、ディザスタリカバリやバックアップ計画に特に影響を与える可能性があります。ONTAP ストレージでPVCクローン機能を利用したい開発チームの場合、この方法はを使用する場合にのみ使用できます ontap-nas、ontap-san または ontap-san-economy ドライバ。



。solidfire-san また、ドライバはPVCをクローニングすることもできます。

Cloud Volumes ONTAP バックエンドドライバ

Cloud Volumes ONTAP は、ファイル共有や NAS および SAN プロトコル（NFS、SMB / CIFS、iSCSI）を提供するブロックレベルストレージなど、さまざまなユースケースでデータ制御とエンタープライズクラスのストレージ機能を提供します。Cloud Volume ONTAP の互換性のあるドライバはです ontap-nas、ontap-nas-economy、ontap-san および ontap-san-economy。Cloud Volume ONTAP for Azure と Cloud Volume ONTAP for GCP に該当します。

ONTAP バックエンドドライバ用のAmazon FSX

Amazon FSX for ONTAP を使用すると、お客様は使い慣れたネットアップの機能、パフォーマンス、管理機能を活用しながら、AWS にデータを格納する際のシンプルさ、即応性、セキュリティ、拡張性を活用できます。FSX for ONTAP は、ONTAP のファイルシステム機能と管理 API の多くをサポートしています。Cloud Volume ONTAP の互換性のあるドライバはです ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、ontap-san および ontap-san-economy。

NetApp HCI / SolidFire バックエンドドライバ

。solidfire-san NetApp HCI / SolidFire プラットフォームで使用されるドライバ。管理者は、QoS制限に基づいてTrident用にElementバックエンドを設定できます。Tridentでプロビジョニングされるボリュームに特定のQoS制限を設定するためにバックエンドを設計する場合は、を使用してください type バックエンドファイル内のパラメータ。また、管理者は、を使用してストレージに作成できるボリュームサイズを制限することもできます limitVolumeSize パラメータ現在のところ、ボリュームのサイズ変更やボリュームのレプリケーションなどのElementストレージ機能は、ではサポートされていません solidfire-san ドライバ。これらの処理は、Element ソフトウェアの Web UI から手動で実行する必要があります。

SolidFire ドライバ	Snapshot	クローン	マルチアタッチ	CHAP	QoS	サイズ変更	レプリケーション
solidfire-san	はい。	はい。	○脚注： 2 []	はい。	はい。	はい。	Yesfootnote: 1[]

脚注：

Yes [1]：Astra Tridentで管理されない

Yes [2]:raw-blockボリュームでサポート

Azure NetApp Files バックエンドドライバ

Astra Tridentがを使用 azure-netapp-files を管理するドライバ "Azure NetApp Files の特長" サービス

このドライバの詳細と設定方法については、を参照してください "Azure NetApp Files 向けの Trident バックエンド構成"。

Azure NetApp Files ドライバ	Snapshot	クローン	マルチアタッチ	QoS	を展開します	レプリケーション
azure-netapp-files	はい。	はい。	はい。	はい。	はい。	Yesfootnote: 1[]

脚注：

Yes [1]：Astra Tridentで管理されない

Cloud Volumes Service とGCPバックエンドドライバ

Astra Tridentがを使用 gcp-cvs GCPバックエンドのCloud Volumes Service とリンクするドライバ。TridentでGCPバックエンドを設定するには、を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 GCP Web ポータルで確認できますが、GCPでCloud Volume のAPI アクセスを設定する際に作成したサービスアカウントの秘密鍵ファイルからAPI キーを取得する必要があります。Astra Trident なら、CVS ボリュームを 2 つのうちの 1 つで作成できます "サービスタイプ"：

1. * CVS *：基本 CVS サービスのタイプ。パフォーマンスレベルが限定的か中程度かに関係なく、高ゾーンの可用性を実現します。

2. * CVS - パフォーマンス * : パフォーマンスを重視する本番環境のワークロードに最適な、パフォーマンスに最適化されたサービスタイプ。3つの独自のサービスレベルから選択できます [standard、premium`および`extreme]。

CVSとCVSパフォーマンスのボリュームの最小サイズは100GiBです。

CVS for GCPドライバ	Snapshot	クローン	マルチアタッチ	QoS	を展開します	レプリケーション
gcp-cvs	はい。	はい。	はい。	はい。	はい。	Yes ^{footnote: 1}

脚注：

Yes ^[1] : Astra Tridentで管理されない

。 gcp-cvs ドライバは仮想ストレージプールを使用します。仮想ストレージプールはバックエンドを抽象化し、Astra Trident がボリュームの配置を決定できるようにします。管理者が backend.json ファイルに仮想ストレージプールを定義します。ストレージクラスは、ラベルを使用する仮想ストレージプールを識別します。

ストレージクラスの設計

Kubernetes ストレージクラスオブジェクトを作成するには、個々のストレージクラスを設定して適用する必要があります。このセクションでは、アプリケーション用のストレージクラスの設計方法について説明します。

特定のバックエンド使用率

フィルタリングは、特定のストレージクラスオブジェクト内で使用でき、そのストレージクラスで使用するストレージプールまたはプールのセットを決定します。ストレージクラスでは、次の3セットのフィルタを設定できます。 storagePools、 additionalStoragePools`または`excludeStoragePools。

。 storagePools パラメータを指定すると、指定した属性に一致するプールのセットだけにストレージが制限されます。。 additionalStoragePools パラメータは、属性とで選択されたプールのセットに加えて、Astra Tridentがプロビジョニングに使用する一連のプールを拡張するために使用されます storagePools パラメータどちらか一方のパラメータを単独で使用することも、両方を使用して、適切なストレージプールセットが選択されていることを確認することもできます。

。 excludeStoragePools パラメータを使用すると、属性に一致する一連のプールが具体的に除外されます。

QoSポリシーをエミュレートします

ストレージクラスを設計してQoSポリシーをエミュレートする場合は、でストレージクラスを作成します media 属性の形式 hdd または ssd。に基づきます media ストレージクラスで説明されている属性の中から、Tridentが提供する適切なバックエンドを選択します hdd または ssd media属性に一致するアグリゲートを作成し、ボリュームのプロビジョニングを特定のアグリゲートに転送します。そこで、Premiumストレージクラスを作成します media 属性をとして設定します ssd Premium QoSポリシーに分類できます。メディア属性を「hdd」に設定し、標準のQoSポリシーとして分類できる、別のストレージクラス標準を作成できます。また、ストレージクラスの「IOPS」属性を使用して、QoSポリシーとして定義できる Element アプライアンスにプロビジョニングをリダイレクトすることもできます。

特定の機能に基づいてバックエンドを利用する

ストレージクラスは、シンプロビジョニングとシックプロビジョニング、Snapshot、クローン、暗号化などの機能が有効になっている特定のバックエンドでボリュームを直接プロビジョニングするように設計できます。使用するストレージを指定するには、必要な機能を有効にしてバックエンドに適したストレージクラスを作成します。

仮想ストレージプール

Virtual Storage Pool は、すべての Astra Trident バックエンドで利用可能Trident が提供する任意のドライバを使用して、任意のバックエンドに対して仮想ストレージプールを定義できます。

仮想ストレージプールを使用すると、管理者はストレージクラスで参照可能なバックエンド経由で抽象化レベルを作成して、バックエンドにボリュームを柔軟かつ効率的に配置できます。同じサービスクラスを使用して異なるバックエンドを定義できます。さらに、同じバックエンドに異なる特性を持つ複数のストレージプールを作成することもできます。セレクトラ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トラ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トラのラベルが複数のストレージプールに一致する場合、Astra Trident がボリュームのプロビジョニングに使用するストレージクラスを 1 つ選択します。

仮想ストレージプールの設計

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。この問題は、仮想ストレージプールの導入に伴って、軽減されています。仮想ストレージプールは、バックエンドと Kubernetes ストレージクラスの間に抽象化されたレベルです。管理者は、Kubernetes ストレージクラスを介してパラメータとラベルを定義でき、セレクトラとしてバックエンドに依存しない方法で参照できます。Virtual Storage Pools は、サポート対象のすべてのネットアップバックエンドに Astra Trident を使用して定義できます。リストには、SolidFire / NetApp HCI、ONTAP、GCP 上の Cloud Volumes Service、Azure NetApp Files が含まれます。



仮想ストレージプールを定義する場合は、バックエンド定義内の既存の仮想プールの順序を変更しないことを推奨します。また、既存の仮想プールの属性を編集または変更したり、新しい仮想プールを定義したりしないことを推奨します。

さまざまなサービスレベル/QoS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ストレージプールを設計することができます。Cloud Volume Service for Azure NetApp Files の仮想プール実装を使用して、さまざまなサービスクラスをセットアップする方法を見ていきましょう。さまざまなパフォーマンスレベルを表す複数のラベルで ANF バックエンドを設定します。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。では、別の仮想ストレージプールにマッピングする別の Kubernetes ストレージクラスを作成します。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。

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ストレージプールは、単一のストレージバックエンドから設計できます。そのため、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 異なる仮想ストレージ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、選択した仮想ストレージプールに定義された設定が適用されます。

ストレージプロビジョニングに影響する PVC 特性

要求されたストレージクラスを超えたパラメータの一部は、PVC の作成時に Astra Trident のプロビジョニング決定プロセスに影響を与える可能性があります。

アクセスモード

PVC 経由でストレージを要求する場合、必須フィールドの 1 つがアクセスモードです。必要なモードは、ストレージ要求をホストするために選択されたバックエンドに影響を与える可能性があります。

Astra Trident は、次のマトリックスで指定されたアクセス方法で使用されているストレージプロトコルと一致するかどうかを試みます。これは、基盤となるストレージプラットフォームに依存しません。

	ReadWriteOnce コマンドを使用します	ReadOnlyMany	ReadWriteMany
iSCSI	はい。	はい。	○ (Raw ブロック)
NFS	はい。	はい。	はい。

NFS バックエンドが設定されていない Trident 環境に送信された ReadWriteMany PVC が要求された場合、ボリュームはプロビジョニングされません。このため、リクエストは、アプリケーションに適したアクセスモードを使用する必要があります。

ボリューム操作

永続ボリュームの変更

永続ボリュームとは、Kubernetes で変更不可のオブジェクトを 2 つだけ除いてです。再利用ポリシーとサイズは、いったん作成されると変更できます。ただし、これにより、ボリュームの一部の側面が Kubernetes 以外で変更されることが防止されるわけではありません。特定のアプリケーション用にボリュームをカスタマイズしたり、誤って容量が消費されないようにしたり、何らかの理由でボリュームを別のストレージコントローラに移動したりする場合に便利です。



Kubernetes のツリー内プロビジョニングツールは、現時点では NFS または iSCSI PVS のボリュームサイズ変更処理をサポートしていません。Astra Trident では、NFS ボリュームと iSCSI ボリュームの両方の拡張がサポートされています。

作成後に PV の接続の詳細を変更することはできません。

オンデマンドのボリューム **Snapshot** を作成

Astra Trident は、CSI フレームワークを使用して、オンデマンドでボリュームスナップショットを作成し、スナップショットから PVC を作成できます。Snapshot は、データのポイントインタイムコピーを管理し、Kubernetes のソース PV とは無関係にライフサイクルを管理する便利な方法です。これらの Snapshot を使用して、PVC をクローニングできます。

Snapshot からボリュームを作成します

Astra Trident は、ボリューム Snapshot からの PersistentVolumes の作成もサポートしています。これを実現するには、PersistentVolumeClaimを作成し、を指定します datasource ボリュームの作成元となる必要がある Snapshot。Astra Trident がこの PVC を処理するには、Snapshot にデータが存在するボリュームを作成します。この機能を使用すると、複数のリージョン間でデータを複製したり、テスト環境を作成したり、破損し

た本番ボリューム全体を交換したり、特定のファイルとディレクトリを取得して別の接続ボリュームに転送したりできます。

クラスタ内でボリュームを移動します

ストレージ管理者は、ONTAP クラスタ内のアグリゲート間およびコントローラ間で、ストレージ利用者への無停止でボリュームを移動できます。この処理は、デスティネーションアグリゲートが Trident が使用している SVM からアクセス可能なアグリゲートであるかぎり、Astra Trident または Kubernetes クラスタには影響しません。この点が重要なのは、アグリゲートが SVM に新たに追加された場合、Astra Trident に再追加してバックエンドを更新する必要があることです。これにより、Astra Trident が SVM のインベントリを再作成し、新しいアグリゲートが認識されるようになります。

ただし、バックエンド間でのボリュームの移動は Astra Trident では自動ではサポートされていません。これには、同じクラスタ内の SVM 間、クラスタ間、または別のストレージプラットフォーム上の SVM 間が含まれます（たとえストレージシステムが Trident から Astra に接続されている場合でも）。

ボリュームが別の場所にコピーされた場合、ボリュームインポート機能を使用して現在のボリュームを Astra Trident にインポートできます。

ボリュームを展開します

Astra Trident は、NFS と iSCSI PVS のサイズ変更をサポートしています。これにより、ユーザは Kubernetes レイヤを介してボリュームのサイズを直接変更できます。ボリュームを拡張できるのは、ONTAP、SolidFire / NetApp HCI、Cloud Volumes Service バックエンドなど、主要なすべてのネットアップストレージプラットフォームです。あとで拡張できるようにするには、をに設定します `allowVolumeExpansion` 終了: `true` ボリュームに関連付けられているストレージクラス内のストレージクラス。永続ボリュームのサイズを変更する必要がある場合は、を編集します `spec.resources.requests.storage Persistent Volume Claim` のアノテーションを、必要なボリュームサイズに設定します。Tridentによって、ストレージクラスタ上のボリュームのサイズが自動的に変更されます。

既存のボリュームを **Kubernetes** にインポートする

Volume Import では、既存のストレージボリュームを Kubernetes 環境にインポートできます。これは現在、でサポートされています `ontap-nas`、`ontap-nas-flexgroup`、`solidfire-san`、`azure-netapp-files` および `gcp-cvs` ドライバ。この機能は、既存のアプリケーションを Kubernetes に移植する場合や、ディザスタリカバリシナリオで使用する場合に便利です。

ONTAP およびを使用する場合 `solidfire-san` ドライバの場合は、コマンドを使用します `tridentctl import volume <backend-name> <volume-name> -f /path/pvc.yaml` 既存のボリュームを Kubernetes にインポートして Astra Trident で管理 `import volume` コマンドで使った PVC YAML または JSON ファイルは、Astra Trident をプロビジョニングツールとして識別するストレージクラスを指定します。NetApp HCI / SolidFire バックエンドを使用する場合は、ボリューム名が一意であることを確認してください。ボリューム名が重複している場合は、ボリュームインポート機能で区別できるように、ボリュームを一意の名前にクローニングします。

状況に応じて `azure-netapp-files` または `gcp-cvs` ドライバを使用する場合は、コマンドを使用します `tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml` から Kubernetes にボリュームをインポートして Astra Trident で管理。これにより、ボリューム参照が一意になります。

上記のコマンドを実行すると、Astra Trident がバックエンド上にボリュームを検出し、サイズを確認します。設定された PVC のボリュームサイズが自動的に追加（必要に応じて上書き）されます。次に Astra Trident が新しい PV を作成し、Kubernetes が PVC を PV にバインド

特定のインポートされた PVC を必要とするようにコンテナを導入した場合、ボリュームインポートプロセスによって PVC/PV ペアがバインドされるまで、コンテナは保留状態のままになります。PVC/PV ペアがバインドされると、他に問題がなければコンテナが起動します。

OpenShift サービスを導入します

OpenShift の付加価値クラスタサービスは、クラスタ管理者とホストされているアプリケーションに重要な機能を提供します。これらのサービスが使用するストレージはノードローカルリソースを使用してプロビジョニングできますが、これにより、サービスの容量、パフォーマンス、リカバリ性、持続可能性が制限されることがよくあります。エンタープライズストレージアレイを活用してこれらのサービスに容量を提供することで、劇的に向上したサービスを実現できます。ただし、すべてのアプリケーションと同様に、OpenShift とストレージ管理者は、緊密に連携してそれぞれに最適なオプションを決定する必要があります。Red Hat のドキュメントは、要件を決定し、サイジングとパフォーマンスのニーズを確実に満たすために大きく活用する必要があります。

レジストリサービス

レジストリのストレージの導入と管理については、に記載されています ["netapp.io のコマンドです"](#) を参照してください ["ブログ"](#)。

ロギングサービス

他の OpenShift サービスと同様に、ログ記録サービスは、Ansible と、インベントリファイル（別名）で提供される構成パラメータを使用して導入されますホスト。プレイブックに含まれています。インストール方法には、OpenShiftの初期インストール時にログを導入する方法と、OpenShiftが終了した後にログを導入する方法の2つがあります。
インストール済み。



Red Hat OpenShift バージョン 3.9 以降、データ破損に関する懸念があるため、記録サービスに NFS を使用しないことを公式のドキュメントで推奨しています。これは、Red Hat 製品のテストに基づいています。ONTAP の NFS サーバにはこのような問題はなく、簡単にロギング環境をバックアップできます。ロギングサービスには最終的にどちらかのプロトコルを選択する必要がありますが、両方のプロトコルがネットアッププラットフォームを使用する場合に適していることと、NFS を使用する理由がないことを確認してください。

ロギングサービスでNFSを使用する場合は、Ansible変数を設定する必要があります

`openshift_enable_unsupported_configurations` 終了: `true` インストーラが失敗しないようにします。

はじめに

ロギングサービスは、必要に応じて、両方のアプリケーションに導入することも、OpenShift クラスタ自体のコア動作に導入することもできます。操作ログを配置する場合は、変数を指定します

`openshift_logging_use_ops` として `'true'` サービスのインスタンスが2つ作成されます。操作のロギングインスタンスを制御する変数には「ops」が含まれ、アプリケーションのインスタンスには含まれません。

導入方法に基づいて Ansible 変数を設定することは、基盤のサービスが正しいストレージを利用できるようにするために重要です。各導入方法のオプションを見てみましょう。



以下の表には、ロギングサービスに関連するストレージ構成に関連する変数のみが含まれています。その他のオプションは、で確認できます ["Red Hat OpenShift のロギングに関するドキュメント"](#) 導入環境に応じて、確認、設定、使用する必要があります。

次の表の変数では、入力した詳細を使用してロギングサービスの PV と PVC を作成する Ansible プレイブックが作成されます。この方法は、OpenShift インストール後にコンポーネントインストールプレイブックを使用するよりもはるかに柔軟性に劣るが、既存のボリュームがある場合はオプションとなります。

変数（ Variable ）	詳細
openshift_logging_storage_kind	をに設定します nfs ログ記録サービス用のNFS PVを作成するため。
openshift_logging_storage_host	NFS ホストのホスト名または IP アドレス。仮想マシンのデータ LIF に設定してください。
openshift_logging_storage_nfs_directory	NFS エクスポートのマウントパス。たとえば、ボリュームがとしてジャンクションされている場合などで <code>/openshift_logging</code> この変数には、このパスを使用します。
openshift_logging_storage_volume_name	名前。例 <code>pv_ose_logs</code> 作成するPVの。
openshift_logging_storage_volume_size	たとえば、NFSエクスポートのサイズ 100Gi。

OpenShift クラスタがすでに実行中で、そのため Trident を導入して設定した場合、インストーラは動的プロビジョニングを使用してボリュームを作成できます。次の変数を設定する必要があります。

変数（ Variable ）	詳細
openshift_logging_es_pvc_dynamic	動的にプロビジョニングされたボリュームを使用する場合は true に設定します。
openshift_logging_es_pvc_storage_class_name	PVC で使用されるストレージクラスの名前。
openshift_logging_es_pvc_size	PVC で要求されたボリュームのサイズ。
openshift_logging_es_pvc_prefix	ロギングサービスで使用する PVC のプレフィックス。
openshift_logging_es_ops_pvc_dynamic	をに設定します true 動的にプロビジョニングされたボリュームをopsロギングインスタンスに使用する。
openshift_logging_es_ops_pvc_storage_class_name	処理ロギングインスタンスのストレージクラスの名前。
openshift_logging_es_ops_pvc_size	処理インスタンスのボリューム要求のサイズ。
openshift_logging_es_ops_pvc_prefix	ops インスタンス PVC のプレフィックス。

ロギングスタックを導入します

初期の OpenShift インストールプロセスの一部としてロギングを導入する場合、標準の導入プロセスに従うだけで済みます。Ansible は、必要なサービスと OpenShift オブジェクトを構成および導入して、Ansible が完了したらすぐにサービスを利用できるようにします。

ただし、最初のインストール後に導入する場合は、コンポーネントプレイブックを Ansible で使用する必要があります。このプロセスは、OpenShift のバージョンが異なるためわずかに変更される場合があるので、必ず読んで従うようにしてください ["Red Hat OpenShift Container Platform 3.11 のドキュメント"](#) 使用しているバージョンに対応した

指標サービス

この指標サービスは、OpenShift クラスタのステータス、リソース利用率、可用性に関する重要な情報を管理者に提供します。ポッドの自動拡張機能にも必要であり、多くの組織では、チャージバックやショーバックのアプリケーションに指標サービスのデータを使用しています。

ロギングサービスや OpenShift 全体と同様に、Ansible を使用して指標サービスを導入します。また、ロギングサービスと同様に、メトリックサービスは、クラスタの初期セットアップ時またはコンポーネントのインストール方法を使用して運用可能になった後に導入できます。次の表に、指標サービスに永続的ストレージを設定する際に重要となる変数を示します。



以下の表には、指標サービスに関連するストレージ構成に関連する変数のみが含まれています。このドキュメントには、他にも導入環境に応じて確認、設定、使用できるオプションが多数あります。

変数 (Variable)	詳細
<code>openshift_metrics_storage_kind</code>	をに設定します <code>nfs</code> ログ記録サービス用の NFS PV を作成するため。
<code>openshift_metrics_storage_host</code>	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。
<code>openshift_metrics_storage_nfs_directory</code>	NFS エクスポートのマウントパス。たとえば、ボリュームがとしてジャンクションされている場合などで <code>/openshift_metrics</code> この変数には、このパスを使用します。
<code>openshift_metrics_storage_volume_name</code>	名前、 例： <code>'pv_ose_metrics'</code> 作成する PV の。
<code>openshift_metrics_storage_volume_size</code>	たとえば、NFS エクスポートのサイズ 100Gi。

OpenShift クラスタがすでに実行中で、そのため Trident を導入して設定した場合、インストーラは動的プロビジョニングを使用してボリュームを作成できます。次の変数を設定する必要があります。

変数 (Variable)	詳細
<code>openshift_metrics_cassandra_pvc_prefix</code>	メトリック PVC に使用するプレフィックス。
<code>openshift_metrics_cassandra_pvc_size</code>	要求するボリュームのサイズ。
<code>openshift_metrics_cassandra_storage_type</code>	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 <code>dynamic</code> に設定する必要があります。
<code>openshift_metrics_cassandra_pvc_storage_class_name</code>	使用するストレージクラスの名前。

指標サービスを導入する

ホスト / インベントリファイルに適切な Ansible 変数を定義して、Ansible でサービスを導入します。OpenShift インストール時に導入する場合は、PV が自動的に作成されて使用されます。コンポーネントプレイブックを使用して導入する場合、OpenShift のインストール後に Ansible によって必要な PVC が作成

されます。また、Trident 用のストレージをプロビジョニングしたあとにサービスを導入します。

上記の変数と導入プロセスは、OpenShift の各バージョンで変更される可能性があります。必ず見直しを行ってください ["RedHat OpenShift 導入ガイド"](#) をバージョンに合わせて設定し、環境に合わせて設定します。

データ保護

ネットアップのストレージプラットフォームが提供するデータ保護とリカバリのオプションについて説明します。Astra Trident では、こうした機能の一部を活用できるボリュームをプロビジョニングできます。永続性に関する要件があるアプリケーションごとに、データ保護とリカバリの戦略を用意しておく必要があります。

をバックアップします etcd クラスタデータ

Astra Tridentは、Kubernetesクラスタのメタデータを格納します etcd データベース：を定期的にバックアップしてください etcd クラスタデータは、災害発生時にKubernetesクラスタをリカバリする際に重要です。

手順

1. `etcdctl snapshot save` コマンドを使用すると、のポイントインタイムスナップショットを作成できます etcd クラスタ：

```
sudo docker run --rm -v /backup:/backup \
  --network host \
  -v /etc/kubernetes/pki/etcd:/etc/kubernetes/pki/etcd \
  --env ETCDCTL_API=3 \
  registry.k8s.io/etcd-amd64:3.2.18 \
  etcdctl --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt \
  --key=/etc/kubernetes/pki/etcd/healthcheck-client.key \
  snapshot save /backup/etcd-snapshot.db
```

このコマンドは、etcdコンテナをスピンアップしてetcd Snapshotを作成し、に保存します /backup ディレクトリ。

2. 災害が発生した場合は、etcd Snapshot を使用して Kubernetes クラスタをスピンアップできます。を使用します `etcdctl snapshot restore` に作成された特定のSnapshotをリストアするコマンド /var/lib/etcd フォルダ。リストア後、を確認します /var/lib/etcd フォルダに追加されました member フォルダ。次に、の例を示します `etcdctl snapshot restore` コマンドを実行します

```
etcdctl snapshot restore '/backup/etcd-snapshot-latest.db' ; mv
/default.etcd/member/ /var/lib/etcd/
```

3. Kubernetes クラスタを初期化する前に、必要な証明書をすべてコピーしておきます。
4. を使用してクラスタを作成します `--ignore-preflight-errors=DirAvailable-var-lib-etcd` フ

ラグ。

5. クラスタが起動したら、 kube-system ポッドが起動していることを確認します。
6. を使用します `kubectl get crd Trident`で作成されたカスタムリソースが存在するかどうかを確認し、 Tridentオブジェクトを取得してすべてのデータが利用可能であることを確認するコマンド。

ONTAP スナップショットを使用して日付をリカバリします

Snapshot は、アプリケーションデータのポイントインタイムリカバリオプションを提供することで重要な役割を果たします。ただし、スナップショットは単独ではバックアップされず、ストレージシステムの障害やその他の災害に対する保護は行われません。しかし、ほとんどのシナリオで、データをすばやく簡単にリカバリできる便利な方法です。ONTAP Snapshot テクノロジーを使用してボリュームのバックアップを作成する方法とリストアする方法について説明します。

- Snapshotポリシーがバックエンドで定義されていない場合、デフォルトでが使用されます none ポリシー：そのため、 ONTAP では自動 Snapshot は作成されません。ただし、ストレージ管理者は、 ONTAP 管理インターフェイスから手動で Snapshot を作成したり、 Snapshot ポリシーを変更したりできます。これは Trident の動作には影響しません。
- デフォルトでは、 snapshot ディレクトリは表示されません。これにより、を使用してプロビジョニングしたボリュームの互換性を最大限に高めることができます `ontap-nas` および `ontap-nas-economy` ドライバ。を有効にします `.snapshot` を使用するときのディレクトリ `ontap-nas` および `ontap-nas-economy` アプリケーションがスナップショットからデータを直接リカバリできるようにするドライバ。
- を使用して、以前のSnapshotに記録されている状態にボリュームをリストアします `volume snapshot restore` ONTAP CLIコマンド。Snapshot コピーをリストアすると、既存のボリューム構成は上書きされます。Snapshot コピーの作成後にボリューム内のデータに加えた変更はすべて失われます。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot vol3_snap_archive
```

ONTAP を使用してデータをレプリケート

データのレプリケートは、ストレージレイの障害によるデータ損失から保護する上で重要な役割を果たします。



ONTAP レプリケーションテクノロジーの詳細については、を参照してください ["ONTAP のドキュメント"](#)。

SnapMirror Storage Virtual Machine (SVM) レプリケーション

を使用できます **"SnapMirror"** 設定とそのボリュームを含む SVM 全体をレプリケートすること。災害が発生した場合は、 SnapMirror デスティネーション SVM をアクティブ化してデータの提供を開始できます。システムがリストアされたら、プライマリに戻すことができます。

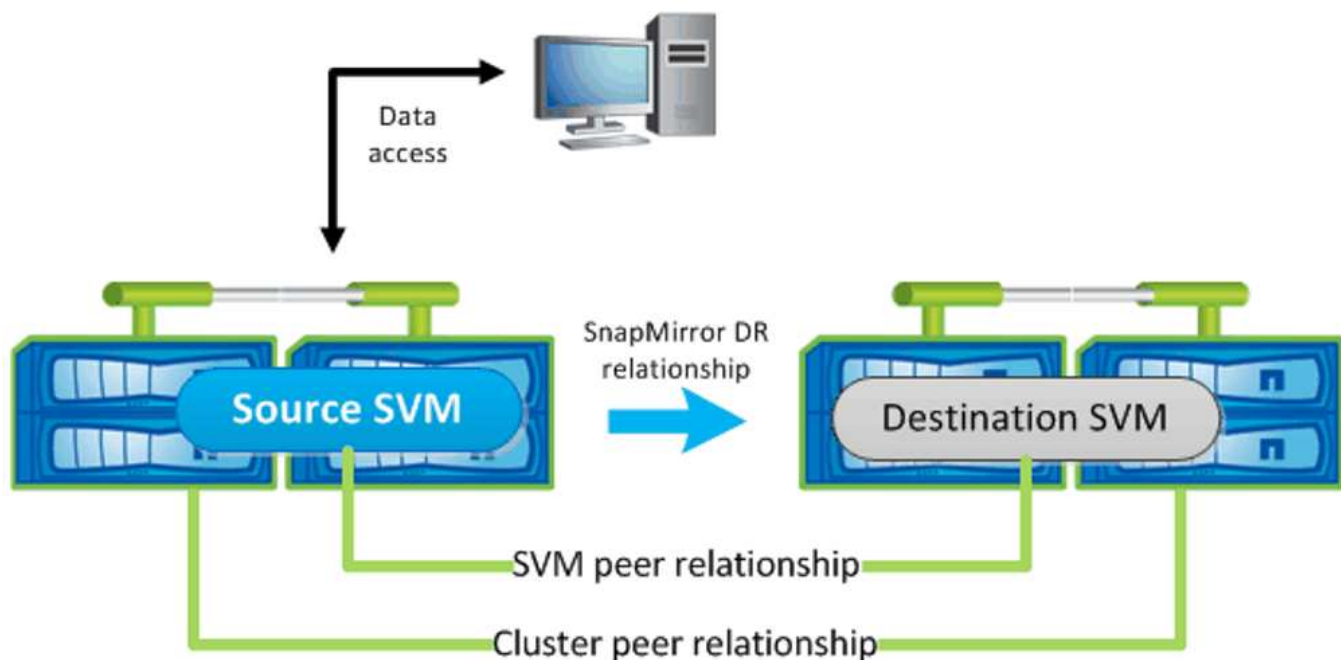
Astra Trident は、レプリケーション関係自体を構成できないため、ストレージ管理者は ONTAP の SnapMirror SVM レプリケーション機能を使用して、ボリュームをディザスタリカバリ (DR) デスティネーションに自動的にレプリケートできます。

SnapMirror SVM レプリケーション機能を使用する場合や、現在この機能を使用している場合は、次の点を考慮してください。

- SVM-DR が有効になっている SVM ごとに別個のバックエンドを作成する必要があります。
- レプリケートされたバックエンドを必要な場合を除き選択しないようにストレージクラスを設定する必要があります。SVM DR をサポートするバックエンドにレプリケーション関係の保護をプロビジョニングする必要がないボリュームがある場合、この問題を回避することが重要です。
- アプリケーション管理者は、データのレプリケーションに伴う追加のコストと複雑さを理解し、リカバリプランを決定してから、データレプリケーションを利用する必要があります。
- SnapMirror デスティネーション SVM をアクティブ化する前に、スケジュールされたすべての SnapMirror 転送を停止し、実行中のすべての SnapMirror 転送を中止してレプリケーション関係を解除し、ソース SVM を停止してから、SnapMirror デスティネーション SVM を起動します。
- Astra Trident では、SVM の障害は自動では検出されない。そのため、障害が発生した場合は、管理者が実行する必要があります `tridentctl backend update` 新しいバックエンドへのTridentのフェールオーバーをトリガーするコマンド。

SVM のセットアップ手順の概要を次に示します。

- ソースクラスタとデスティネーションクラスタ間にピア関係を設定します。
- を使用してデスティネーションSVMを作成します `-subtype dp-destination` オプション
- レプリケーションジョブスケジュールを作成して、必要な間隔でレプリケーションが実行されるようにします。
- を使用して、デスティネーションSVMからソースSVMへのSnapMirrorレプリケーションを作成します `-identity-preserve true` ソースSVM構成とソースSVMインターフェイスをデスティネーションに確実にコピーするオプション。デスティネーション SVM から、 SnapMirror SVM レプリケーション関係を初期化します。



Trident のディザスタリカバリワークフロー

Astra Trident 19.07 以降では、Kubernetes の SSD を使用して独自の状態を保存、管理しています。Kubernetes クラスタを使用します `etcd` をクリックしてメタデータを格納します。ここで

は、Kubernetesを使用することを前提としています。etcd データファイルと証明書はネットアップFlexVol に格納されています。この FlexVol は SVM にあり、SVM の SnapMirror SVM-DR 関係はセカンダリサイトの デスティネーション SVM と一緒にあります。

災害発生時に Astra Trident を使用して、単一のマスター Kubernetes クラスタをリカバリする手順を次に示します。

1. ソース SVM で障害が発生した場合は、SnapMirror デスティネーション SVM をアクティブ化します。そのためには、スケジュールされた SnapMirror 転送を停止し、実行中の SnapMirror 転送を中止して、レプリケーション関係を解除し、ソース SVM を停止して、デスティネーション SVM を起動します。
2. デスティネーションSVMから、Kubernetesが含まれているボリュームをマウントします。etcd マスターノードとしてセットアップされるホストのデータファイルと証明書。
3. Kubernetesクラスタに関連する必要な証明書をのにすべてコピーします。/etc/kubernetes/pki そしてetcd member のファイル /var/lib/etcd。
4. を使用してKubernetesクラスタを作成します。kubeadm init コマンドにを指定します --ignore -preflight-errors=DirAvailable-var-lib-etcd フラグ。Kubernetes ノードに使用するホスト名は、ソースの Kubernetes クラスタと同じであることが必要です。
5. を実行します kubectl get crd コマンドを使用して、すべてのTridentカスタムリソースが稼働しているかどうかを確認し、Tridentオブジェクトを取得して、すべてのデータが利用可能であることを確認します。
6. を実行して、必要なすべてのバックエンドを更新し、新しいデスティネーションSVM名を反映させます。
`./tridentctl update backend <backend-name> -f <backend-json-file> -n <namespace>` コマンドを実行します



アプリケーション永続ボリュームの場合、デスティネーション SVM がアクティブ化されると、Trident によってプロビジョニングされたすべてのボリュームがデータの提供を開始します。前述の手順に従って Kubernetes クラスタをデスティネーション側でセットアップしたら、すべての導入ポッドとポッドが開始され、コンテナ化されたアプリケーションは問題なく実行されます。

SnapMirror ボリュームのレプリケーション

ONTAP SnapMirrorボリュームレプリケーションはディザスタリカバリ機能で、ボリュームレベルのプライマリストレージからデスティネーションストレージへのフェイルオーバーを可能にします。SnapMirror は、Snapshot を同期することで、セカンダリストレージ上のプライマリストレージのボリュームレプリカまたはミラーを作成します。

ONTAP の SnapMirror ボリュームレプリケーションのセットアップ手順の概要を次に示します。

- ボリュームが配置されているクラスタとボリュームからデータを提供する SVM 間のピアリングを設定します。
- 関係の動作を制御する SnapMirror ポリシーを作成し、その関係の設定属性を指定します。
- を使用して、デスティネーションボリュームとソースボリューム間の SnapMirror 関係を作成します [snapmirror create コマンド]を押して、適切なSnapMirrorポリシーを割り当てます。
- SnapMirror 関係の作成後、ソースボリュームからデスティネーションボリュームへのベースライン転送が完了するように、関係を初期化します。



Trident の SnapMirror ボリュームディザスタリカバリワークフロー

Astra Trident で単一のマスター Kubernetes クラスタをリカバリする手順を次に示します。

1. 災害が発生した場合は、スケジュールされたすべての SnapMirror 転送を停止し、実行中のすべての SnapMirror 転送を中止します。デスティネーションボリュームが読み取り / 書き込み可能になるように、デスティネーションボリュームとソースボリュームの間のレプリケーション関係を解除します。
2. デスティネーションSVMから、Kubernetesが含まれているボリュームをマウントします etcd ホストに保存されるデータファイルと証明書で、マスターノードとして設定されます。
3. Kubernetesクラスタに関連する必要な証明書をのにすべてコピーします /etc/kubernetes/pki そして etcd member のファイル /var/lib/etcd。
4. を実行してKubernetesクラスタを作成します kubeadm init コマンドにを指定します --ignore -preflight-errors=DirAvailable-var-lib-etcd フラグ。ホスト名はソースの Kubernetes クラスタと同じにする必要があります。
5. を実行します kubectl get crd すべてのTridentカスタムリソースが稼働しているかどうかを確認するコマンドです。すべてのデータが利用可能かどうかを確認するためにTridentオブジェクトを取得します。
6. 前のバックエンドをクリーンアップし、Trident に新しいバックエンドを作成します。デスティネーション SVM の新しい管理 LIF とデータ LIF 、新しい SVM 名、パスワードを指定します。

アプリケーション永続ボリュームのディザスタリカバリワークフロー

次の手順は、災害発生時に SnapMirror デスティネーションボリュームをコンテナ化されたワークロードで使用できるようにする方法を示しています。

1. スケジュールされたすべての SnapMirror 転送を中止し、実行中のすべての SnapMirror 転送を中止します。デスティネーションボリュームが読み取り / 書き込み可能になるように、デスティネーションボリュームとソースボリュームの間のレプリケーション関係を解除します。ソース SVM のボリュームにバインドされた PVC を使用していた環境をクリーンアップします。
2. 前述の手順に従ってデスティネーション側で Kubernetes クラスタをセットアップしたら、Kubernetes クラスタから導入環境、PVC、PV をクリーンアップします。

3. Trident で新しい管理 LIF とデータ LIF、デスティネーション SVM の新しい SVM 名とパスワードを指定して、新しいバックエンドを作成します。
4. Trident のインポート機能を使用して、必要なボリュームを、新しい PVC にバインドされた PV としてインポートします。
5. 新しく作成した PVC を使用してアプリケーション展開を再展開します。

Element Snapshot を使用してデータをリカバリします

ボリュームの Snapshot スケジュールを設定し、必要な間隔で Snapshot が作成されていることを確認して、Element ボリューム上のデータをバックアップします。Snapshot スケジュールは、Element UI または API を使用して設定します。現在、を使用してボリュームに Snapshot スケジュールを設定することはできません `solidfire-san` ドライバ。

データが破損した場合は、特定の Snapshot を選択し、Element UI または API を使用してボリュームを手動で Snapshot にロールバックできます。その Snapshot の作成後にボリュームに対して行われた変更はすべて元に戻ります。

セキュリティ

ここに記載された推奨事項を参考に、Astra Tridentのインストールを安全に行ってください。

Astra Trident を独自のネームスペースで実行

アプリケーション、アプリケーション管理者、ユーザ、および管理アプリケーションが Astra Trident オブジェクト定義またはポッドにアクセスしないようにして、信頼性の高いストレージを確保し、悪意のあるアクティビティをブロックすることが重要です。

他のアプリケーションやユーザを Astra Trident から分離するには、Astra Trident を必ず独自の Kubernetes ネームスペースにインストールしてください (`trident`)。Astra Trident を独自の名前空間に配置することで、Kubernetes 管理担当者のみが Astra Trident ポッドにアクセスでき、名前空間 CRD オブジェクトに格納されたアーティファクト（バックエンドや CHAP シークレット（該当する場合））にアクセスできるようになります。

Astra Trident のネームスペースにアクセスできるのは管理者だけであることを確認してから、にアクセスできるようにしてください `tridentctl` アプリケーション：

ONTAP SAN バックエンドで CHAP 認証を使用します

Astra Trident は、ONTAP SAN ワークロードに対して（を使用して）CHAP ベースの認証をサポート (`ontap-san` および `ontap-san-economy` ドライバ)。ネットアップでは、ホストとストレージバックエンドの間の認証に、双方向 CHAP と Astra Trident を使用することを推奨しています。

SAN ストレージドライバを使用する ONTAP バックエンドの場合、Astra Trident は双方向 CHAP を設定し、を使用して CHAP ユーザ名とシークレットを管理できます `tridentctl`。
を参照してください ["こちらをご覧ください"](#) ONTAP バックエンドで Trident が CHAP を構成する方法をご確認ください。



ONTAP バックエンドの CHAP サポートは Trident 20.04 以降で利用可能

NetApp HCI および SolidFire バックエンドで CHAP 認証を使用します

ホストと NetApp HCI バックエンドと SolidFire バックエンドの間の認証を確保するために、双方向の CHAP を導入することを推奨します。Astra Trident は、テナントごとに 2 つの CHAP パスワードを含むシークレットオブジェクトを使用します。Trident を CSI プロビジョニングツールとしてインストールすると、CHAP シークレットが管理され、に格納されます `tridentvolume` 対応する PV の CR オブジェクト。PV を作成すると、CSI Astra Trident は CHAP シークレットを使用して iSCSI セッションを開始し、CHAP を介して NetApp HCI および SolidFire システムと通信します。



CSI Trident によって作成されたボリュームは、どのボリュームアクセスグループにも関連付けられていません。

CSI 以外のフロントエンドでは、ワーカーノード上のデバイスとしてのボリュームの接続は Kubernetes で処理されます。ボリュームの作成後、Astra Trident が NetApp HCI / SolidFire システムに対して API 呼び出しを実行し、テナントのシークレットがない場合はシークレットを取得します。Trident が Kubernetes にシークレットを渡します。各ノード上の kubelet は Kubernetes API を介してシークレットにアクセスし、ボリュームにアクセスする各ノードとボリュームが配置されている NetApp HCI / SolidFire システム間で CHAP を実行 / 有効化するために使用します。

NVE および NAE で Astra Trident を使用する

NetApp ONTAP は、保管データの暗号化を提供し、ディスクが盗難、返却、転用された場合に機密データを保護します。詳細については、を参照してください ["NetApp Volume Encryption の設定の概要"](#)。

- NAE がバックエンドで有効になっている場合は、Astra Trident でプロビジョニングされたすべてのボリュームが NAE に対応します。
- NAE がバックエンドで有効になっていない場合、NVE 暗号化フラグをに設定していないかぎり、Astra Trident でプロビジョニングされたすべてのボリュームが NVE 対応になります `false` バックエンド構成

NAE 対応バックエンドの Astra Trident で作成されるボリュームは、NVE または NAE で暗号化されている必要があります。



- NVE 暗号化フラグはに設定できます `true` Trident バックエンド構成で NAE 暗号化を無効にし、ボリューム単位で特定の暗号化キーを使用します。
- NVE 暗号化フラグをに設定する `false` NAE が有効なバックエンドでは、NAE が有効なボリュームが作成されます。NAE 暗号化を無効にするには、NVE 暗号化フラグをに設定します `false`。

- 明示的に NVE 暗号化フラグをに設定することで、Astra Trident で NVE ボリュームを手動で作成できます `true`。

バックエンド構成オプションの詳細については、以下を参照してください。

- ["ONTAP の SAN 構成オプション"](#)
- ["ONTAP NAS の構成オプション"](#)

Linux Unified Key Setup (LUKS ; 統合キーセットアップ) を使用したボリューム単位 のホスト側暗号化を有効にする

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"**。

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"**

作業を開始する前に

- ワーカーノードにはcryptsetup 2.1以上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) "**。

手順

1. バックエンド構成でLUKS暗号化属性を定義します。

```
"storage": [  
  {  
    "labels":{"luks": "true"},  
    "zone":"us_east_1a",  
    "defaults": {  
      "luksEncryption": "true"  
    }  
  },  
  {  
    "labels":{"luks": "false"},  
    "zone":"us_east_1a",  
    "defaults": {  
      "luksEncryption": "false"  
    }  
  },  
]
```

2. 使用 `parameters.selector` LUKS暗号化を使用してストレージプールを定義する方法。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: netapp.io/trident
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

3. LUKSパスフレーズを含むシークレットを作成します。例：

```
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
```

制限

- LUKS暗号化されたボリュームは、ONTAP 重複排除と圧縮を利用できません。
- 現時点では、LUKSパスフレーズのローテーションはサポートされていません。パスフレーズを変更するには、PVC間でデータを手動でコピーします。

参照

Astra Trident ポート

Trident が通信するポートの詳細をご確認ください。

Astra Trident は次のポート経由で通信：

ポート	目的
8443	バックチャネル HTTPS
8001	Prometheus 指標エンドポイント
8、000	Trident REST サーバ
17546	Trident デミ作用 / レディネスプローブポートは、Trident デミ作用ポッドで使用されます



活性/レディネスプローブポートは、を使用してインストール時に変更できます `--probe -port` フラグ。このポートがワーカーノード上の別のプロセスで使用されていないことを確認することが重要です。

Astra Trident REST API

間 "[tridentctl コマンドとオプション](#)" Trident REST APIを使用するには、RESTエンドポイントを直接使用する方法が最も簡単です。

REST APIを使用する状況

REST APIは、Kubernetes以外の環境でAstra Tridentをスタンドアロンバイナリとして使用する高度なインストールに役立ちます。

セキュリティ強化のため、Astra Tridentをぜひご利用ください REST API ポッド内で実行されている場合は、デフォルトでlocalhostに制限されます。この動作を変更するには、Astra Tridentを設定する必要があります `-address` 引数をポッド構成で指定します。

REST APIを使用する

API は次のように機能します。

GET

- GET `<trident-address>/trident/v1/<object-type>`:そのタイプのすべてのオブジェクトを一覧表示します。
- GET `<trident-address>/trident/v1/<object-type>/<object-name>`:名前付きオブジェクトの詳細を取得します。

POST

POST <trident-address>/trident/v1/<object-type>:指定した型のオブジェクトを作成します。

- オブジェクトを作成するには JSON 構成が必要です。各オブジェクトタイプの仕様については、tridentctl.htmlを参照してください[tridentctl コマンドとオプション]。
- オブジェクトがすでに存在する場合、動作は一定ではありません。バックエンドが既存のオブジェクトを更新しますが、それ以外のすべてのオブジェクトタイプで処理が失敗します。

DELETE

DELETE <trident-address>/trident/v1/<object-type>/<object-name>:指定したリソースを削除します。



バックエンドまたはストレージクラスに関連付けられているボリュームは削除されず、削除されません。詳細については、tridentctl.htmlを参照してください[tridentctl コマンドとオプション]。

これらのAPIの呼び出し方法の例については、デバッグを渡してください(-d) フラグ。詳細については、tridentctl.htmlを参照してください[tridentctl コマンドとオプション]。

コマンドラインオプション

Trident は、Trident オーケストレーションツールのコマンドラインオプションをいくつか公開しています。これらのオプションを使用して、導入環境を変更できます。

ロギング

- -debug:デバッグ出力をイネーブルにします。
- -loglevel <level>:ログレベルを設定します(デバッグ、情報、警告、エラー、致命的)。デフォルトは info です。

Kubernetes

- -k8s_pod: このオプションまたはを使用します -k8s_api_server をクリックしてKubernetesのサポートを有効にしこれを設定すると、Trident はポッドのKubernetes サービスアカウントのクレデンシャルを使用してAPI サーバに接続します。これは、サービスアカウントが有効になっている Kubernetes クラスタでTrident がポッドとして実行されている場合にのみ機能します。
- -k8s_api_server <insecure-address:insecure-port>: このオプションまたはを使用します -k8s_pod をクリックしてKubernetesのサポートを有効にしTrident を指定すると、セキュアでないアドレスとポートを使用して Kubernetes API サーバに接続されます。これにより、Trident をポッドの外部に導入することができますが、サポートされるのはAPI サーバへのセキュアでない接続だけです。セキュアに接続するには、Tridentをポッドに搭載し、を使用して導入します -k8s_pod オプション
- -k8s_config_path <file>:必須。このパスをKubeConfigファイルに指定する必要があります。

Docker です

- `-volume_driver <name>`: Dockerプラグインの登録時に使用するドライバ名。デフォルトは `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サーバがリッ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) のみをリッスンして処理するように設定できます。

- `-port <port-number>`: TridentのRESTサーバがリッスンするポートを指定します。デフォルトは 8000 です。
- `-rest`: RESTインターフェイスを有効にします。デフォルトは `true` です。

ネットアップの製品が **Kubernetes** と統合されます

ネットアップのストレージ製品ポートフォリオは、Kubernetes クラスタのさまざまな要素と統合され、高度なデータ管理機能を提供して、Kubernetes 環境の機能、機能、パフォーマンス、可用性を強化します。

アストラ

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

ONTAP

ONTAP は、あらゆるアプリケーションに高度なデータ管理機能を提供する、ネットアップのマルチプロトコルユニファイドストレージオペレーティングシステムです。ONTAP システムには、オールフラッシュ、ハイブリッド、オール HDD のいずれかの構成が採用されており、自社開発のハードウェア (FAS と AFF)、ノーブランド製品 (ONTAP Select)、クラウドのみ (Cloud Volumes ONTAP) など、さまざまな導入モデルが用意されています。



Trident は、上記の ONTAP 導入モデルをすべてサポートしています。

Cloud Volumes ONTAP

"Cloud Volumes ONTAP" は、クラウドで ONTAP データ管理ソフトウェアを実行するソフトウェア型ストレージアプライアンスです。Cloud Volumes ONTAP は、本番ワークロード、ディザスタリカバリ、DevOps、ファイル共有、データベース管理に使用できます。ストレージ効率、高可用性、データレプリケーション、データ階層化、アプリケーションの整合性を提供することで、エンタープライズストレージをクラウドに拡張します。

NetApp ONTAP 対応の Amazon FSX

"NetApp ONTAP 対応の Amazon FSX" は、NetApp ONTAP ストレージ・オペレーティング・システムを搭載したファイル・システムの起動と実行を可能にする、フルマネージドの AWS サービスです。FSX for ONTAP を使用すると、お客様は使い慣れたネットアップの機能、パフォーマンス、管理機能を活用しながら、AWS にデータを格納する際のシンプルさ、即応性、セキュリティ、拡張性を活用できます。FSX for ONTAP は、ONTAP のファイルシステム機能と管理 API の多くをサポートしています。

Element ソフトウェア

"要素 (Element)" ストレージ管理者は、パフォーマンスを保証し、ストレージの設置面積を合理化することで、ワークロードを統合できます。Element と API を組み合わせることでストレージ管理のあらゆる要素を自動化できるため、ストレージ管理者は少ない労力で多くの作業を行うことができます。

NetApp HCI

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

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

Azure NetApp Files の特長

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

Cloud Volumes Service for Google Cloud

"NetApp Cloud Volumes Service for Google Cloud" は、NFS や SMB 経由で NAS ボリュームにオールフラッシュのパフォーマンスを提供する、クラウドネイティブのファイルサービスです。このサービスにより、従来型アプリケーションを含むあらゆるワークロードをGCPクラウドで実行できます。フルマネージドサービスを提供し、一貫したハイパフォーマンス、瞬時のクローニング、データ保護、Google Compute Engine (GCE) インスタンスへのセキュアなアクセスを実現します。

Kubernetes オブジェクトと Trident オブジェクト

リソースオブジェクトの読み取りと書き込みを行うことで、REST API を使用して Kubernetes や Trident を操作できます。Kubernetes と Trident、Trident とストレージ、Kubernetes とストレージの関係を決定するリソースオブジェクトがいくつかあり

ます。これらのオブジェクトの中には Kubernetes で管理されるものと Trident で管理されるものがあります。

オブジェクトは相互にどのように相互作用しますか。

おそらく、オブジェクト、その目的、操作方法を理解する最も簡単な方法は、Kubernetes ユーザからのストレージ要求を 1 回だけ処理することです。

1. ユーザがを作成します PersistentVolumeClaim 新しいを要求しています PersistentVolume 特定のサイズのを Kubernetes から取得します StorageClass 以前に管理者によって設定されていたもの。
2. Kubernetes StorageClass Trident をプロビジョニングツールとして特定し、要求されたクラスのボリュームのプロビジョニング方法を Trident に指示するパラメータを設定します。
3. Trident はその外観を独自にしています StorageClass 一致するものと同じ名前を使用します Backends および StoragePools を使用して、クラスのボリュームをプロビジョニングできます。
4. Trident は、一致するバックエンドにストレージをプロビジョニングし、2 つのオブジェクトを作成します。A PersistentVolume Kubernetes で、ボリュームと Trident 内のボリュームを検出、マウント、処理し、間の関係を保持する方法を指示します PersistentVolume 実際のストレージをサポートします。
5. Kubernetes がをバインド PersistentVolumeClaim を新しいに変更します PersistentVolume。を含むポッド PersistentVolumeClaim この PersistentVolume を、実行されている任意のホストにマウントします。
6. ユーザがを作成します VolumeSnapshot を使用した既存の PVC の VolumeSnapshotClass Trident を指しています。
7. Trident が PVC に関連付けられているボリュームを特定し、バックエンドにボリュームの Snapshot を作成します。また、を作成します VolumeSnapshotContent これにより、Snapshot の識別方法を Kubernetes に指示します。
8. ユーザはを作成できます PersistentVolumeClaim を使用します VolumeSnapshot をソースとして使用します。
9. Trident が必要な Snapshot を特定し、の作成と同じ手順を実行します PersistentVolume および Volume。



Kubernetes オブジェクトの詳細については、を参照することを強く推奨します "[永続ボリューム](#)" Kubernetes のドキュメントのセクション。

Kubernetes PersistentVolumeClaim オブジェクト

Kubernetes を PersistentVolumeClaim オブジェクトは、Kubernetes クラスターユーザが作成するストレージの要求です。

Trident では、標準仕様に加えて、バックエンド構成で設定したデフォルト設定を上書きする場合に、ボリューム固有の次のアノテーションを指定できます。

アノテーション	ボリュームオプション	サポートされているドライバ
trident.netapp.io/fileSystem	ファイルシステム	ONTAP-SAN、solidfire-san-エコノミー 構成、solidfire-san-SAN 間にある SolidFire を実現します

アノテーション	ボリュームオプション	サポートされているドライバ
trident.netapp.io/cloneFromPVC	cloneSourceVolume の実行中です	ONTAP-NAS、 ontap-san、solidfire-san、azure- netapp-files、gcp-cvs、 ONTAP - SAN - 経済性
trident.netapp.io/splitOnClone	splitOnClone	ONTAP - NAS 、 ONTAP - SAN
trident.netapp.io/protocol	プロトコル	任意
trident.netapp.io/exportPolicy	エクスポートポリシー	ONTAP-NAS、 ontap-nas-economy、ontap-nas- flexgroup
trident.netapp.io/snapshotPolicy	Snapshot ポリシー	ONTAP-NAS、 ontap-nas-economy、ontap-nas- flexgroup、ontap-SAN
trident.netapp.io/snapshotReserve	Snapshot リザーブ	ONTAP-NAS、 ontap-nas-flexgroup、ontap-san 、 gcp-cvs
trident.netapp.io/snapshotDirectory	snapshotDirectory の略	ONTAP-NAS、 ontap-nas-economy、ontap-nas- flexgroup
trident.netapp.io/unixPermissions	unixPermissions	ONTAP-NAS、 ontap-nas-economy、ontap-nas- flexgroup
trident.netapp.io/blockSize	ブロックサイズ	solidfire - SAN

作成されたPVにがある場合 Delete ポリシーを再利用すると、PVが解放されたとき（つまり、ユーザがPVCを削除したとき）に、TridentはPVと元のボリュームの両方を削除します。削除操作が失敗した場合、TridentはPVをマークします。そのような状態で操作が成功するか、PVが手動で削除されるまで、定期的に再試行します。PVがを使用している場合 Retain Tridentはポリシーを無視し、管理者がKubernetesとバックエンドからクリーンアップすることを前提としているため、ボリュームを削除する前にバックアップや検査を実行できます。PVを削除しても、原因 Trident で元のボリュームが削除されないことに注意してください。REST APIを使用して削除する必要があります (tridentctl)。

Trident では CSI 仕様を使用したボリュームスナップショットの作成がサポートされています。ボリュームスナップショットを作成し、それをデータソースとして使用して既存の PVC のクローンを作成できます。これにより、PVS のポイントインタイムコピーを Kubernetes にスナップショットの形で公開できます。作成した Snapshot を使用して新しい PVS を作成できます。を参照してください On-Demand Volume Snapshots これがどのように機能するかを確認します。

Tridentが提供するのも cloneFromPVC および splitOnClone クローンを作成するためのアノテーションこれらのアノテーションを使用すると、CSI実装（Kubernetes 1.13以前）を使用せずにPVCをクローニングできます。また、KubernetesリリースでボリュームSnapshotのベータ版がサポートされていない場合（Kubernetes 1.16以前）も同様です。Trident 19.10 は、PVCからのクローニングのCSIワークフローをサポートしていることに注意してください。



を使用できます cloneFromPVC および splitOnClone CSI Tridentの注釈と従来のCSI以外のフロントエンド。

次に例を示します。ユーザがすでにというPVCを持っている場合 `mysql`` を使用すると、ユーザはという新しいPVCを作成できます ``mysqlclone` などのアノテーションを使用する `trident.netapp.io/cloneFromPVC: mysql。` このアノテーションセットを使用すると、Trident はボリュームをゼロからプロビジョニングするのではなく、MySQL PVC に対応するボリュームのクローンを作成します。

次の点を考慮してください。

- アイドルボリュームのクローンを作成することを推奨します。
- PVC とそのクローンは、同じ Kubernetes ネームスペースに存在し、同じストレージクラスを持つ必要があります。
- を使用 `ontap-nas` および `ontap-san` ドライバが必要な場合は、PVC注釈を設定することをお勧めします `trident.netapp.io/splitOnClone` と組み合わせて使用します `trident.netapp.io/cloneFromPVC。` を使用 `trident.netapp.io/splitOnClone` をに設定します `true`` Tridentでは、クローニングされたボリュームを親ボリュームからスプリットするため、ストレージ効率を維持しないまま、クローニングされたボリュームのライフサイクルを完全に分離します。設定されていません ``trident.netapp.io/splitOnClone` またはに設定します `false` 親ボリュームとクローンボリューム間の依存関係を作成するのではなく、バックエンドのスペース消費が削減されます。そのため、クローンを先に削除しないかぎり親ボリュームを削除できません。クローンをスプリットするシナリオでは、空のデータベースボリュームをクローニングする方法が効果的です。このシナリオでは、ボリュームとそのクローンで使用するデータベースボリュームのサイズが大きく異なっており、ONTAPではストレージ効率化のメリットはありません。

。 `sample-input Directory`には、Tridentで使用するPVC定義の例が含まれています。Trident ボリュームに関連するパラメータと設定の完全な概要については、Trident ボリュームオブジェクトを参照してください。

Kubernetes PersistentVolume オブジェクト

Kubernetesを PersistentVolume オブジェクトは、Kubernetesクラスタで使用可能になるストレージを表します。ポッドに依存しないライフサイクルがあります。



Tridentが実現 PersistentVolume オブジェクトを作成し、プロビジョニングするボリュームに基づいてKubernetesクラスタに自動的に登録します。自分で管理することは想定されていません。

Tridentベースを参照するPVCを作成する場合 ``StorageClass`` Tridentは、対応するストレージクラスを使用して新しいボリュームをプロビジョニングし、そのボリュームに新しいPVを登録します。プロビジョニングされたボリュームと対応する PV の構成では、Trident は次のルールに従います。

- Trident は、Kubernetes に PV 名を生成し、ストレージのプロビジョニングに使用する内部名を生成します。どちらの場合も、名前がスコープ内で一意であることが保証されます。
- ボリュームのサイズは、PVC で要求されたサイズにできるだけ近いサイズに一致しますが、プラットフォームによっては、最も近い割り当て可能な数量に切り上げられる場合があります。

Kubernetes StorageClass オブジェクト

Kubernetes StorageClass オブジェクトは、の名前で指定します PersistentVolumeClaims 一連のプロパティを指定してストレージをプロビジョニングします。ストレージクラス自体が、使用するプロビジョニングツールを特定し、プロビジョニングツールが理解できる一連のプロパティを定義します。

管理者が作成および管理する必要がある 2 つの基本オブジェクトのうちの 1 つです。もう 1 つは Trident バックエンドオブジェクトです。

Kubernetesを StorageClass Tridentを使用するオブジェクトは次のようになります。

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

これらのパラメータは Trident 固有で、クラスのボリュームのプロビジョニング方法を Trident に指示します。

ストレージクラスのパラメータは次のとおりです。

属性	を入力します	必須	説明
属性（Attributes）	[string] 文字列をマップします	いいえ	後述の「属性」セクションを参照してください
ストレージプール	[string] StringList をマップします	いいえ	バックエンド名とリストのマッピング ストレージプール
AdditionalStoragePools	[string] StringList をマップします	いいえ	バックエンド名のマップ ストレージプールノリスト
excludeStoragePools	[string] StringList をマップします	いいえ	バックエンドメイノマップ ストレージプールノリスト

ストレージ属性とその有効な値は、ストレージプールの選択属性と Kubernetes 属性に分類できます。

ストレージプールの選択の属性

これらのパラメータは、特定のタイプのボリュームのプロビジョニングに使用する Trident で管理されているストレージプールを決定します。

属性	を入力します	値	提供	リクエスト	でサポートされます
メディア ^1	文字列	HDD、ハイブリッド、SSD	プールにはこのタイプのメディアが含まれています。ハイブリッドは両方を意味します	メディアタイプが指定されました	ONTAPNAS、ONTAPNAS エコノミー、ONTAP-NAS-flexgroup、ONTAPSAN、solidfire-san-SAN、solidfire-san-SAN のいずれかに対応しています
プロビジョニングタイプ	文字列	シン、シック	プールはこのプロビジョニング方法をサポートします	プロビジョニング方法が指定されました	シック：All ONTAP；thin：All ONTAP & solidfire-san-SAN
backendType	文字列	ONTAPNAS、ONTAPNASエコノミー、ONTAP-NAS-flexgroup、ONTAPSAN、solidfire-san-SAN、solidfire-san-SAN、GCP-cvs、azure-NetApp-files、ONTAP-SAN-bエコノミー	プールはこのタイプのバックエンドに属しています	バックエンドが指定されて	すべてのドライバ
Snapshot	ブール値	true false	プールは、Snapshot を含むボリュームをサポートします	Snapshot が有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
クローン	ブール値	true false	プールはボリュームのクローニングをサポートします	クローンが有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
暗号化	ブール値	true false	プールでは暗号化されたボリュームをサポート	暗号化が有効なボリューム	ONTAP-NAS、ONTAP-NAS-エコノミー、ONTAP-NAS-FlexArray グループ、ONTAP-SAN

属性	を入力します	値	提供	リクエスト	でサポートされます
IOPS	整数	正の整数	プールは、この範囲内で IOPS を保証する機能を備えています	ボリュームで IOPS が保証されました	solidfire - SAN

^1 ^ : ONTAP Select システムではサポートされていません

ほとんどの場合、要求された値はプロビジョニングに直接影響します。たとえば、シックプロビジョニングを要求した場合、シックプロビジョニングボリュームが使用されます。ただし、Element ストレージプールでは、提供されている IOPS の最小値と最大値を使用して、要求された値ではなく QoS 値を設定します。この場合、要求された値はストレージプールの選択のみに使用されます。

理想的には、を使用できます attributes 特定のクラスのニーズを満たすために必要なストレージの品質をモデル化することだけを目的としています。Tridentは、の_all_に一致するストレージプールを自動的に検出して選択します attributes を指定します。

自分が使用できない場合は attributes クラスに適したプールを自動的に選択するには、を使用します storagePools および additionalStoragePools プールをさらに細かく指定するためのパラメータ、または特定のプールセットを選択するためのパラメータ。

を使用できます storagePools 指定したパラメータに一致するプールをさらに制限します attributes。つまり、Tridentはによって識別されたプールの交点を使用します attributes および storagePools プロビジョニングのパラメータ。どちらか一方のパラメータを単独で使用することも、両方を同時に使用することも

を使用できます additionalStoragePools Tridentがプロビジョニングに使用する一連のプールを、で選択されているプールに関係なく拡張するためのパラメータ attributes および storagePools パラメータ

を使用できます excludeStoragePools Tridentがプロビジョニングに使用する一連のプールをフィルタリングするためのパラメータ。このパラメータを使用すると、一致するプールがすべて削除されます。

を参照してください storagePools および additionalStoragePools パラメータを指定すると、各エントリの形式がになります <backend>:<storagePoolList>、ここで <storagePoolList> は、指定したバックエンドのストレージプールをカンマで区切ったリストです。たとえば、の値などで

additionalStoragePools 次のように表示されます

```
ontapnas_192.168.1.100:aggr1,aggr2;solidfire_192.168.1.101:bronze.
```

これらのリストでは、バックエンド値とリスト値の両方に正規表現値を使用できます。を使用できます tridentctl get backend バックエンドとそのプールのリストを取得します。

Kubernetes の属性

これらの属性は、動的プロビジョニングの際に Trident が選択するストレージプール / バックエンドには影響しません。代わりに、Kubernetes Persistent Volume でサポートされるパラメータを提供するだけです。ワーカーノードはファイルシステムの作成操作を担当し、xfsprogs などのファイルシステムユーティリティを必要とする場合があります。

属性	を入力します	値	説明	関連するドライバ	Kubernetes バージョン
----	--------	---	----	----------	------------------

FSstypе (英語)	文字列	ext4、ext3、xfs など	ブロックのファイルシステムタイプのボリューム	solidfire-san-group、ontap/nas、ontap-nas-エコノミー、ontap-nas-flexgroup、ontap-san、ONTAP-SAN-経済性	すべて
allowVolumeExpansion の略	ブール値	true false	PVC サイズの拡張のサポートをイネーブルまたはディセーブルにします	ONTAPNAS、ONTAPNAS エコノミー、ONTAP-NAS-flexgroup、ONTAPSAN、ONTAP-SAN-エコノミー、solidfire-san-gcvs、azure-netapp-files	1.11 以上
volumeBindingMode のようになりました	文字列	即時、WaitForFirstConsumer	ボリュームバインドと動的プロビジョニングを実行するタイミングを選択します	すべて	1.19 ~ 1.25

- 。 fsType パラメータは、SAN LUNに必要なファイルシステムタイプを制御する場合に使用します。また、Kubernetesでは、の機能も使用されます fsType ファイルシステムが存在することを示すために、ストレージクラスに格納します。ボリューム所有権は、を使用して制御できます fsGroup ポッドのセキュリティコンテキスト（使用する場合のみ） fsType が設定されます。を参照してください ["Kubernetes：ポッドまたはコンテナのセキュリティコンテキストを設定します"](#) を使用したボリューム所有権の設定の概要については、を参照してください fsGroup コンテキスト（Context）。Kubernetesでが適用されます fsGroup 次の場合のみ値を指定します

- fsType はストレージクラスで設定されます。

- PVC アクセスモードは RWO です。

NFS ストレージドライバの場合、NFS エクスポートにはファイルシステムがすでに存在します。を使用します fsGroup ストレージクラスでは、引き続きを指定する必要があります fsType。に設定できます nfs またはnull以外の値。

- を参照してください ["ボリュームを展開します"](#) ボリューム拡張の詳細については、を参照してください。
- Tridentのインストーラバンドルには、でTridentで使用するストレージクラス定義の例がいくつか含まれています sample-input/storage-class-*.yaml。Kubernetes ストレージクラスを削除すると、対応する Trident ストレージクラスも削除されます。



Kubernetes VolumeSnapshotClass オブジェクト

Kubernetes VolumeSnapshotClass オブジェクトはに似ています StorageClasses。この Snapshot コピーは、複数のストレージクラスの定義に役立ちます。また、ボリューム Snapshot によって参照され、Snapshot を必要な Snapshot クラスに関連付けます。各ボリューム Snapshot は、単一のボリューム Snapshot クラスに関連付けられます。

A VolumeSnapshotClass Snapshotを作成するには、管理者によって定義されている必要があります。ボリューム Snapshot クラスは、次の定義で作成されます。

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

。 driver のボリュームSnapshotを要求するKubernetesに指定します csi-snapclass クラスはTridentによって処理されます。。 deletionPolicy Snapshotを削除する必要がある場合に実行する処理を指定します。いつ deletionPolicy がに設定されます Delete`を指定すると、Snapshotが削除されたときに、ボリュームSnapshotオブジェクトおよびストレージクラス上の基盤となるSnapshotが削除されます。または、に設定します `Retain はそのことを示します VolumeSnapshotContent 物理スナップショットが保持されます。

Kubernetes VolumeSnapshot オブジェクト

Kubernetesを VolumeSnapshot objectは、ボリュームのSnapshotを作成する要求です。PVC がボリュームに対するユーザからの要求を表すのと同様に、ボリュームスナップショットは、ユーザが既存の PVC のスナップショットを作成する要求です。

ボリュームSnapshot要求が開始されると、TridentはバックエンドでのボリュームのSnapshotの作成を自動的に管理し、一意のを作成してSnapshotを公開します

VolumeSnapshotContent オブジェクト。既存の PVC からスナップショットを作成し、新しい PVC を作成するときにスナップショットを DataSource として使用できます。



VolumeSnapshot のライフサイクルはソース PVC とは無関係です。ソース PVC が削除されても、スナップショットは維持されます。スナップショットが関連付けられている PVC を削除すると、Trident はその PVC のバックアップボリュームを **Deleting** 状態でマークしますが、完全には削除しません。関連付けられている Snapshot がすべて削除されると、ボリュームは削除されます。

Kubernetes VolumeSnapshotContent オブジェクト

Kubernetesを VolumeSnapshotContent オブジェクトは、すでにプロビジョニングされているボリュームから作成されたSnapshotを表します。これはに似ています PersistentVolume とは、ストレージクラスにプロビジョニングされたSnapshotを表します。に似ています PersistentVolumeClaim および PersistentVolume オブジェクト。スナップショットが作成されると、が表示されます

VolumeSnapshotContent オブジェクトは、への1対1のマッピングを保持します VolumeSnapshot オブジェクト。オブジェクトはSnapshotの作成を要求しました。



Tridentが実現 VolumeSnapshotContent オブジェクトを作成し、プロビジョニングするボリュームに基づいてKubernetesクラスタに自動的に登録します。自分で管理することは想定されていません。

。VolumeSnapshotContent Objectには、など、Snapshotを一意に識別する詳細が含まれます snapshotHandle。これ snapshotHandle は、PVの名前との名前を一意に組み合わせたものです VolumeSnapshotContent オブジェクト。

Trident では、スナップショット要求を受信すると、バックエンドにスナップショットが作成されます。スナップショットが作成されると、Tridentによってが設定されます VolumeSnapshotContent オブジェクトを作成することで、SnapshotをKubernetes APIに公開します。

Kubernetes CustomResourceDefinition オブジェクト

Kubernetes カスタムリソースは、管理者が定義した Kubernetes API 内のエンドポイントであり、類似するオブジェクトのグループ化に使用されます。Kubernetes では、オブジェクトのコレクションを格納するためのカスタムリソースの作成をサポートしています。を実行すると、これらのリソース定義を取得できます `kubectl get crds`。

カスタムリソース定義（CRD）と関連するオブジェクトメタデータは、Kubernetes によってメタデータストアに格納されます。これにより、Trident の独立したストアが不要になります。

19.07リリース以降、Tridentはいくつかのを使用します CustomResourceDefinition Tridentバックエンド、Tridentストレージクラス、Tridentボリュームなど、TridentオブジェクトのIDを保持するオブジェクト。これらのオブジェクトは Trident によって管理されます。また、CSI のボリュームスナップショットフレームワークには、ボリュームスナップショットの定義に必要ないくつかの SSD が導入されています。

CRD は Kubernetes の構成要素です。上記で定義したリソースのオブジェクトは Trident によって作成されます。簡単な例として、を使用してバックエンドを作成する場合を示します `tridentctl`` に対応します ``tridentbackends` CRDオブジェクトは、Kubernetesによって消費されるために作成されます。

Trident の CRD については、次の点に注意してください。

- Trident をインストールすると、一連の CRD が作成され、他のリソースタイプと同様に使用できるようになります。
- 以前のバージョンのTrident（使用していたもの）からアップグレードする場合 `etcd` ステートを維持するために）、Tridentインストーラがからデータを移行します `etcd` キーバリュースタアと対応するCRDオブジェクトの作成。
- Tridentをアンインストールするには、を使用します `tridentctl uninstall` コマンドであるTridentポッドが削除されましたが、作成されたSSDはクリーンアップされません。を参照してください ["Trident をアンインストールします"](#) Trident を完全に削除して再構成する方法を理解する。

Trident StorageClass オブジェクト

TridentではKubernetesに対応するストレージクラスが作成されます StorageClass を指定するオブジェクト `csi.trident.netapp.io/netapp.io/trident` プロビジョニング担当者のフィールドに入力します。ストレージクラス名がKubernetesの名前と一致していること StorageClass 表すオブジェクト。



Kubernetesでは、これらのオブジェクトはKubernetesのときに自動的に作成されます StorageClass Tridentをプロビジョニングツールとして使用していることが登録されます。

ストレージクラスは、ボリュームの一連の要件で構成されます。Trident は、これらの要件と各ストレージプール内の属性を照合し、一致する場合は、そのストレージプールが、そのストレージクラスを使用するボリュームのプロビジョニングの有効なターゲットになります。

REST API を使用して、ストレージクラスを直接定義するストレージクラス設定を作成できます。ただし、Kubernetes環境では、新しいKubernetesを登録するときにKubernetes環境が作成されることを想定しています StorageClass オブジェクト。

Trident バックエンドオブジェクト

バックエンドとは、Trident がボリュームをプロビジョニングする際にストレージプロバイダを表します。1 つの Trident インスタンスであらゆる数のバックエンドを管理できます。



これは、自分で作成および管理する 2 つのオブジェクトタイプのうちの 1 つです。もう1つはKubernetesです StorageClass オブジェクト。

これらのオブジェクトの作成方法の詳細については、を参照してください ["バックエンドの設定"](#)。

Trident StoragePool オブジェクト

ストレージプールは、各バックエンドでのプロビジョニングに使用できる個別の場所を表します。ONTAP の場合、これらは SVM 内のアグリゲートに対応します。NetApp HCI / SolidFire では、管理者が指定した QoS 帯域に対応します。Cloud Volumes Service の場合、これらはクラウドプロバイダのリージョンに対応します。各ストレージプールには、パフォーマンス特性とデータ保護特性を定義するストレージ属性があります。

他のオブジェクトとは異なり、ストレージプールの候補は常に自動的に検出されて管理されます。

Trident Volume オブジェクト

ボリュームは、NFS 共有や iSCSI LUN などのバックエンドエンドエンドエンドポイントで構成される、プロビジョニングの基本単位です。Kubernetesでは、これらはに直接対応します PersistentVolumes。ボリュームを作成するときは、そのボリュームにストレージクラスが含まれていることを確認します。このクラスによって、ボリュームをプロビジョニングできる場所とサイズが決まります。



Kubernetes では、これらのオブジェクトが自動的に管理されます。Trident がプロビジョニングしたものを表示できます。



関連付けられた Snapshot がある PV を削除すると、対応する Trident ボリュームが * Deleting * 状態に更新されます。Trident ボリュームを削除するには、ボリュームの Snapshot を削除する必要があります。

ボリューム構成は、プロビジョニングされたボリュームに必要なプロパティを定義します。

属性	を入力します	必須	説明
バージョン	文字列	いいえ	Trident API のバージョン（「1」）
名前	文字列	はい。	作成するボリュームの名前
ストレージクラス	文字列	はい。	ボリュームのプロビジョニング時に使用するストレージクラス
サイズ	文字列	はい。	プロビジョニングするボリュームのサイズ（バイト単位）
プロトコル	文字列	いいえ	使用するプロトコルの種類：「file」または「block」
インターン名	文字列	いいえ	Trident が生成した、ストレージシステム上のオブジェクトの名前
cloneSourceVolume の実行中です	文字列	いいえ	ONTAP（NAS、SAN） & SolidFire - *：クローン元のボリュームの名前
splitOnClone	文字列	いいえ	ONTAP（NAS、SAN）：クローンを親からスプリットします
Snapshot ポリシー	文字列	いいえ	ONTAP - *：使用する Snapshot ポリシー
Snapshot リザーブ	文字列	いいえ	ONTAP - *：Snapshot 用にリザーブされているボリュームの割合
エクスポートポリシー	文字列	いいえ	ONTAP-NAS*：使用するエクスポートポリシー
snapshotDirectory の略	ブール値	いいえ	ONTAP-NAS*：Snapshot ディレクトリが表示されているかどうか
unixPermissions	文字列	いいえ	ONTAP-NAS*：最初の UNIX 権限
ブロックサイズ	文字列	いいえ	SolidFire - *：ブロック / セクターサイズ
ファイルシステム	文字列	いいえ	ファイルシステムのタイプ

Tridentが生成 internalName ボリュームを作成する場合。この構成は2つのステップで構成されます。最初に、ストレージプレフィックス（デフォルトのプレフィックス）を先頭に追加します trident またはバックエンド構成内のプレフィックス）をボリューム名に変更して、形式の名前を指定します <prefix>-<volume-name>。その後、名前の完全消去が行われ、バックエンドで許可されていない文字が置き換えられます。ONTAP バックエンドの場合、ハイフンをアンダースコアに置き換えます（内部名はになります）

<prefix>_<volume-name>)。Element バックエンドの場合、アンダースコアはハイフンに置き換えられます。

ボリューム構成を使用してREST APIを使用してボリュームを直接プロビジョニングできますが、Kubernetes 環境ではほとんどのユーザが標準のKubernetesを使用することを想定しています PersistentVolumeClaim メソッドプロビジョニングの一環としてTridentがこのボリュームオブジェクトを自動的に作成プロセス：

Trident Snapshot オブジェクト

Snapshot はボリュームのポイントインタイムコピーで、新しいボリュームのプロビジョニングやリストア状態に使用できます。Kubernetesでは、これらはに直接対応します VolumeSnapshotContent オブジェクト。各 Snapshot には、Snapshot のデータのソースであるボリュームが関連付けられます。

各 Snapshot オブジェクトには、次のプロパティが含まれます。

属性	を入力します	必須	説明
バージョン	文字列	はい。	Trident API のバージョン (「1」)
名前	文字列	はい。	Trident Snapshot オブジェクトの名前
インターン名	文字列	はい。	ストレージシステム上の Trident Snapshot オブジェクトの名前
ボリューム名	文字列	はい。	Snapshot を作成する永続的ボリュームの名前
ボリュームの内部名	文字列	はい。	ストレージシステムに関連付けられている Trident ボリュームオブジェクトの名前



Kubernetes では、これらのオブジェクトが自動的に管理されます。Trident がプロビジョニングしたものを表示できます。

Kubernetesを導入したとき VolumeSnapshot オブジェクト要求が作成されると、TridentはバックিংストレージシステムにSnapshotオブジェクトを作成することで機能します。。 internalName このSnapshotオブジェクトのプレフィックスを組み合わせると、が生成されます snapshot- を使用 UID の VolumeSnapshot オブジェクト (例： snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660) 。 volumeName および volumeInternalName 裏付けの詳細を取得することで入力されます。 ボリューム：

Astra Trident ResourceQuota オブジェクト

Tridentのデーモンは、を消費します system-node-critical 優先度クラス：Kubernetesで最も高い優先度クラスです。Astra Tridentは、ノードの正常なシャットダウン中にボリュームを識別してクリーンアップし、Tridentのデミスタポッドがリソースの負荷が高いクラスタでより低い優先度でワークロードをプリエンブトできるようにします。

そのために、Astra Tridentはを採用しています ResourceQuota Tridentのデミスタに対する「システムノー

ドクリティカル」の優先クラスを満たすことを保証するオブジェクト。導入とデマ作用の開始前に、Astra Tridentが探します ResourceQuota オブジェクトを検出し、検出されない場合は適用します。

デフォルトのリソースクォータおよび優先クラスをより詳細に制御する必要がある場合は、を生成できます custom.yaml またはを設定します ResourceQuota Helmチャートを使用するオブジェクト。

次に示すのは'ResourceQuota'オブジェクトがTridentのデマ作用を優先する例です

```
apiVersion: <version>
kind: ResourceQuota
metadata:
  name: trident-csi
  labels:
    app: node.csi.trident.netapp.io
spec:
  scopeSelector:
    matchExpressions:
      - operator : In
        scopeName: PriorityClass
        values: ["system-node-critical"]
```

リソース・クォータの詳細については'を参照してください "[Kubernetes：リソースクォータ](#)".

クリーンアップ ResourceQuota インストールが失敗した場合

まれに、のあとにインストールが失敗する場合があります ResourceQuota オブジェクトが作成されました。最初に実行してください "[アンインストール中です](#)" を再インストールします。

うまくいかない場合は、を手動で削除します ResourceQuota オブジェクト。

取り外します ResourceQuota

独自のリソース割り当てを制御する場合は、Astra Tridentを削除できます ResourceQuota 次のコマンドを使用したオブジェクトの削除：

```
kubectl delete quota trident-csi -n trident
```

tridentctl コマンドとオプション

。 "[Trident インストーラバンドル](#)" コマンドラインユーティリティを搭載しています。`tridentctl` Astra Tridentに簡単にアクセスできます。十分な権限を持つ Kubernetes ユーザは、このロールを使用して Astra Trident をインストールしたり、Astra Trident ポッドが含まれるネームスペースを直接管理したりできます。

使用可能なコマンドとオプション

使用方法については、を実行してください `tridentctl --help`。

使用可能なコマンドとグローバルオプションは次のとおりです。

```
Usage:
  tridentctl [command]
```

使用可能なコマンド：

- `create`：Astra Tridentにリソースを追加
- `delete`：Astra Tridentから1つ以上のリソースを削除
- `get`：Astra Tridentから1つ以上のリソースを入手
- `help`:任意のコマンドに関するヘルプ。
- `images`：Tridentが必要とするコンテナイメージを表で印刷
- `import`：既存のリソースをAstra Tridentにインポート
- `install`：Astra Tridentをインストール
- `logs`：Astra Tridentからログを印刷
- `send`：Astra Tridentからリソースを送信
- `uninstall`：TridentからAstraをアンインストール。
- `update`：Astra Tridentのリソースを変更
- `upgrade`：Astra Tridentのリソースをアップグレード
- `version`：Astra Tridentのバージョンを印刷

フラグ：

- `-d, --debug`:デバッグ出力。
- `-h, --help`:のヘルプ `tridentctl`。
- `-n, --namespace string`:Astra Tridentのネームスペースの導入。
- `-o, --output string`:出力形式。JSON の 1 つ | `yaml` | `name` | `wide` | `ps` （デフォルト）。
- `-s, --server string`：Astra Trident RESTインターフェイスのアドレス/ポート。



Trident REST インターフェイスは、127.0.0.1（IPv4 の場合）または `:::1`（IPv6 の場合）のみをリスンして処理するように設定できます。



Trident REST インターフェイスは、127.0.0.1（IPv4 の場合）または `:::1`（IPv6 の場合）のみをリスンして処理するように設定できます。

create

を実行します create Astra Tridentにリソースを追加するコマンド。

```
Usage:
  tridentctl create [option]
```

使用可能なオプション：

backend：Astra Tridentにバックエンドを追加

delete

を実行できます delete コマンドを使用して、Astra Tridentから1つ以上のリソースを削除します。

```
Usage:
  tridentctl delete [option]
```

使用可能なオプション：

- backend：Tridentから1つ以上のストレージバックエンドを削除
- snapshot：Astra Tridentから1つ以上のボリュームSnapshotを削除
- storageclass：Astra Tridentから1つ以上のストレージクラスを削除
- volume：Astra Tridentから1つ以上のストレージボリュームを削除

get

を実行できます get Astra Tridentから1つ以上のリソースを取得するためのコマンドです。

```
Usage:
  tridentctl get [option]
```

使用可能なオプション：

- backend：Tridentから1つ以上のストレージバックエンドを取得
- snapshot：Astra Tridentから1つ以上のスナップショットを取得
- storageclass：Astra Tridentから1つ以上のストレージクラスを取得
- volume：Astra Tridentから1つ以上のボリュームを取得

volume フラグ：

* -h, --help：ボリュームのヘルプ。

* --parentOfSubordinate string：クエリを下位のソースボリュームに制限します。

* --subordinateOf string：クエリをボリュームの下位に制限します。

images

を実行できます images Astra Tridentが必要とするコンテナイメージの表を印刷するためのフラグ。

```
Usage:
  tridentctl images [flags]
```

フラグ：

- * -h, --help` : Help for images.
- * -v, --k8s-version string` : Kubernetesクラスタのセマンティックバージョン。

import volume

を実行できます import volume コマンドを使用して、既存のボリュームをAstra Tridentにインポートします。

```
Usage:
  tridentctl import volume <backendName> <volumeName> [flags]
```

エイリアス：

volume, v

フラグ：

- -f, --filename string: YAMLまたはJSON PVCファイルへのパス。
- -h, --help : ボリュームのヘルプ。
- --no-manage : PV/PVCのみを作成します。ボリュームのライフサイクル管理を想定しないでください。

install

を実行できます install Astra Tridentのインストールにフラグを付けます。

```
Usage:
  tridentctl install [flags]
```

フラグ：

- --autosupport-image string : AutoSupport テレメトリのコンテナイメージ（デフォルトは「NetApp / Trident autosupport : 20.07.0」）。
- --autosupport-proxy string : AutoSupport テレメトリを送信するプロキシのアドレス/ポート。
- --csi : CSI Tridentをインストールします（Kubernetes 1.13のみを上書きします。機能ゲートが必要です）。

- `--enable-node-prep`: ノードに必要なパッケージをインストールします。
- `--generate-custom-yaml`: インストールを行わずにYAMLファイルを生成します。
- `-h, --help`: インストールのヘルプ。
- `--http-request-timeout`: TridentコントローラのREST APIのHTTP要求タイムアウトを上書きします (デフォルトは1分30秒)。
- `--image-registry string`: 内部イメージレジストリのアドレス/ポート。
- `--k8s-timeout duration`: すべてのKubernetes処理のタイムアウト (デフォルトは3分0)。
- `--kubelet-dir string`: kubeletの内部状態のホストの場所(デフォルトは/var/lib/kubelet)。
- `--log-format string`: Astra Tridentのログ形式(テキスト、JSON)(デフォルトは「text」)。
- `--pv string`: Astra Tridentが使用するレガシーPVの名前は、存在しないことを確認します(デフォルトは"trident")。
- `--pvc string`: Astra Tridentで使用されている従来のPVCの名前。このPVCが存在しないことを確認します (デフォルトは「trident」)。
- `--silence-autosupport`: AutoSupport バンドルを自動的にネットアップに送信しない (デフォルトはtrue)。
- `--silent`: インストール中は、ほとんどの出力を無効にします。
- `--trident-image string`: インストールするAstra Tridentのイメージ
- `--use-custom-yaml`: setupディレクトリに存在する既存のYAMLファイルを使用します。
- `--use-ipv6`: Astra Tridentの通信にIPv6を使用

logs

を実行できます logs Astra Tridentからログを印刷するためのフラグ。

```
Usage:
  tridentctl logs [flags]
```

フラグ:

- `-a, --archive`: 特に指定がないかぎり、すべてのログを含むサポートアーカイブを作成します。
- `-h, --help`: ログのヘルプ。
- `-l, --log string`: Astra Tridentのログが表示されます。trident | auto | trident-operator | all (デフォルトは「auto」) のいずれかです。
- `--node string`: ノードポッドログの収集元のKubernetesノード名。
- `-p, --previous`: 以前のコンテナインスタンスのログが存在する場合は、それを取得します。
- `--sidecars`: サイドカーコンテナのログを取得します。

send

を実行できます send Astra Tridentからリソースを送信するコマンド。

```
Usage:
  tridentctl send [option]
```

使用可能なオプション：

autosupport：ネットアップにAutoSupport アーカイブを送信します。

uninstall

を実行できます uninstall Astra Tridentをアンインストールするためのフラグ。

```
Usage:
  tridentctl uninstall [flags]
```

フラグ：

* -h, --help:アンインストールのヘルプ。

* --silent:アンインストール中のほとんどの出力を無効にします。

update

を実行できます update Astra Tridentのリソースを変更するコマンド。

```
Usage:
  tridentctl update [option]
```

使用可能なオプション：

backend：Astra Tridentのバックエンドを更新。

upgrade

を実行できます upgrade Astra Tridentのリソースをアップグレードするためのコマンド。

```
Usage:
  tridentctl upgrade [option]
```

使用可能なオプション：

volume：1つ以上の永続ボリュームをNFS/iSCSIからCSIにアップグレードします。

version

を実行できます version のバージョンを印刷するためのフラグ tridentctl 実行中のTridentサービス

Usage:

```
tridentctl version [flags]
```

フラグ:

* --client:クライアントバージョンのみ(サーバは不要)。

* -h, --help:バージョンのヘルプ。

PODセキュリティ標準 (PSS) およびセキュリティコンテキストの制約 (SCC)

Kubernetesポッドのセキュリティ標準 (PSS) とポッドのセキュリティポリシー (PSP) によって、権限レベルが定義され、ポッドの動作が制限されます。また、OpenShift Security Context Constraints (SCC) でも、OpenShift Kubernetes Engine固有のポッド制限を定義します。このカスタマイズを行うために、Astra Tridentはインストール時に特定の権限を有効にします。次のセクションでは、Astra Tridentによって設定された権限の詳細を説明します。



PSSは、Podセキュリティポリシー (PSP) に代わるものです。PSPはKubernetes v1.21で廃止され、v1.25で削除されます。詳細については、[を参照してください "Kubernetes：セキュリティ"](#)。

必須のKubernetes Security Contextと関連フィールド

アクセス権	説明
権限があります	CSIでは、マウントポイントが双方向である必要があります。つまり、Tridentノードポッドで特権コンテナを実行する必要があります。詳細については、 を参照してください "Kubernetes：マウントの伝播" 。
ホストネットワーク	iSCSIデーモンに必要です。iscsiadm iSCSIマウントを管理し、ホストネットワークを使用してiSCSIデーモンと通信します。
ホストIPC	NFSはIPC（プロセス間通信）を使用してnfsdと通信します
ホストPID	開始する必要があります rpc-statd NFSの場合 ：Astra Tridentがホストプロセスを照会して、状況を特定 rpc-statd を実行してからNFSボリュームをマウントしてください。

アクセス権	説明
機能	<p>。SYS_ADMIN この機能は、特権コンテナのデフォルト機能の一部として提供されます。たとえば、Dockerは特権コンテナに次の機能を設定します。</p> <p>CapPrm: 0000003fffffffffff CapEff: 0000003fffffffffff</p>
Seccom	<p>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。</p>
SELinux	<p>OpenShiftでは、特権のあるコンテナがで実行されず spc_t (「スーパー特権コンテナ」) ドメインおよび非特権コンテナは、で実行されます container_t ドメイン: オン containerd`を使用`container-selinux インストールすると、すべてのコンテナがで実行されます spc_t domain。SELinuxは無効になります。そのため、Astra Tridentは機能しません seLinuxOptions コンテナへ。</p>
DAC	<p>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。</p>

PODセキュリティ標準 (PSS)

ラベル	説明	デフォルト
pod-security.kubernetes.io/enforce	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。	enforce: privileged
pod-security.kubernetes.io/enforce-version	ネームスペースラベルは変更しないでください。	enforce-version: <version of the current cluster or highest version of PSS tested.>



名前空間ラベルを変更すると、ポッドがスケジュールされず、「Error creating: ...」または「Warning: trident-csi-...」が表示される場合があります。その場合は、のネームスペースラベルを確認してください privileged が変更されました。その場合は、Tridentを再インストールします。

PoDセキュリティポリシー (PSP)

フィールド	説明	デフォルト
allowPrivilegeEscalation	特権コンテナは、特権昇格を許可する必要があります。	true
allowedCSIDrivers	TridentはインラインCSIエフェメラルボリュームを使用しません。	空です

フィールド	説明	デフォルト
allowedCapabilities	権限のないTridentコンテナにはデフォルトよりも多くの機能が必要ないため、特権コンテナには可能なすべての機能が付与されます。	空です
allowedFlexVolumes	Tridentはを利用しません "FlexVolドライバ" そのため、これらのボリュームは許可されるボリュームのリストに含まれていません。	空です
allowedHostPaths	Tridentノードポッドでノードのルートファイルシステムがマウントされるため、このリストを設定してもメリットはありません。	空です
allowedProcMountTypes	Tridentでは使用していません ProcMountTypes。	空です
allowedUnsafeSysctls	Tridentでは安全でないリソースは不要です sysctls。	空です
defaultAddCapabilities	特権コンテナに追加する機能は必要ありません。	空です
defaultAllowPrivilegeEscalation	権限の昇格は、各Tridentポッドで処理されます。	false
forbiddenSysctls	いいえ sysctls 許可されています。	空です
fsGroup	Tridentコンテナはrootとして実行されます。	RunAsAny
hostIPC	NFSボリュームをマウントするには、ホストIPCがと通信する必要があります nfsd	true
hostNetwork	iscsiadmには、iSCSIデーモンと通信するためのホストネットワークが必要です。	true
hostPID	ホストPIDが必要かどうかを確認します rpc-statd ノードで実行されている。	true
hostPorts	Tridentはホストポートを使用しません。	空です
privileged	Tridentノードのポッドでは、ボリュームをマウントするために特権コンテナを実行する必要があります。	true
readOnlyRootFilesystem	Tridentノードのポッドは、ノードのファイルシステムに書き込む必要があります。	false

フィールド	説明	デフォルト
requiredDropCapabilities	Tridentノードのポッドは特権コンテナを実行するため、機能をドロップすることはできません。	none
runAsGroup	Tridentコンテナはrootとして実行されます。	RunAsAny
runAsUser	Tridentコンテナはrootとして実行されます。	runAsAny
runtimeClass	Tridentは使用しません RuntimeClasses。	空です
seLinux	Tridentが設定されていません seLinuxOptions 現在のところ、コンテナの実行時間とKubernetesのディストリビューションでのSELinuxの処理に違いがあるためです。	空です
supplementalGroups	Tridentコンテナはrootとして実行されます。	RunAsAny
volumes	Tridentポッドには、このボリュームプラグインが必要です。	hostPath, projected, emptyDir

セキュリティコンテキストの制約 (SCC)

ラベル	説明	デフォルト
allowHostDirVolumePlugin	Tridentノードのポッドは、ノードのルートファイルシステムをマウントします。	true
allowHostIPC	NFSボリュームをマウントするには、ホストIPCがと通信する必要があります nfsd。	true
allowHostNetwork	iscsiadmには、iSCSIデーモンと通信するためのホストネットワークが必要です。	true
allowHostPID	ホストPIDが必要かどうかを確認します rpc-statd ノードで実行されている。	true
allowHostPorts	Tridentはホストポートを使用しません。	false
allowPrivilegeEscalation	特権コンテナは、特権昇格を許可する必要があります。	true
allowPrivilegedContainer	Tridentノードのポッドでは、ボリュームをマウントするために特権コンテナを実行する必要があります。	true

ラベル	説明	デフォルト
allowedUnsafeSysctls	Tridentでは安全でないリソースは不要です <code>sysctls</code> 。	none
allowedCapabilities	権限のないTridentコンテナにはデフォルトよりも多くの機能が必要ないため、特権コンテナには可能なすべての機能が付与されます。	空です
defaultAddCapabilities	特権コンテナに追加する機能は必要ありません。	空です
fsGroup	Tridentコンテナはrootとして実行されます。	RunAsAny
groups	このSCCはTridentに固有で、ユーザにバインドされています。	空です
readOnlyRootFilesystem	Tridentノードのポッドは、ノードのファイルシステムに書き込む必要があります。	false
requiredDropCapabilities	Tridentノードのポッドは特権コンテナを実行するため、機能をドロップすることはできません。	none
runAsUser	Tridentコンテナはrootとして実行されます。	RunAsAny
seLinuxContext	Tridentが設定されていません <code>seLinuxOptions</code> 現在のところ、コンテナの実行時間とKubernetesのディストリビューションでのSELinuxの処理に違いがあるためです。	空です
seccompProfiles	特権のあるコンテナは常に「閉鎖的」な状態で実行されます。	空です
supplementalGroups	Tridentコンテナはrootとして実行されます。	RunAsAny
users	このSCCをTrident名前空間のTridentユーザにバインドするエントリが1つあります。	該当なし
volumes	Tridentポッドには、このボリュームプラグインが必要です。	hostPath, downwardAPI, projected, emptyDir

以前のバージョンのドキュメント

Astra Trident 22.10を実行していない場合は、以前のリリースのドキュメントを参照できます。

- ["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"](#)

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/"](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/"](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/"](https://www.netapp.com/company/legal/privacy-policy/)

オープンソース

通知ファイルには、ネットアップソフトウェアで使用するサードパーティの著作権およびライセンスに関する情報が記載されています。

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权を有し、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权については、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。