Astra Trident を使用

Astra Trident

NetApp
April 16, 2024

目次

Astra Trident を使用

バックエンドを設定

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

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

Azure NetApp Files バックエンドを設定します

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

> Azure NetApp Files サービスでは、 100GB 未満のボリュームはサポートされません。100 GB のボリュームが小さい場合は、 Trident が自動的に作成します。

必要なもの

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

- Azure NetApp Files が有効な Azure サブスクリプションのスクリプト ID 。
- 「 tenantID 」、「 clientID 」、「 clientSecret 」を「」から選択します "アプリケーション登録" Azure Active Directory で、 Azure NetApp Files サービスに対する十分な権限がある。App Registration では、 Azure で事前定義されている「 Owner 」または「 Contributor 」の役割を使用する必要があります。

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

- 少なくとも 1 つを含む Azure の「場所」 "委任されたサブネット"。Trident 22.01 では、「 location 」パラメータはバックエンド構成ファイルの最上位にある必須フィールドです。仮想プールで指定された場所の値は無視されます。
- Azure NetApp Files を初めて使用する場合や新しい場所で使用する場合は、いくつかの初期設定が必要です。を参照してください "クイックスタートガイド"。

このタスクについて

Trident は、バックエンド構成（サブネット、仮想ネットワーク、サービスレベル、場所）に基づいて、要求された場所で利用可能な容量プールに ANF ボリュームを作成し、要求されたサービスレベルとサブネットに

対応します。

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

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

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

PVC の作成時に「 No capacity pools found 」エラーが発生した場合、アプリケーション登録に必要な権限とリソース（サブネット、仮想ネットワーク、容量プール）が関連付けられていない可能性があります。Astra Trident は、デバッグが有効なときにバックエンドが作成されたときに、検出した Azure リソースをログに記録します。適切なロールが使用されているかどうかを確認してください。

NFS バージョン 4.1 を使用してボリュームをマウントする場合は、 NFS v4.1 を選択するために、カンマ区切りのマウントオプションリストに「 nfsvers=4` 」を含めることができます。ストレージクラスで設定されたマウントオプションは、バックエンド構成ファイルで設定されたマウントオプションよりも優先されます。

resourceGroups’netappAccounts’capacityPools’virtualNetwork' および 'subnet' の値は ' 短い名前または完全修飾名を使用して指定できます省略形は同じ名前の複数のリソースに一致している可能性があるため、ほとんどの場合は完全修飾名を使用することを推奨します。resourceGroups’netappAccounts’capacityPools' の値は ' 検出されたリソースのセットをこのストレージバックエンドで使用可能なリソースに制限するフィルタであり ' 任意の組み合わせで指定できます完全修飾名の形式は次のとおりです。

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

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

パラメータ	説明	デフォルト
「 exportRule 」	新しいボリュームのエクスポートルール	"0.0.0.0/0 "
「スナップショット方向」	.snapshot ディレクトリの表示を制御します	いいえ
「 size 」	新しいボリュームのデフォルトサイズ	" 100G "

パラメータ	説明	デフォルト
「 unixPermissions 」	新しいボリュームの UNIX 権限（ 8 進数の 4 桁）	"" （プレビュー機能、サブスクリプションでホワイトリスト登録が必要）

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

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

例 **1** ：最小限の構成

これは、バックエンドの絶対的な最小構成です。この構成では、 ANF に委譲されたネットアップアカウント、容量プール、サブネットがすべて検出され、それらのプールまたはサブネットの 1 つに新しいボリュームがランダムに配置されます。

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

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

例 **2** ：容量プールフィルタを使用した特定のサービスレベル設定

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

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

例 **3** ：高度な設定

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

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

例 **4** ：仮想ストレージプールの構成

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

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

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

以下の「 torageClass 」定義は、上記のストレージプールを参照しています。「 parameters.selector` 」フィールドを使用すると、ボリュームのホストに使用される仮想プールを「 S torageClass 」ごとに指定できます。ボリュームには、選択したプールで定義された要素があります。

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

次の手順

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs
```

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

GCP バックエンドの CVS を設定します

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

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

必要なもの

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

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

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

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

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

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

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

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

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

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

「 apiRegion 」は、 Astra Trident が CVS ボリュームを作成する GCP リージョンです。複数リージョンの Kubernetes クラスタを作成する場合、「 apiRegion 」で作成した CVS ボリュームは、複数の GCP リージョンのノードでスケジュールされたワークロードで使用できます。リージョン間トラフィックは追加コストがかかることに注意してください。

- クロスリージョンアクセスを有効にするには、「 allowedTopologies 」の StorageClass 定義にすべてのリージョンを含める必要があります。例：

```
- key: topology.kubernetes.io/region
  values:
  - us-east1
  - europe-west1
```

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

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

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

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

パラメータ	説明	デフォルト
「 exportRule 」	新しいボリュームのエクスポートルール	"0.0.0.0/0 "
「スナップショット方向」	「 .snapshot 」ディレクトリにアクセスします	いいえ

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

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

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

例 **1** ：最小限の構成

これは、バックエンドの絶対的な最小構成です。

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

```
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbO
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKe
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRA
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
        "client_email": "cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com",
        "client_id": "123456789012345678901",
        "auth_uri": "https://accounts.google.com/o/oauth2/auth",
        "token_uri": "https://oauth2.googleapis.com/token",
        "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
        "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com"
    }
}
```

例 **2** ：基本 **CVS** サービスタイプの設定

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

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

```
SaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyA
ZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz
lzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrt
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbO
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKe
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRA
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
        "client_email": "cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com",
        "client_id": "123456789012345678901",
        "auth_uri": "https://accounts.google.com/o/oauth2/auth",
        "token_uri": "https://oauth2.googleapis.com/token",
        "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
        "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com"
    }
}
```

例 **3** ：単一のサービスレベルの設定

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

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

```
chRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlz
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
zZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHi
sIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOgu
SaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyA
ZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz
lzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrt
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbO
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKe
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRA
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
        "client_email": "cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com",
        "client_id": "123456789012345678901",
        "auth_uri": "https://accounts.google.com/o/oauth2/auth",
        "token_uri": "https://oauth2.googleapis.com/token",
        "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
        "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com"
    },
    "proxyURL": "http://proxy-server-hostname/",
    "nfsMountOptions": "vers=3,proto=tcp,timeo=600",
    "limitVolumeSize": "10Ti",
    "serviceLevel": "premium",
    "defaults": {
        "snapshotDir": "true",
        "snapshotReserve": "5",
        "exportRule": "10.0.0.0/24,10.0.1.0/24,10.0.2.100",
        "size": "5Ti"
    }
}
```

例 **4** ：仮想ストレージプールの構成

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

torageClasses 」を示しています。

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

```
{
    "version": 1,
    "storageDriverName": "gcp-cvs",
    "projectNumber": "012345678901",
    "apiRegion": "us-west2",
    "apiKey": {
        "type": "service_account",
        "project_id": "my-gcp-project",
        "private_key_id": "1234567890123456789012345678901234567890",
        "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisI
sAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSa
PIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZN
chRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlz
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
zZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHi
sIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOgu
SaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyA
ZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz
lzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrt
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbO
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKe
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRA
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
        "client_email": "cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com",
        "client_id": "123456789012345678901",
        "auth_uri": "https://accounts.google.com/o/oauth2/auth",
        "token_uri": "https://oauth2.googleapis.com/token",
        "auth_provider_x509_cert_url":
```

```
"https://www.googleapis.com/oauth2/v1/certs",
        "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com"
    },
    "nfsMountOptions": "vers=3,proto=tcp,timeo=600",

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

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

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

```
            }
        },

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

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

次の StorageClass 定義は、上記のストレージプールを参照してください。parameters.selector` フィールドを使用すると ' ボリュームのホストに使用される仮想プールを各 StorageClass に指定できますボリュームには、選択したプールで定義された要素があります。

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

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

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

次の手順

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

バックエンドの作成に失敗した場合は、バックエンドの設定に何か問題があります。次のコマンドを実行する

と、ログを表示して原因を特定できます。

```
tridentctl logs
```

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

NetApp HCI または SolidFire バックエンドを設定します

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

必要なもの

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

知っておくべきこと

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

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

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

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

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

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

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

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

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

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

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

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

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

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

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

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

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

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

次の StorageClass 定義は、上記の仮想ストレージプールを参照してください。parameters.selector` フィールドを使用すると ' 各 StorageClass は ' ボリュームのホストに使用できる仮想プールを呼び出しますボリュームには、選択した仮想プール内で定義された要素があります。

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

ボリューム・タイプ QoS を備えた唯一のゴールド・パフォーマンスを提供するプールです最後の StorageClass （ 'olidfire-cin`) は、シルバーパフォーマンスを提供するストレージプールをすべて呼び出します。Trident が、どの仮想ストレージプールを選択するかを判断し、ストレージ要件を確実に満たすようにします。

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

詳細については、こちらをご覧ください

- "ボリュームアクセスグループ"

バックエンドに ONTAP または Cloud Volumes ONTAP SAN ドライバを設定します

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

- "準備"
- "設定と例"

ユーザ権限

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

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

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

準備

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

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

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

認証

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

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

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

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

クレデンシャルベースの認証を有効にします

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

バックエンド定義の例は次のようになります。

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

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

証明書ベースの認証を有効にします

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

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

一般的なワークフローは次の手順で構成されます。

手順

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

Name （ CN ；共通名）を設定します。

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

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

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

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

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

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

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

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

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

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

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

認証方法を更新するか、クレデンシャルをローテーションして

既存のバックエンドを更新して、別の認証方式を使用したり、クレデンシャルをローテーションしたりすることができます。これはどちらの方法でも機能します。ユーザ名とパスワードを使用するバックエンドは証明書を使用するように更新できますが、証明書を使用するバックエンドはユーザ名とパスワードに基づいて更新できます。これを行うには ' tridentctl backend update を実行するために必要なパラメータを含む更新された backend.json ファイルを使用します

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

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

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

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

igroup を指定します

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

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

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

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

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

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

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

双方向 **CHAP** を使用して接続を認証します

Astra Trident は 'ONTAP-SAN' ドライバと 'ONTAP-SAN-エコノミー ドライバの双方向 CHAP を使用して iSCSI セッションを認証できますこれには ' バックエンド定義で useCHAP オプションを有効にする必要がありますtrue に設定すると 'Astra Trident は SVM のデフォルトイニシエータセキュリティを双方向 CHAP に構成し ' バックエンドファイルからのユーザ名とシークレットを設定します接続の認証には双方向 CHAP を使用することを推奨します。次の設定例を参照してください。

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "backendName": "ontap_san_chap",
    "managementLIF": "192.168.0.135",
    "svm": "ontap_iscsi_svm",
    "useCHAP": true,
    "username": "vsadmin",
    "password": "FaKePaSsWoRd",
    "igroupName": "trident",
    "chapInitiatorSecret": "cl9qxIm36DKyawxy",
    "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
}
```

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

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

動作の仕組み

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

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

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

クレデンシャルをローテーションし、バックエンドを更新

CHAP 証明書を更新するには 'backend.json ファイルの CHAP パラメータを更新しますこれには 'CHAP シー

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

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

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

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

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

設定オプションと例

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

バックエンド設定オプションについては、次の表を参照してください。

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

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

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

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

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

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

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

バックエンドを作成するときは、作成後に「 datalif 」と「 toragePrefix 」を変更できないことに注意してください。これらのパラメータを更新するには、新しいバックエンドを作成する必要があります。

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

バックエンドは、作成後に igroup を更新することもできます。

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

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

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

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

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

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

「 ONTAP-SAN-エコノミー 」ドライバの場合は、「 limitVolumeSize 」オプションによって、 qtree および LUN 用に管理するボリュームの最大サイズも制限されます。

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

ボリュームのプロビジョニング用のバックエンド構成オプション

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

パラメータ	説明	デフォルト
「平和の配分」	space-allocation for LUN のコマンドを指定します	正しいです
「平和のための準備」を参照してください	スペースリザベーションモード：「 none 」（シン）または「 volume 」（シック）	なし
「ナプショットポリシー」	使用する Snapshot ポリシー	なし
「 QOSPolicy 」	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	「」
「 adaptiveQosPolicy 」を参照してください	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	「」
「スナップショット予約」	スナップショット "0" 用に予約されたボリュームの割合	「 napshotPolicy 」が「 none 」の場合、それ以外の場合は「」
'plitOnClone	作成時にクローンを親からスプリットします	いいえ
'plitOnClone	作成時にクローンを親からスプリットします	いいえ
「暗号化」	ネットアップのボリューム暗号化を有効にします	いいえ
'ecurityStyle'	新しいボリュームのセキュリティ形式	「 UNIX 」
階層ポリシー	「なし」を使用する階層化ポリシー	ONTAP 9.5 よりも前の SVM-DR 構成の「スナップショットのみ」

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

次に、デフォルトが定義されている例を示します。

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

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

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

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

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

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

最小限の設定例

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

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

`ontap-san` 証明書ベースの認証を使用するドライバ

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

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

`ontap-san` 双方向**CHAP**を備えたドライバ

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

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

`ontap-san-economy` ドライバ

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

仮想ストレージプールを使用するバックエンドの例

以下に示すバックエンド定義ファイルの例では ' すべてのストレージ・プールに対して特定のデフォルトが設定されていますたとえば 'paceReserve at none`paceAllocation] at false' と 'encryption' は false です仮想ストレージプールは、ストレージセクションで定義します。

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

```
```

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

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

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

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

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

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

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

バックエンドを **StorageClasses** にマッピングします

次の StorageClass 定義は、上記の仮想ストレージプールを参照してください。parameters.selector` フィールドを使用すると ' 各 StorageClass は ' ボリュームのホストに使用できる仮想プールを呼び出しますボリュームには、選択した仮想プール内で定義された要素があります。

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

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

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

バックエンドに **ONTAP NAS** ドライバを設定します

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

- "準備"
- "設定と例"

ユーザ権限

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

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

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

準備

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

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

複数のドライバを実行し、 1 つまたは複数のドライバを参照するストレージクラスを作成することもできます。たとえば 'ONTAP-NAS' ドライバを使用する Gold クラスと 'ONTAP-NAS-エコノミー 'one を使用する Bronze クラスを構成できます

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

認証

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

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

ト証明書、キー、および信頼された CA 証明書（推奨）が含まれている必要があります。

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

クレデンシャルベースの認証を有効にします

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

バックエンド定義の例は次のようになります。

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

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

証明書ベースの認証を有効にします

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

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

一般的なワークフローは次の手順で構成されます。

手順

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

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

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

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

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

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

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

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

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

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

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

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

認証方法を更新するか、クレデンシャルをローテーションして

既存のバックエンドを更新して、別の認証方式を使用したり、クレデンシャルをローテーションしたりすることができます。これはどちらの方法でも機能します。ユーザ名とパスワードを使用するバックエンドは証明書を使用するように更新できますが、証明書を使用するバックエンドはユーザ名とパスワードに基づいて更新できます。これを行うには ' tridentctl backend update を実行するために必要なパラメータを含む更新された backend.json ファイルを使用します

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

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

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

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

NFS エクスポートポリシーを管理します

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

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

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

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

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

エクスポートポリシーを動的に管理

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

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

例

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

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

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

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

- 「 autoExportPolicy 」は「 true 」に設定されています。これは、 Astra Trident が「 vm1 」 SVM のエクスポートポリシーを作成し、「 autoExportCIDRs 」アドレスブロックを使用してルールの追加と削除を処理することを示しています。たとえば、 UUID 403b5326-842-40dB-96d0-d83fb3f4daec と「 autoExportPolicy 」が「 true 」に設定されているバックエンドは、 SVM 上に「 trident-403b5326-842-40dB-96d0-d83f3f4daec 」という名前のエクスポートポリシーを作成します。
- 「 autoExportCI` 」には、アドレスブロックのリストが含まれています。このフィールドは省略可能で、デフォルト値は ["0.0.0.0/0" 、 "::/0" です。定義されていない場合は、 Astra Trident が、ワーカーノードで

検出されたすべてのグローバルにスコープ指定されたユニキャストアドレスを追加します。

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

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

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

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

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

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

以前のバックエンドの場合は 'tridentctl update backend でバックエンドを更新することで 'Astra Trident がエクスポートポリシーを自動的に管理できるようになりますこれにより、バックエンドの UUID のあとにという

名前の新しいエクスポートポリシーが作成され、バックエンドに存在するボリュームは、新しく作成したエクスポートポリシーを使用して、再びマウントします。

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

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

設定オプションと例

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

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

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

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

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

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

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

バックエンドを作成するときは、作成後に「 dataLIF' 」と「 toragePrefix 」を変更できないことに注意してください。これらのパラメータを更新するには、新しいバックエンドを作成する必要があります。

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

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

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

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

「 ONTAP-NAS-エコノミー 」ドライバの場合、「 limitVolumeSize 」オプションを使用すると、 qtree および LUN 用に管理するボリュームの最大サイズも制限されます。「 qtreesPerFlexvol 」オプションを使用すると、 FlexVol あたりの最大 qtree 数をカスタマイズできます。

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

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

ボリュームのプロビジョニング用のバックエンド構成オプション

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

パラメータ	説明	デフォルト
「平和の配分」	space-allocation for LUN のコマンドを指定します	正しいです
「平和のための準備」を参照してください	スペースリザベーションモード：「 none 」（シン）または「 volume 」（シック）	なし
「ナプショットポリシー」	使用する Snapshot ポリシー	なし
「 QOSPolicy 」	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	「」
「 adaptiveQosPolicy 」を参照してください	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します。経済性に影響する ONTAP - NAS ではサポートされません。	「」
「スナップショット予約」	スナップショット "0" 用に予約されたボリュームの割合	「 napshotPolicy 」が「 none 」の場合、それ以外の場合は「」
'plitOnClone	作成時にクローンを親からスプリットします	いいえ
「暗号化」	ネットアップのボリューム暗号化を有効にします	いいえ
'ecurityStyle'	新しいボリュームのセキュリティ形式	「 UNIX 」
階層ポリシー	「なし」を使用する階層化ポリシー	ONTAP 9.5 よりも前の SVM-DR 構成の「スナップショットのみ」
unixPermissions	新しいボリュームのモード	777

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

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

次に、デフォルトが定義されている例を示します。

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

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

は、次の例を参照してください。

計算は次のとおりです。

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

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

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

最小限の設定例

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

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

`ontap-nas` 証明書ベースの認証を使用するドライバ

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

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

`ontap-nas` ドライバと自動エクスポートポリシー

この例は、動的なエクスポートポリシーを使用してエクスポートポリシーを自動的に作成および管理するように Astra Trident に指示する方法を示しています。これは「 ONTAP-NAS-エコノミー 」と「 ONTAP-NAS-flexgroup 」ドライバで同様に機能します。

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

`ontap-nas-flexgroup` ドライバ

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

`ontap-nas` **IPv6**対応ドライバ

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

`ontap-nas-economy` ドライバ

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

仮想ストレージプールを使用するバックエンドの例

以下に示すバックエンド定義ファイルの例では ' すべてのストレージ・プールに対して特定のデフォルトが設定されていますたとえば 'paceReserve at none'`paceAllocation] at false' と 'encryption' は false です仮想ストレージプールは、ストレージセクションで定義します。

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

ontap-nas ドライバ

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

    "defaults": {
          "spaceReserve": "none",
          "encryption": "false",
          "qosPolicy": "standard"
    },
    "labels":{"store":"nas_store", "k8scluster": "prod-cluster-1"},
    "region": "us_east_1",
    "storage": [
        {
            "labels":{"app":"msoffice", "cost":"100"},
            "zone":"us_east_1a",
            "defaults": {
                "spaceReserve": "volume",
                "encryption": "true",
                "unixPermissions": "0755",
                "adaptiveQosPolicy": "adaptive-premium"
            }
        },
        {
            "labels":{"app":"slack", "cost":"75"},
            "zone":"us_east_1b",
            "defaults": {
                "spaceReserve": "none",
                "encryption": "true",
                "unixPermissions": "0755"
            }
        },
        {
            "labels":{"app":"wordpress", "cost":"50"},
            "zone":"us_east_1c",
```

```
            "defaults": {
                "spaceReserve": "none",
                "encryption": "true",
                "unixPermissions": "0775"
            }
        },
        {
            "labels":{"app":"mysqldb", "cost":"25"},
            "zone":"us_east_1d",
            "defaults": {
                "spaceReserve": "volume",
                "encryption": "false",
                "unixPermissions": "0775"
            }
        }
    ]
}
```

`ontap-nas-flexgroup` ドライバ

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

    "defaults": {
          "spaceReserve": "none",
          "encryption": "false"
    },
    "labels":{"store":"flexgroup_store", "k8scluster": "prod-cluster-1"},
    "region": "us_east_1",
    "storage": [
        {
            "labels":{"protection":"gold", "creditpoints":"50000"},
            "zone":"us_east_1a",
            "defaults": {
                "spaceReserve": "volume",
                "encryption": "true",
                "unixPermissions": "0755"
            }
        },
```

```
        {
            "labels":{"protection":"gold", "creditpoints":"30000"},
            "zone":"us_east_1b",
            "defaults": {
                "spaceReserve": "none",
                "encryption": "true",
                "unixPermissions": "0755"
            }
        },
        {
            "labels":{"protection":"silver", "creditpoints":"20000"},
            "zone":"us_east_1c",
            "defaults": {
                "spaceReserve": "none",
                "encryption": "true",
                "unixPermissions": "0775"
            }
        },
        {
            "labels":{"protection":"bronze", "creditpoints":"10000"},
            "zone":"us_east_1d",
            "defaults": {
                "spaceReserve": "volume",
                "encryption": "false",
                "unixPermissions": "0775"
            }
        }
    ]
}
```

ontap-nas-economy ドライバ

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

    "defaults": {
          "spaceReserve": "none",
          "encryption": "false"
    },
```

```
    "labels":{"store":"nas_economy_store"},
    "region": "us_east_1",
    "storage": [
        {
            "labels":{"department":"finance", "creditpoints":"6000"},
            "zone":"us_east_1a",
            "defaults": {
                "spaceReserve": "volume",
                "encryption": "true",
                "unixPermissions": "0755"
            }
        },
        {
            "labels":{"department":"legal", "creditpoints":"5000"},
            "zone":"us_east_1b",
            "defaults": {
                "spaceReserve": "none",
                "encryption": "true",
                "unixPermissions": "0755"
            }
        },
        {
            "labels":{"department":"engineering", "creditpoints":"3000"},
            "zone":"us_east_1c",
            "defaults": {
                "spaceReserve": "none",
                "encryption": "true",
                "unixPermissions": "0775"
            }
        },
        {
            "labels":{"department":"humanresource",
"creditpoints":"2000"},
            "zone":"us_east_1d",
            "defaults": {
                "spaceReserve": "volume",
                "encryption": "false",
                "unixPermissions": "0775"
            }
        }
    ]
}
```

バックエンドを **StorageClasses** にマッピングします

次の StorageClass 定義は、上記の仮想ストレージプールを参照してください。parameters.selector` フィール

ドを使用すると ' 各 StorageClass は ' ボリュームのホストに使用できる仮想プールを呼び出しますボリュームには、選択した仮想プール内で定義された要素があります。

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

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

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

Amazon FSX for NetApp ONTAP で Astra Trident を使用

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

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

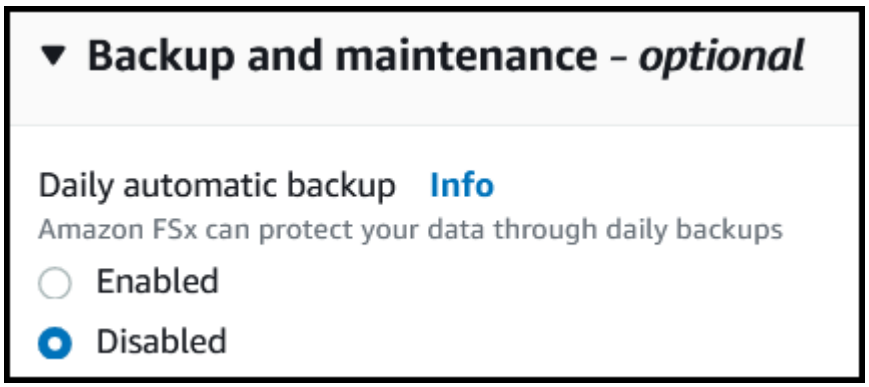
Amazon Elastic Kubernetes Service （ EKS ）で実行されている Astra Trident と Amazon FSX for NetApp ONTAP を使用すると、 ONTAP がサポートするブロックボリュームとファイル永続ボリュームを確実にプロビジョニングできます。

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

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

この問題 を回避するには、次の手順

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



Astra Trident の詳細をご確認ください

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

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

ラクティス"

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

ドライバー機能の詳細をご覧ください "こちらをご覧ください"。

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

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

ドライバ

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

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

認証

Astra Trident には、次の 2 つの認証モードがあります。

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

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

認証の詳細については、次のリンクを参照してください。

- "ONTAP NAS"
- "ONTAP SAN"

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

必要なもの

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

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

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

手順

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

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

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

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

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

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

バックエンドの作成については、次のリンクを参照してください。

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

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

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

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

詳細については、こちらをご覧ください

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

kubectl を使用してバックエンドを作成します

バックエンドは、 Astra Trident とストレージシステムの関係を定義します。Trident がストレージシステムとの通信方法を Trident から指示し、 Astra Trident がボリュームをプロビジョニングする方法も解説します。Astra Trident のインストールが完了したら、次の手順でバックエンドを作成します。TridentBackendConfig カスタムリソース定義（ CRD ）を使用すると、 Trident バックエンドを Kubernetes インターフェイスから直接作成および管理できます。これを行うには 'kubectl' または Kubernetes ディストリビューションに相当する CLI ツールを使用します

`TridentBackendConfig`

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

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

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

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

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

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

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

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

「 PEC 」は、バックエンド固有の設定パラメータをとります。この例では ' バックエンドは 'ONTAP-SAN' ストレージ・ドライバを使用し ' ここで表に示す構成パラメータを使用します使用するストレージドライバの設定オプションの一覧については、を参照してください "ストレージドライバのバックエンド設定情報"。

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

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

が削除されます。これがデフォルト値です。

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

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

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

手順の概要

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

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

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

手順 1 ： Kubernetes Secret を作成します

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

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

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

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

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

手順2：を作成します `TridentBackendConfig` CR

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

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

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

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

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

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

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

フェーズには次のいずれかの値を指定できます。

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

- 「削除」：「 TridentBackendConfig 」 CR の「要素ポリシー」は「削除」に設定されています。「 TridentBackendConfig 」 CR が削除されると、「 Deleting 」状態に移行します。
  - バックエンドに永続ボリューム要求（ PVC ）が存在しない場合、「 TridentBackendConfig 」を削除すると、 Astra Trident はバックエンドと「 TridentBackendConfig 」 CR を削除します。
  - バックエンドに 1 つ以上の PVC が存在する場合は、削除状態になります。次に 'TridentBackendConfig'CR が削除フェーズに入りますバックエンドおよび TridentBackendConfig は、すべての PVC が削除された後にのみ削除されます。
- `lost` ：「 TridentBackendConfig 」 CR に関連付けられているバックエンドが誤って削除されたか、意図的に削除されました。「 TridentBackendConfig 」 CR には削除されたバックエンドへの参照があります。「 TridentBackendConfig 」 CR は、「 $eleetionPolicy 」の値に関係なく削除できます。
- Unknown`: Astra Trident は 'TridentBackendConfig'CR に関連付けられたバックエンドの状態または存在を判断できませんたとえば、 API サーバが応答していない場合や、「 tridentbackends.trident.netapp.io` CRD 」がない場合などです。これには、ユーザの介入が必要な場合があります。

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

（オプション）手順 4 ：詳細を確認します

バックエンドに関する詳細情報を確認するには、次のコマンドを実行します。

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

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

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

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

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

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

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

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

kubectl を使用してバックエンド管理を実行します

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

バックエンドを削除します

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

次のコマンドを実行します。

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

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

既存のバックエンドを表示します

次のコマンドを実行します。

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

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

バックエンドを更新します

バックエンドを更新する理由はいくつかあります。

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

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

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

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

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

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

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

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

tridentctl を使用してバックエンド管理を実行します

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

バックエンドを作成します

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

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

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

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

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

バックエンドを削除します

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

1. バックエンド名を取得します。

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

2. バックエンドを削除します。

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

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

既存のバックエンドを表示します

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

- 概要を取得するには、次のコマンドを実行します。

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

- すべての詳細を確認するには、次のコマンドを実行します。

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

バックエンドを更新します

新しいバックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

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

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

バックエンドを使用するストレージクラスを特定します

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

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

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

バックエンド管理オプション間を移動します

Astra Trident でバックエンドを管理するさまざまな方法をご確認ください。「 TridentBackendConfig 」が導入されたことで、管理者はバックエンドを 2 つの方法で管理できるようになりました。これには、次のような質問があります。

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

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

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

これは、次のシナリオに該当します。

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

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

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

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

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

手順 **0** ：バックエンドを特定します

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

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

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

$ cat ontap-nas-backend.json

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

手順 **0** ：バックエンドを特定します

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ストレージクラスを管理する

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

ストレージクラスを設計する

を参照してください "こちらをご覧ください" ストレージクラスとその設定方法の詳細については、を参照してください。

ストレージクラスを作成する。

ストレージクラスファイルが作成されたら、次のコマンドを実行します。

```
kubectl create -f <storage-class-file>
```

「 <storage-class-file> 」は、ストレージクラスのファイル名に置き換える必要があります。

ストレージクラスを削除する

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

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

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

このストレージクラスで作成された永続ボリュームには変更はなく、 Astra Trident によって引き続き管理されます。

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

既存のストレージクラスを表示します

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

```
kubectl get storageclass
```

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

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

- Astra Trident の同期されたストレージクラスを表示するには、次のコマンドを実行します。

```
tridentctl get storageclass
```

- Astra Trident の同期されたストレージクラスの詳細を表示するには、次のコマンドを実行します。

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

デフォルトのストレージクラスを設定する

Kubernetes 1.6 では、デフォルトのストレージクラスを設定する機能が追加されています。永続ボリューム要求（ PVC ）に永続ボリュームが指定されていない場合に、永続ボリュームのプロビジョニングに使用するストレージクラスです。

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

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

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

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

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

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

クラスタには、常に 1 つのデフォルトストレージクラスだけを設定してください。Kubernetes では、技術的に複数のストレージを使用することはできますが、デフォルトのストレージクラスがまったくない場合と同様に動作します。

ストレージクラスのバックエンドを特定します

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

```
tridentctl get storageclass -o json | jq  '[.items[] | {storageClass:
.Config.name, backends: [.storage]|unique}]'
```

ボリューム操作を実行する

Trident がボリュームを管理するための各種機能をご紹介します。

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

CSI トポロジを使用します

Astra Trident では、を使用して、 Kubernetes クラスタ内にあるノードにボリュームを選択的に作成して接続できます "CSI トポロジ機能"。CSI トポロジ機能を使用すると、領域およびアベイラビリティゾーンに基づいて、ボリュームへのアクセスをノードのサブセットに制限できます。現在、クラウドプロバイダは、Kubernetes 管理者がゾーンベースのノードを生成できるようになっています。ノードは、リージョンによって異なるアベイラビリティゾーンに配置することも、リージョンによって配置することもできます。マルチゾーンアーキテクチャでワークロード用のボリュームをプロビジョニングするために、 Astra Trident は CSI トポロジを使用します。

CSI トポロジ機能の詳細については、を参照してください "こちらをご覧ください"。

Kubernetes には、 2 つの固有のボリュームバインドモードがあります。

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

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

必要なもの

CSI トポロジを使用するには、次のものが必要です。

- 1.17 以降を実行する Kubernetes クラスタ。

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

- クラスタ内のノードには ' トポロジー認識を導入するラベルが必要です（ topology .Kubernetes .io/region および topology .Kubernetes .io/zone ）このラベル * は、 Astra Trident をトポロジ対応としてインストールする前に、クラスタ内のノードに存在する必要があります。

```
$ kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},
{.metadata.labels}]{"\n"}{end}' | grep --color "topology.kubernetes.io"
[node1,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node1","kubernetes.io/
os":"linux","node-
role.kubernetes.io/master":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-a"}]
[node2,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node2","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-b"}]
[node3,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-c"}]
```

手順 **1** ：トポロジ対応バックエンドを作成する

Astra Trident ストレージバックエンドは、アベイラビリティゾーンに基づいてボリュームを選択的にプロビジョニングするように設計できます。各バックエンドは、サポートする必要があるゾーンおよびリージョンのリストを表すオプションの「 upportedTopologies 」ブロックを伝送できます。ストレージクラスがそのようなバックエンドを使用する場合、ボリュームは、サポートされているリージョン / ゾーンでスケジュールされているアプリケーションから要求された場合にのみ作成されます。

バックエンド定義の例を次に示します。

```
{
 "version": 1,
 "storageDriverName": "ontap-san",
 "backendName": "san-backend-us-east1",
 "managementLIF": "192.168.27.5",
 "svm": "iscsi_svm",
 "username": "admin",
 "password": "xxxxxxxxxxxx",
 "supportedTopologies": [
{"topology.kubernetes.io/region": "us-east1",
"topology.kubernetes.io/zone": "us-east1-a"},
{"topology.kubernetes.io/region": "us-east1",
"topology.kubernetes.io/zone": "us-east1-b"}
]
}
```

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

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

```
{"version": 1,
"storageDriverName": "ontap-nas",
"backendName": "nas-backend-us-central1",
"managementLIF": "172.16.238.5",
"svm": "nfs_svm",
"username": "admin",
"password": "Netapp123",
"supportedTopologies": [
      {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-a"},
      {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-b"}
    ]
"storage": [
   {
       "labels": {"workload":"production"},
        "region": "Iowa-DC",
        "zone": "Iowa-DC-A",
        "supportedTopologies": [
            {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-a"}
        ]
    },
    {
        "labels": {"workload":"dev"},
         "region": "Iowa-DC",
         "zone": "Iowa-DC-B",
         "supportedTopologies": [
             {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-b"}
         ]
     }
]
}
```

この例では、「 re Gion 」および「 zone 」ラベルはストレージプールの場所を表しています。「 topology.efix/region' 」と「 topology.uns.io/zone 」は、ストレージプールの消費元を決定します。

手順 **2** ：トポロジを認識するストレージクラスを定義する

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

次の例を参照してください。

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

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

ステップ **3** ： **PVC** を作成して使用する

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

以下の例「 PEC 」を参照してください。

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

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

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

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

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

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

次の出力を参照してください。

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

バックエンドを更新して追加 `supportedTopologies`

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

詳細については、こちらをご覧ください

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

スナップショットを操作します

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

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

必要なもの

- ボリュームSnapshotを作成するには、外部のSnapshotコントローラとカスタムリソース定義（CRD）を作成する必要があります。使用されているKubernetes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、次のように導入できます。

1. ボリュームのSnapshot作成

Kubernetes 1.20以降では、v5.0以上のスナップショットコンポーネントでv1スナップショットCRDを使用します。Kubernetesバージョン1.18と1.19の場合は、v1beta1にv3.0.3のスナップショットコンポーネントを使用します。

v5.0コンポーネント

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

v3.0.3コンポーネント

```
$ cat snapshot-setup.sh
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-
snapshotter/v3.0.3/client/config/crd/snapshot.storage.k8s.io_volumes
napshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-
snapshotter/v3.0.3/client/config/crd/snapshot.storage.k8s.io_volumes
napshotcontents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-
snapshotter/v3.0.3/client/config/crd/snapshot.storage.k8s.io_volumes
napshots.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。

   Kubernetes 1.20以降ではv5.0以上を使用してください。Kubernetesバージョン1.18および1.19では、v3.0.3を使用します

GKE環境でオンデマンドボリュームスナップショットを設定する場合は、スナップショットコントローラを作成しないでください。GKE では、内蔵の非表示のスナップショットコントローラを使用します。

v5.0コントローラ

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-5.0/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-5.0/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
```

v3.0.3コントローラ

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/v3.0.3/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/v3.0.3/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
```

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

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

手順**1**：を設定します `VolumeSnapshotClass`

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

```
$ cat snap-sc.yaml
#Use apiVersion v1 for Kubernetes 1.20 and above. For Kubernetes 1.18 and
1.19, use apiVersion v1beta1.
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

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

手順 **2** ：既存の **PVC** のスナップショットを作成します

```
$ cat snap.yaml
#Use apiVersion v1 for Kubernetes 1.20 and above. For Kubernetes 1.18 and
1.19, use apiVersion v1beta1.
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: pvc1-snap
spec:
  volumeSnapshotClassName: csi-snapclass
  source:
    persistentVolumeClaimName: pvc1
```

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

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

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

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

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

```
$ kubectl describe volumesnapshots pvc1-snap
Name:         pvc1-snap
Namespace:    default
.
.
.
Spec:
  Snapshot Class Name:    pvc1-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-525400f3f660
  Source:
    API Group:
    Kind:       PersistentVolumeClaim
    Name:       pvc1
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
.
.
```

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

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

スナップショットを使用して PVC を作成する例は、次のとおりです。

```
$ cat pvc-from-snap.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

「 dataSource 」は、「 PVC1-SNAP 」という名前のボリューム Snapshot をデータのソースとして使用して PVC を作成する必要があることを示します。このコマンドを実行すると、 Astra Trident が Snapshot から PVC を作成するように指示します。作成された PVC は、ポッドに接続して、他の PVC と同様に使用できます。

> スナップショットが関連付けられている永続ボリュームを削除すると、対応する Trident ボリュームが「削除状態」に更新されます。Astra Trident ボリュームを削除するには、ボリュームの Snapshot を削除する必要があります。

詳細については、こちらをご覧ください

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

ボリュームを展開します

Astra Trident により、 Kubernetes ユーザは作成後にボリュームを拡張できます。ここでは、 iSCSI ボリュームと NFS ボリュームの拡張に必要な設定について説明します。

iSCSI ボリュームを展開します

CSI プロビジョニングを使用して、 iSCSI Persistent Volume （ PV ）を拡張できます。

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

概要

iSCSI PV の拡張には、次の手順が含まれます。

- StorageClass 定義を編集して 'allowVolumeExpansion フィールドを true に設定します
- PVC 定義を編集し 'PEC.resources.requests.storage を更新して ' 新たに必要とされるサイズを反映しますこれは ' 元のサイズより大きくなければなりません
- サイズを変更するには、 PV をポッドに接続する必要があります。iSCSI PV のサイズ変更には、次の 2 つのシナリオがあります。
  - PV がポッドに接続されている場合、 Astra Trident はストレージバックエンドのボリュームを拡張し、デバイスを再スキャンし、ファイルシステムのサイズを変更します。
  - 未接続の PV のサイズを変更しようとすると、 Astra Trident がストレージバックエンドのボリュームを拡張します。PVC がポッドにバインドされると、 Trident はデバイスを再スキャンし、ファイルシステムのサイズを変更します。展開操作が正常に完了すると、 Kubernetes は PVC サイズを更新します。

次の例は、 iSCSI PVS の仕組みを示しています。

手順 **1** ：ボリュームの拡張をサポートするようにストレージクラスを設定する

```
$ cat storageclass-ontapsan.yaml
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

既存の StorageClass の場合は 'allowVolumeExpansion パラメータを含めるように編集します

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

```
$ cat pvc-ontapsan.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Astra Trident が、永続的ボリューム（ PV ）を作成し、この永続的ボリューム要求（ PVC ）に関連付けます。

```
$ kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO            ontap-san      8s

$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi        RWO
Delete           Bound    default/san-pvc    ontap-san               10s
```

手順 **3** ： **PVC** を接続するポッドを定義します

この例では、ポッドが作成され、「 1-pvc 」が使用されます。

```
$  kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
centos-pod   1/1     Running   0          65s

$  kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    centos-pod
```

ステップ **4** ： **PV** を展開します

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

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

手順 **5** ：拡張を検証する

PVC 、 PV 、 Astra Trident のボリュームのサイズを確認することで、拡張が正しく機能しているかどうかを検証できます。

```
$ kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
$ tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

NFS ボリュームを拡張します

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

手順 **1** ：ボリュームの拡張をサポートするようにストレージクラスを設定する

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

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

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

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

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

Astra Trident が、この PVC に対して 20MiB の NFS PV を作成する必要があります。

```
$ kubectl get pvc
NAME           STATUS   VOLUME
CAPACITY     ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi
RWO            ontapnas        9s

$ kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi       RWO
Delete           Bound    default/ontapnas20mb   ontapnas
2m42s
```

ステップ 3 ： PV を展開します

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

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

手順 **4** ：拡張を検証する

PVC 、 PV 、 Astra Trident のボリュームのサイズを確認することで、サイズ変更が正しく機能しているかどうかを検証できます。

```
$ kubectl get pvc ontapnas20mb
NAME           STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi
RWO            ontapnas        4m44s

$ kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi        RWO
Delete           Bound    default/ontapnas20mb   ontapnas
5m35s

$ tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n
trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

ボリュームをインポート

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

ボリュームインポートをサポートするドライバ

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

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

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

ボリュームをインポートする理由

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

- アプリケーションのコンテナ化と既存のデータセットの再利用
- エフェメラルアプリケーション用のデータセットのクローンを使用する
- 障害が発生した Kubernetes クラスタの再構築
- ディザスタリカバリ時にアプリケーションデータを移行する

インポートはどのように機能しますか。

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

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

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

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

ストレージボリュームをインポートするには、ボリュームが含まれている Astra Trident バックエンドの名前と、ストレージ上のボリュームを一意に識別する名前（ ONTAP FlexVol 、 Element Volume 、 CVS ボリュームパスなど）を指定します。ストレージボリュームは、読み取り / 書き込みアクセスを許可し、指定された Astra Trident バックエンドからアクセスできる必要があります。引数 -f 文字列は必須であり、 YAML または JSON PVC ファイルへのパスを指定します。

Astra Trident がインポートボリューム要求を受信すると、既存のボリュームサイズが決定され、 PVC で設定されます。ストレージドライバによってボリュームがインポートされると、 PV は ClaimRef を使用して PVC

に作成されます。再生ポリシーは、最初に PV 内の「そのまま」に設定されます。Kubernetes が PVC と PV を正常にバインドすると、再利用ポリシーがストレージクラスの再利用ポリシーに合わせて更新されます。ストレージクラスの再利用ポリシーが「削除」の場合、 PV を削除するとストレージボリュームは削除されます。

--no-manage 引数を指定してボリュームをインポートすると、 Trident はオブジェクトのライフサイクルについて PVC または PV に対する追加の操作を実行しません。Trident は '--no-managed' オブジェクトの PV イベントと PVC イベントを無視するため 'PV を削除してもストレージ・ボリュームは削除されませんボリュームのクローンやサイズ変更などの他の処理も無視されます。このオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、 Kubernetes 以外でストレージボリュームのライフサイクルを管理する場合に便利です。

PVC と PV にアノテーションが追加されます。このアノテーションは、ボリュームがインポートされたこと、および PVC と PV が管理されていることを示す二重の目的を果たします。このアノテーションは変更または削除しないでください。

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

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

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

> Trident がインポートする ONTAP のタイプは RW である必要があります。DP タイプのボリュームは SnapMirror デスティネーションボリュームです。 Trident にボリュームをインポートする前に、ミラー関係を解除する必要があります。

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

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

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

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

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

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

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

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

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

`ontap-san` インポート

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

- 「 ontap - san 」バックエンドの名前を含めます。

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

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

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

既存のアクティブな接続がないボリュームをインポートすることを推奨します。アクティブに使用されているボリュームをインポートする場合は、最初にボリュームをクローニングしてからインポートを実行します。

例

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

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

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

ONTAP ボリュームのタイプが RW であることが Astra Trident でインポートされる必要があります。DP タイプのボリュームは SnapMirror デスティネーションボリュームです。ボリュームを Astra Trident にインポートする前に、ミラー関係を解除する必要があります。

`element` インポート

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

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

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

Element ドライバではボリューム名の重複がサポートされます。ボリューム名が重複している場合、 Trident のボリュームインポートプロセスはエラーを返します。回避策として、ボリュームをクローニングし、一意のボリューム名を指定します。次に、クローンボリュームをインポートします。

`gcp-cvs` インポート

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

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

```
$ tridentctl import volume gcpcvs_YEppr adroit-jolly-swift -f <path-to-
pvc-file> -n trident

+-----------------------------------------+--------+--------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+--------+--------------
+----------+-----------------------------------------+--------+---------+
| pvc-a46ccab7-44aa-4433-94b1-e47fc8c0fa55 | 93 GiB | gcp-storage   | file
| e1a6e65b-299e-4568-ad05-4f0a105c888f | online | true    |
+-----------------------------------------+--------+--------------
+----------+-----------------------------------------+--------+---------+
```

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

`azure-netapp-files` インポート

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

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

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

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

ワーカーノードを準備します

Kubernetes クラスタ内のすべてのワーカーノードが、ポッド用にプロビジョニングしたボリュームをマウントできる必要があります。いずれかのバックエンドに「 ONTAP-NAS' 」、「 ONTAP-NAS-Bエコノミー 」、または「 ONTAP-NAS-flexgroup 」ドライバを使用している場合は、ワーカーノードに NFS ツールが必要です。それ以外の場合は iSCSI ツールが必要です。

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

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

NFS ボリューム

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

ブート時に NFS サービスが開始されていることを確認してください。

iSCSI ボリューム

iSCSI ボリュームを使用するときは、次の点に注意してください。

- Kubernetes クラスタ内の各ノードには一意の IQN を割り当てる必要があります。* これは必須の前提条件です * 。
- RHCOS バージョン 4.5 以降、または RHEL または CentOS バージョン 8.2 以降で「 olidfire-san 」ドライバを使用している場合は、 CHAP 認証アルゴリズムが /etc/iscsi/iscsid.conf で MD5 に設定されていることを確認してください。

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

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

プロトコル	オペレーティングシステム	コマンド
iSCSI	RHEL/CentOS	1. 次のシステムパッケージをインストールします。 'sudo yum install -y lsscsi iscsi-initiator-utils SG3_utils device-mapper-multipath` 2. iscsi-initiator-utils のバージョンが 6.2.0.874-2.el7 以降であることを確認します。 「 rpm -q iscsi-initiator-utils 」のように入力します 3. スキャンを手動に設定： 'sudo sed-i'/ ^ \ （ node.session.scan\).*/\1 = manual/ '/etc/iscsi/iscsid.conf 4. マルチパスを有効化： 'UDO mpathconf—enable --with _multipathd y — find _multipaths n 「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。 5. 「 iscsid 」と「 multipathd 」が実行されていることを確認します。 'sudo systemctl enable — 現在 iscsid multipathd 6. 'iSCSI' を有効にして開始します 'sudo systemctl enable — 現在の iSCSI

プロトコル	オペレーティングシステム	コマンド
iSCSI	Ubuntu / Debian	1. 次のシステムパッケージをインストールします。 'UDO apt-get install-y open-iscsi lsscsi SG3-utils multipath-tools scsitools' 2. open-iscsi バージョンが 2.0.874-5ubuntu2.10 以降（ bionic の場合）または 2.0.874-7.1ubuntu6.1 以降（ Focal の場合）であることを確認します。 `d pkg -l open-iscsi` 3. スキャンを手動に設定： 'sudo sed-i'/ ^ \ （ node.session.scan\).*/\1 = manual/ '/etc/iscsi/iscsid.conf 4. マルチパスを有効化： 'sudo tee//etc/multipath.conf ←' EOF ' defaults ｛ user_friendy_names yes find _multipaths no ｝ EOF sudo systemctl enable — 今では multipath-tools.service sudo service multipath-tools restart' 「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。 5. 「 open-iSCSI」 および「マルチパスツール」が有効で実行されていることを確認します。 'sudo systemctl status multipath-tools `sudo systemctl enable` －現在の `open-iscsi.service` 'udo systemctl status open-iscsi'

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

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

ワーカーノードの自動準備

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

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

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

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

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

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

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

Astra Trident を監視

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

Astra Trident が提供する指標を使用すると、次のことが可能になります。

- Astra Trident の健常性と設定を保持処理が成功した方法と、想定どおりにバックエンドと通信できるかどうかを調べることができます。
- バックエンドの使用状況の情報を調べて、バックエンドでプロビジョニングされているボリュームの数や消費されているスペースなどを確認します。
- 利用可能なバックエンドにプロビジョニングされたボリュームの量のマッピングを維持します。
- パフォーマンスを追跡する。Astra Trident がバックエンドと通信して処理を実行するのにどれくらいの時間がかかるかを調べることができます。

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

必要なもの

- Astra Trident がインストールされた Kubernetes クラスタ
- Prometheus インスタンス。これは a である場合もある "コンテナ化された Prometheus 環境" または、Prometheus をとして実行することもできます "ネイティブアプリケーション"。

手順 1 ： Prometheus ターゲットを定義する

Prometheus ターゲットを定義して指標を収集し、 Astra Trident が管理するバックエンド、作成するボリュームなどの情報を取得する必要があります。これ "ブログ" Prometheus と Grafana を Astra Trident とともに使用して指標を取得する方法について説明します。ブログでは、 Kubernetes クラスタでオペレータとして Prometheus を実行する方法と、 Astra Trident のメトリックを取得する ServiceMonitor を作成する方法について説明しています。

手順 2 ： Prometheus ServiceMonitor を作成します

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

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

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

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

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

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

次に、 PrompQL クエリーのいくつかを示します。

Trident の健常性情報を取得

- **Astra Trident** からの **HTTP 2XX** 応答の割合

```
(sum (trident_rest_ops_seconds_total_count{status_code=~"2.."} OR on()
vector(0)) / sum (trident_rest_ops_seconds_total_count)) * 100
```

- **Astra Trident** からのステータスコードによる **REST** 応答の割合

```
(sum (trident_rest_ops_seconds_total_count) by (status_code)  / scalar
(sum (trident_rest_ops_seconds_total_count))) * 100
```

- **Astra Trident** によって実行された処理の平均時間（ミリ秒）

```
sum by (operation)
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by
(operation)
(trident_operation_duration_milliseconds_count{success="true"})
```

Astra Trident の使用状況に関する情報を入手

- 平均体積サイズ

```
trident_volume_allocated_bytes/trident_volume_count
```

- 各バックエンドによってプロビジョニングされた合計ボリューム容量

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

個々のボリュームの使用状況を取得する

これは、 kubelet 指標も収集された場合にのみ有効になります。

- 各ボリュームの使用済みスペースの割合

```
kubelet_volume_stats_used_bytes / kubelet_volume_stats_capacity_bytes *
100
```

Astra Trident AutoSupport の計測データ

デフォルトでは、 Astra Trident は Prometheus 指標と基本バックエンド情報を毎日定期的にネットアップに送信します。

- Astra Trident が Prometheus 指標と基本バックエンド情報をネットアップに送信しないようにするには、 Astra Trident のインストール時に「 --silence -autosupport 」フラグを渡します。
- Astra Trident は 'tridentctl send AutoSupport を介してコンテナ・ログをオンデマンドでネットアップ・サポートに送信することもできますAstra Trident をトリガーしてログをアップロードする必要があります。ログを送信する前に、ネットアップのに同意する必要がありますhttps://www.netapp.com/company/legal/privacy-policy/["プライバシーポリシー"^]。
- 指定しないと、 Astra Trident は過去 24 時間からログを取得します。
- ログ保持期間は '--since ' フラグで指定できますたとえば 'tridentctl は AutoSupport --since =1h` を送信しますこの情報は、 Astra Trident と一緒にインストールされた 'trident-autosupport' コンテナを介して収集および送信されます。コンテナイメージは、で取得できます "Trident AutoSupport の略"。
- Trident AutoSupport は、個人情報（ PII ）や個人情報を収集または送信しません。それにはが付いていま

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

Astra Trident から送信されるペイロードの例を次に示します。

```
{
  "items": [
    {
      "backendUUID": "ff3852e1-18a5-4df4-b2d3-f59f829627ed",
      "protocol": "file",
      "config": {
        "version": 1,
        "storageDriverName": "ontap-nas",
        "debug": false,
        "debugTraceFlags": null,
        "disableDelete": false,
        "serialNumbers": [
          "nwkvzfanek_SN"
        ],
        "limitVolumeSize": ""
      },
      "state": "online",
      "online": true
    }
  ]
}
```

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

Astra Trident の指標を無効化

- メトリックがレポートされないようにするには '--generate-custom-yaml' フラグを使用してカスタム YAML を生成し ' これらを編集して 'trident-main' コンテナに対して --metrics フラグが呼び出されないようにします

著作権に関する情報

商標に関する情報