



# Astra Trident を使用

## Astra Trident

NetApp  
June 28, 2024

# 目次

Astra Trident を使用 .....	1
ワーカーノードを準備します .....	1
バックエンドの構成と管理 .....	7
ストレージクラスの作成と管理 .....	134
ボリュームのプロビジョニングと管理 .....	139

# Astra Trident を使用

## ワーカーノードを準備します

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

### 適切なツールを選択する

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

#### NFS ツール

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

#### iSCSI ツール

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

#### NVMe ツール

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



NVMe/TCP には ONTAP 9.12 以降を推奨します。

## ノードサービスの検出

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



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

### イベントを確認します

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

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

### 検出されたサービスを確認

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

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

## NFS ボリューム

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

### RHEL 8以降

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

### Ubuntu

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



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

## iSCSI ボリューム

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

### iSCSIの自己回復機能

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

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



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

Astra Tridentの自動修復機能は、次のような問題を防止します。

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



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

- iSCSIセッションがありません
- LUNが見つかりません

## iSCSIツールをインストール

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

作業を開始する前に

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

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

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

## RHEL 8以降

1. 次のシステムパッケージをインストールします。

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

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

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

3. スキャンを手動に設定：

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

4. マルチパスを有効化：

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



確認します `etc/multipath.conf` が含まれます `find_multipaths no` の下 defaults。

5. を確認します `iscsid` および `multipathd` 実行中：

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

6. を有効にして開始します `iscsi`：

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

## Ubuntu

1. 次のシステムパッケージをインストールします。

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

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

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

### 3. スキャンを手動に設定：

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

### 4. マルチパスを有効化：

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



確認します `etc/multipath.conf` が含まれます `find_multipaths no` の下 `defaults`。

### 5. を確認します `open-iscsi` および `multipath-tools` 有効になっていて実行中：

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



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

## iSCSI自己回復の設定または無効化

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

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



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

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

### Helm

iSCSIの自己修復設定を構成または変更するには、`iscsiSelfHealingInterval` および `iscsiSelfHealingWaitTime` helmのインストール中またはhelmの更新中のパラメータ。

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

```
helm install trident trident-operator-100.2402.0.tgz --set
iscsiSelfHealingInterval=3m0s --set iscsiSelfHealingWaitTime=6m0s -n
trident
```

### Tridentctl

iSCSIの自己修復設定を構成または変更するには、`iscsi-self-healing-interval` および `iscsi-self-healing-wait-time` tridentctlのインストールまたは更新中のパラメータ。

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

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

## NVMe/TCPホリユウム

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



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



## RHEL 9

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

## Ubuntu

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

インストールを確認します

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

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



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

## バックエンドの構成と管理

### バックエンドを設定

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

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

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

## Azure NetApp Files の特長

### Azure NetApp Files バックエンドを設定します

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

### Azure NetApp Filesドライバの詳細

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

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

### 考慮事項

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

### AKSの管理対象ID

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

- AKSを使用して導入されるKubernetesクラスタ
- AKS Kubernetesクラスタに設定された管理対象ID
- Astra Tridentをインストール (以下を含む) `cloudProvider` 指定するには "Azure"。

## Trident オペレーター

Tridentオペレーターを使用してAstra Tridentをインストールするには、`tridentorchestrator_cr.yaml` をクリックして設定します `cloudProvider` 終了: "Azure"。例:

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

## Helm

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

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

## `tridentctl`

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

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

## AKSのクラウドID

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

AzureでクラウドIDを活用するには、以下が必要です。

- AKSを使用して導入されるKubernetesクラスター
- AKS Kubernetesクラスターに設定されたワークロードIDとoidc-issuer
- Astra Tridentをインストール (以下を含む) `cloudProvider` 指定するには "Azure" および `cloudIdentity` ワークロードIDの指定

## Trident オペレータ

Tridentオペレータを使用してAstra Tridentをインストールするには、`tridentorchestrator_cr.yaml` をクリックして設定します `cloudProvider` 終了: "Azure" をクリックして設定します `cloudIdentity` 終了: `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`。

例:

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

## Helm

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

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

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

```
helm install trident trident-operator-100.2402.0.tgz --set
cloudProvider=$CP --set cloudIdentity=$CI
```

## <code>tridentctl</code>

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

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

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

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

## Azure NetApp Files バックエンドを設定する準備をします

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

### NFSボリュームとSMBボリュームの前提条件

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

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



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

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

```

{
  "id": "/subscriptions/<subscription-
id>/providers/Microsoft.Authorization/roleDefinitions/<role-
definition-id>",
  "properties": {
    "roleName": "custom-role-with-limited-perms",
    "description": "custom role providing limited
permissions",
    "assignableScopes": [
      "/subscriptions/<subscription-id>"
    ],
    "permissions": [
      {
        "actions": [
          "Microsoft.NetApp/netAppAccounts/capacityPools/read",
          "Microsoft.NetApp/netAppAccounts/capacityPools/write",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/read",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/write",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/delete",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/
read",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/
write",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/
delete",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/MountTarge
ts/read",
          "Microsoft.Network/virtualNetworks/read",
          "Microsoft.Network/virtualNetworks/subnets/read",
          "Microsoft.Features/featureProviders/subscriptionFeatureRegistrat
ions/read",

```

```

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

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

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

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

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

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

```

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

#### SMBボリュームに関するその他の要件

SMBボリュームを作成するには、以下が必要です。

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

```

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

```

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

## Azure NetApp Files バックエンド構成のオプションと例

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

### バックエンド構成オプション

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



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

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

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



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

パラメータ	説明	デフォルト
limitVolumeSize	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します	"" (デフォルトでは適用されません)
debugTraceFlags	トラブルシューティング時に使用するデバッグフラグ。例： \{"api": false, "method": true, "discovery": true}。 トラブルシューティングを行って詳細なログダンプが必要な場合を除き、このオプションは使用しないでください。	null
nasType	NFSボリュームまたはSMBボリュームの作成を設定  オプションはです nfs、 smb または null。nullに設定すると、デフォルトでNFSボリュームが使用されます。	nfs



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

## 必要な権限とリソース

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

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

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

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

## ボリュームのプロビジョニング

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

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

### 構成例

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

## 最小限の構成

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

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

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

## AKSの管理対象ID

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

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

## AKSのクラウドID

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

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

## 容量プールフィルタを使用した特定のサービスレベル構成

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

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

## 高度な設定

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

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

## 仮想プール構成

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

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

## ストレージクラスの定義

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

を使用した定義の例 `parameter.selector` フィールド

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

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

### SMBボリュームの定義例

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



## デフォルト名前スペースの基本設定

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

## 名前スペースごとに異なるシークレットを使用する

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

## ボリュームごとに異なるシークレットを使用する

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



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

バックエンドを作成します

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs
```

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

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

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

### Google Cloudドライバの詳細

Astra Tridentの特長 `gcp-cvs` クラスタと通信するドライバ。サポートされているアクセスモードは、`ReadWriteOnce(RWO)`、`ReadOnlyMany(ROX)`、`ReadWriteMany(RWX)`、`ReadWriteOncePod(RWOP)`です。

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

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

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

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

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

- standardsw
- zoneredundantstandardsw

#### 必要なもの

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

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

#### バックエンド構成オプション

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

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

パラメータ	説明	デフォルト
apiRegion	<p>Astra TridentがCloud Volumes Service ボリュームを作成するGoogle Cloudリージョン。複数リージョンのKubernetesクラスタを作成する場合は、に作成されたボリューム apiRegion 複数のGoogle Cloudリージョンのノードでスケジュールされたワークロードで使用できます。</p> <p>リージョン間トラフィックは追加コストを発生させます。</p>	
apiKey	<p>を使用したGoogle CloudサービスアカウントのAPIキー netappcloudvolumes.admin ロール。</p> <p>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）。</p>	
proxyURL	<p>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。</p> <p>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。</p> <p>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。</p>	
nfsMountOptions	NFS マウントオプションのきめ細かな制御。	"nfsvers=3 "
limitVolumeSize	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します。	""（デフォルトでは適用されません）
serviceLevel	<p>新しいボリュームのCVS -パフォーマンスレベルまたはCVSサービスレベル。</p> <p>CVS -パフォーマンスの値はです standard、 premium`または `extreme。</p> <p>CVSの値はです standardsw または zoneredundantstandardsw。</p>	<p>CVS -パフォーマンスのデフォルトは「Standard」です。</p> <p>CVSのデフォルトは"standardsw"です。</p>
network	Cloud Volumes Service ボリュームに使用するGoogle Cloudネットワーク。	デフォルト
debugTraceFlags	<p>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： <code>\{"api":false, "method":true}</code>。</p> <p>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。</p>	null

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

## ボリュームのプロビジョニングオプション

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

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

## CVS -パフォーマンスサービスの種類の例

次の例は、CVS -パフォーマンスサービスタイプの設定例を示しています。



```
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
```





```
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
```



```

znHczZsrtrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
XsYg6gyxy4zq7OlwWgLwGa==
-----END PRIVATE KEY-----
client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
client_id: '123456789012345678901'
auth_uri: https://accounts.google.com/o/oauth2/auth
token_uri: https://oauth2.googleapis.com/token
auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
nfsMountOptions: vers=3,proto=tcp,timeo=600
defaults:
  snapshotReserve: '5'
  exportRule: 0.0.0.0/0
labels:
  cloud: gcp
  region: us-west2
storage:
- labels:
  performance: extreme
  protection: extra
  serviceLevel: extreme
  defaults:
  snapshotDir: 'true'
  snapshotReserve: '10'
  exportRule: 10.0.0.0/24
- labels:
  performance: extreme
  protection: standard
  serviceLevel: extreme
- labels:
  performance: premium
  protection: extra
  serviceLevel: premium
  defaults:
  snapshotDir: 'true'
  snapshotReserve: '10'
- labels:
  performance: premium
  protection: standard
  serviceLevel: premium
- labels:
  performance: standard

```

```
serviceLevel: standard
```

#### ストレージクラスの定義

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

## ストレージクラスの例

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

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

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

### CVSサービスタイプの例

次の例は、CVSサービスタイプの設定例を示しています。



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



## 例2：ストレージプールの構成

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

```
---
version: 1
storageDriverName: gcp-cvs
backendName: gcp-std-so-with-pool
projectNumber: '531265380079'
apiRegion: europe-west1
apiKey:
  type: service_account
  project_id: cloud-native-data
  private_key_id: "<id_value>"
  private_key: |-
    -----BEGIN PRIVATE KEY-----
    MIIEvAIBADANBgkqhkiG9w0BAQEFAASCbKjYwggSiAgEAAoIBAQDaT+Oui9FBAw19
    L1AGEkrYU5xd9K5NlO5jMkIFND5wCD+Nv+jd1GvtFRLaLK5RvXyF5wzvztmODNS+
    qtScpQ+5cFpQkuGtv9U9+N6qtuVYYO3b504Kp5CtqVPJCgMJaK2j8pZTIqUiMum/
    5/Y9oTbZrjAHSmGjM2nHzFq2X0rqVmaHghI6ATm4DOuWx8XGWKTGIPlc0qPqJlqS
    LLaWOH4VIZQZCAyW5IU9CAMwqHgdG0uhFNfCgMmED6PBUvVLsLvcq86X+QSWR9k
    ETqElj/sGCenPF7tilDhGBFafd9hPnxg9PZY29ArEZwY9G/ZjZQX7WPgs0VvxiNR
    DxZRC3GXAgMBAECggEACn5c59bG/qnVEVI1CwMAalM5M2z09JFh1L11jKwntNPj
    Vilw2eTW2+UE7HbJru/S7KQgA5Dnn9kvCraEahPRuddUMrD0vG4kTl/IODV6uFuk
    Y0sZfbqd4jMUQ21smvGsqFzwloYWS5qz01W83ivXH/HW/iqkmY2eW+EPRS/hwSSu
    SscR+SojI7PB0BWSJhlV4yqYf3vcD/D95e12CVHfRCkL85DKumeZ+yHENpiXGZAE
    t8xSs4a500Pm6NHhevCw2a/UQ95/foXNUR450HtbjieJo5o+FF6EYZQGfU2ZHZO8
    37FBKuaJkdGW5xqaI9TL7aqkGkFMF4F2qvOZM+vy8QKBgQD4oVuOkJD1hkTHP86W
    esFlw1kpWyJR9ZA7LI0g/rVpslnX+XdDq0WQf4umdLNau5hYEH9LU6ZSGs1Xk3/B
    NHwR6OXFuqEKNiu83d0zSlHhTy7PZpOZdj5a/vVvQfPDMz7OvsqLRd7YCAbdzuQ0
    +Ahq0Ztwvg0HQ64hdW0ukpYRRwKBgQDgyHj98oqswoYuIa+pPlyS0pPwLmjwKyNm
    /HayzCp+Qjiiyy7Tzg8AUq1H1Ou83XbV428jvg7kDhO7PCKfQ+mMmfqHmTpb0Maq
    KpKnZg4ipsqPlyHNNEoRmcailXbwIhCLewMqMrggUiLOmCw4PscL5nK+4GKu2XE1
    jLqjWAZFMQKBgFhkQ9XXRAJ1kR3XpGHoGN890pZOkCVSrqju6aUef/5KY1Fct8ew
    F/+aIxM2iQsvmWQYOvVCnhuY/F2GfAQ7d0om3decuwIOCX/xy7PjHMkLXa2uaZs4
    WR17sLduj62RqXRLX0c0QkwBiNFyHbRcpdkZJQujbYMhBa+7j7SxT4BtAoGAWMWT
    UucocRXZm/pdvz9wteNH3YDwnJLMxm1KC06qMXbBoYrliY4sm3ywJWMC+iCd/H8A
    Gecxd/xVu5mA2L2N3KMq18Zh8Th0G5DwKyDRJgOQ0Q46yuNXOoYEjlo4Wjy8Me
    +t1Q8iK98E0UmZnhTgfSpSNElzb2AqnzQ3MN9uECgYAqdvvdVPnKGFvdtZ2DjyMoJ
    E89UIC41WjjJGmHsd8W65+3X0RwMzKMT6aZc5tK9J5dHvmWIEtNbM+lTImdBFFga
    NWOC6f3r2xbGXHhaWSl+nobpTuvlo56ZRJVvVk7lFMsidzMuHH8pxfgNJemwA4P
    ThDHCEjv035NNV6Kyo00tA==
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@cloud-native-
  data.iam.gserviceaccount.com
  client_id: '107071413297115343396'
```

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

## 次の手順

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs
```

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

## NetApp HCI または SolidFire バックエンドを設定します

Astra Tridentインストール環境でElementバックエンドを作成して使用方法をご確認ください。

### Elementドライバの詳細

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

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

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

作業を開始する前に

Elementバックエンドを作成する前に、次の情報が必要になります。

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

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

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



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

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

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

```

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

```

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

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

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

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

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

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

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

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

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

詳細については、こちらをご覧ください

- ["ボリュームアクセスグループ"](#)

## ONTAP SANドライバ

### ONTAP SANドライバの概要

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

### ONTAP SANドライバの詳細

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



保護、リカバリ、モビリティにAstra Controlを使用している場合は、[Astra Controlドライバの互換性](#)。

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



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

### Astra Controlドライバの互換性

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



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

### ユーザ権限

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



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

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

#### NVMe/TCPに関するその他の考慮事項

Astra Tridentでは、`ontap-san` 以下を含むドライバー：

- IPv6
- NVMeボリュームのSnapshotとクローン
- NVMeボリュームのサイズ変更
- Astra Tridentでライフサイクルを管理できるように、Astra Tridentの外部で作成されたNVMeボリュームをインポートする
- NVMeネイティブマルチパス
- Kubernetesノードのグレースフルシャットダウンまたはグレースフルシャットダウン (24.02)

Astra Tridentでは次の機能がサポートされません。

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

バックエンドに**ONTAP SAN**ドライバを設定する準備をします

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

#### 要件

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

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

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

#### ONTAPバックエンドの認証

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

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

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



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

### クレデンシャルベースの認証を有効にします

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

バックエンド定義の例は次のようになります。

#### YAML

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

#### JSON

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

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

証明書ベースの認証を有効にします

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

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

一般的なワークフローは次の手順で構成されます。

手順

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

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

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

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

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

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

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

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

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

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

```

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

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

```

## 認証方法を更新するか、クレデンシャルをローテーションして

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

```

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

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

```



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

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

双方向 **CHAP** を使用して接続を認証します

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

```
---
version: 1
storageDriverName: ontap-san
backendName: ontap_san_chap
managementLIF: 192.168.0.135
svm: ontap_iscsi_svm
useCHAP: true
username: vsadmin
password: password
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSD6cNwxyz
```



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

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

### 動作の仕組み

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

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

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

### クレデンシャルをローテーションし、バックエンドを更新

CHAPクレデンシャルを更新するには、でCHAPパラメータを更新します backend.json ファイル。CHAPシークレットを更新し、を使用する必要があります tridentctl update 変更を反映するためのコマンドです。





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

```
cat backend-san.json
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "ontap_san_chap",
  "managementLIF": "192.168.0.135",
  "svm": "ontap_iscsi_svm",
  "useCHAP": true,
  "username": "vsadmin",
  "password": "password",
  "chapInitiatorSecret": "cl9qxUpDaTeD",
  "chapTargetInitiatorSecret": "rqxigXgkeUpDaTeD",
  "chapTargetUsername": "iJF4heBRT0TCwxyz",
  "chapUsername": "uh2aNCLsd6cNwxyz",
}

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

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

## ONTAPのSAN構成オプションと例

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
version		常に 1
storageDriverName	ストレージドライバの名前	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、ontap-san、ontap-san-economy
backendName	カスタム名またはストレージバックエンド	ドライバ名+"_"+dataLIF
managementLIF	<p>クラスタ管理LIFまたはSVM管理LIFのIPアドレス。</p> <p>Fully Qualified Domain Name (FQDN ; 完全修飾ドメイン名) を指定できます。</p> <p>IPv6フラグを使用してAstra Tridentをインストールした場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、次のように角かっこで定義する必要があります。</p> <p>[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。</p> <p>シームレスなMetroClusterスイッチオーバーについては、を参照してください。 <a href="#">MetroClusterの例</a>。</p>	「 10.0.0.1 」、 「 [2001:1234:abcd::fefe] 」
dataLIF	<p>プロトコル LIF の IP アドレス。</p> <p>* iSCSIには指定しないでください。* Astra Tridentが使用します <a href="#">"ONTAP の選択的LUNマップ"</a> iSCSI LIFを検出するには、マルチパスセッションを確立する必要があります。の場合は警告が生成されます dataLIF は明示的に定義されます。</p> <p>* MetroClusterの場合は省略してください。* <a href="#">MetroClusterの例</a>。</p>	SVMの派生物です
svm	<p>使用する Storage Virtual Machine</p> <p>* MetroClusterの場合は省略してください。* <a href="#">MetroClusterの例</a>。</p>	SVMの場合に生成されます managementLIF を指定します
useCHAP	<p>CHAPを使用してONTAP SANドライバのiSCSIを認証します (ブーリアン) 。</p> <p>をに設定します true Astra Tridentでは、バックエンドで指定されたSVMのデフォルト認証として双方向CHAPを設定して使用します。を参照してください <a href="#">"バックエンドにONTAP SANドライバを設定する準備をします"</a> を参照してください。</p>	false
chapInitiatorSecret	CHAP イニシエータシークレット。の場合は必須です useCHAP=true	""
labels	ボリュームに適用する任意の JSON 形式のラベルのセット	""

パラメータ	説明	デフォルト
chapTargetInitiatorSecret	CHAP ターゲットイニシエータシークレット。の場合には必須です useCHAP=true	""
chapUsername	インバウンドユーザ名。の場合には必須です useCHAP=true	""
chapTargetUsername	ターゲットユーザ名。の場合には必須です useCHAP=true	""
clientCertificate	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""
clientPrivateKey	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""
trustedCACertificate	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます。	""
username	ONTAP クラスタとの通信に必要なユーザ名。クレデンシャルベースの認証に使用されます。	""
password	ONTAP クラスタとの通信にパスワードが必要です。クレデンシャルベースの認証に使用されます。	""
svm	使用する Storage Virtual Machine	SVMの場合に生成されます managementLIF を指定します
storagePrefix	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。  あとから変更することはできません。このパラメータを更新するには、新しいバックエンドを作成する必要があります。	trident
limitAggregateUsage	使用率がこの割合を超えている場合は、プロビジョニングが失敗します。  NetApp ONTAP バックエンドにAmazon FSXを使用している場合は、指定しないでください limitAggregateUsage。提供された fsxadmin および vsadmin アグリゲートの使用状況を取得し、Astra Tridentを使用して制限するために必要な権限が含まれていない。	"" (デフォルトでは適用されません)
limitVolumeSize	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。  また、qtreeおよびLUNに対して管理するボリュームの最大サイズを制限します。	"" (デフォルトでは適用されません)
lunsPerFlexvol	FlexVol あたりの最大 LUN 数。有効な範囲は 50、200 です	100

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

#### ボリュームのプロビジョニング用のバックエンド構成オプション

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

パラメータ	説明	デフォルト
spaceAllocation	space-allocation for LUN のコマンドを指定します	"正しい"
spaceReserve	スペースリザーベーションモード：「none」（シン）または「volume」（シック）	"なし"
snapshotPolicy	使用する Snapshot ポリシー	"なし"

パラメータ	説明	デフォルト
qosPolicy	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプール/バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します。  Trident が Astra で QoS ポリシーグループを使用するには、ONTAP 9.8 以降が必要です。非共有の QoS ポリシーグループを使用して、各コンスチチュエントに個別にポリシーグループを適用することを推奨します。共有 QoS ポリシーグループにより、すべてのワークロードの合計スループットに対して上限が適用されます。	""
adaptiveQosPolicy	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプール/バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	""
snapshotReserve	Snapshot 用にリザーブされているボリュームの割合	次の場合は「0」 snapshotPolicy は「none」、 それ以外の場合は「」です。
splitOnClone	作成時にクローンを親からスプリットします	いいえ
encryption	新しいボリュームで NetApp Volume Encryption (NVE) を有効にします。デフォルトは false。このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。  NAE がバックエンドで有効になっている場合は、Astra Trident でプロビジョニングされたすべてのボリュームが NAE に有効になります。  詳細については、以下を参照してください。" <a href="#">Astra Trident と NVE および NAE の相互運用性</a> "。	いいえ
luksEncryption	LUKS 暗号化を有効にします。を参照してください " <a href="#">Linux Unified Key Setup (LUKS; 統合キーセットアップ)</a> を使用"。  LUKS 暗号化は NVMe/TCP ではサポートされません。	""
securityStyle	新しいボリュームのセキュリティ形式	unix
tieringPolicy	「none」を使用する階層化ポリシー	ONTAP 9.5 より前の SVM-DR 設定の場合は「snapshot-only」

## ボリュームプロビジョニングの例

デフォルトが定義されている例を次に示します。

```

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

```



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

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

```

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

```

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

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

3 entries were displayed.

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

#### 最小限の設定例

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



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

#### ONTAP SANの例

これは、ontap-san ドライバ。

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

#### ONTAP SANの経済性の例

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

## MetroClusterの例

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

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

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

## 証明書ベースの認証の例

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

```
---
version: 1
storageDriverName: ontap-san
backendName: DefaultSANBackend
managementLIF: 10.0.0.1
svm: svm_iscsi
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSD6cNwxyz
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
```



## 双方向CHAPの例

次の例では、 useCHAP をに設定します true。

### ONTAP SAN CHAPの例

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
labels:
  k8scluster: test-cluster-1
  backend: testcluster1-sanbackend
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSD6cNwxyz
username: vsadmin
password: <password>
```

### ONTAP SANエコノミーCHAPの例

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSD6cNwxyz
username: vsadmin
password: <password>
```

## NVMe/TCPの例

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

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

### 仮想プールを使用するバックエンドの例

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

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

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



```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSD6cNwxyz
username: vsadmin
password: <password>
defaults:
  spaceAllocation: 'false'
  encryption: 'false'
  qosPolicy: standard
labels:
  store: san_store
  kubernetes-cluster: prod-cluster-1
region: us_east_1
storage:
- labels:
  protection: gold
  creditpoints: '40000'
  zone: us_east_1a
  defaults:
    spaceAllocation: 'true'
    encryption: 'true'
    adaptiveQosPolicy: adaptive-extreme
- labels:
  protection: silver
  creditpoints: '20000'
  zone: us_east_1b
  defaults:
    spaceAllocation: 'false'
    encryption: 'true'
    qosPolicy: premium
- labels:
  protection: bronze
  creditpoints: '5000'
  zone: us_east_1c
  defaults:
    spaceAllocation: 'true'
    encryption: 'false'
```

## ONTAP SANの経済性の例

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSD6cNwxyz
username: vsadmin
password: <password>
defaults:
  spaceAllocation: 'false'
  encryption: 'false'
labels:
  store: san_economy_store
region: us_east_1
storage:
- labels:
  app: oracledb
  cost: '30'
  zone: us_east_1a
  defaults:
    spaceAllocation: 'true'
    encryption: 'true'
- labels:
  app: postgresdb
  cost: '20'
  zone: us_east_1b
  defaults:
    spaceAllocation: 'false'
    encryption: 'true'
- labels:
  app: mysqldb
  cost: '10'
  zone: us_east_1c
  defaults:
    spaceAllocation: 'true'
    encryption: 'false'
- labels:
  department: legal
  creditpoints: '5000'
```

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

## NVMe/TCPの例

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

バックエンドを **StorageClasses** にマッピングします

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## ONTAP NAS ドライバ

### ONTAP NAS ドライバの概要

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



## ONTAP NASドライバの詳細

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



保護、リカバリ、モビリティにAstra Controlを使用している場合は、[Astra Controlドライバの互換性](#)。

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

## Astra Controlドライバの互換性

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



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

## ユーザ権限

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

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



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

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

**ONTAP NAS**ドライバを使用してバックエンドを設定する準備をします

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

要件

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

**ONTAP**バックエンドの認証

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

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

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



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

クレデンシャルベースの認証を有効にします

Trident が ONTAP バックエンドと通信するには、SVM を対象とした管理者またはクラスタを対象とした管理者のクレデンシャルが必要です。などの標準の事前定義されたロールを使用することを推奨します `admin` または `vsadmin`。これにより、今後のリリースの ONTAP との互換性が今後のリリースの Astra Trident で使用

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

バックエンド定義の例は次のようになります。

#### YAML

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

#### JSON

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

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

証明書ベースの認証を有効にします

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

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

一般的なワークフローは次の手順で構成されます。

#### 手順

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

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

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

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

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

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

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

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

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

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

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

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

## 7. 前の手順で得た値を使用してバックエンドを作成します。

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

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

```

### 認証方法を更新するか、クレデンシャルをローテーションして

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

```

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

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

```



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

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

#### NFS エクスポートポリシーを管理します

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

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

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

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

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

## エクスポートポリシーを動的に管理

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



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

## 例

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

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



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

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

- `autoExportPolicy` がに設定されます `true`。これは、Astra Tridentがのエクスポートポリシーを作成することを示します `svm1` SVMで、を使用してルールの追加と削除を処理します `autoExportCIDRs` アドレスブロック。たとえば、UUID 403b5326-842-40dB-96d0-d83fb3f4daecのバックエンドです `autoExportPolicy` をに設定します `true` という名前のエクスポートポリシーを作成します `trident-403b5326-8482-40db-96d0-d83fb3f4daec` 指定します。

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

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

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

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

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

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

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



ントを防止します。

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



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

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

#### SMBボリュームをプロビジョニングする準備をします

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



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

作業を開始する前に

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

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

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

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

手順

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



Amazon FSx for ONTAPにはSMB共有が必要です。

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

a. 必要に応じて、共有のディレクトリパス構造を作成します。

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

b. 指定したSVMに関連付けられているSMB共有を作成します。

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

c. 共有が作成されたことを確認します。

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



を参照してください "[SMB 共有を作成](#)" 詳細については、

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

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

## ONTAP NASの設定オプションと例

Astra Tridentのインストール環境でONTAP NASドライバを作成して使用方法について

で説明します。このセクションでは、バックエンドの構成例と、バックエンドをStorage Classesにマッピングするための詳細を示します。

#### バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
version		常に 1
storageDriveName	ストレージドライバの名前	「ontap-nas」、 「ontap-nas-economy」、 「ontap-nas-flexgroup」、 「ontap-san」、 「ontap-san-economy」
backendName	カスタム名またはストレージバックエンド	ドライバ名+"_" + dataLIF
managementLIF	<p>クラスタ管理 LIF または SVM 管理 LIF の IP アドレス</p> <p>Fully Qualified Domain Name (FQDN ; 完全修飾ドメイン名) を指定できます。</p> <p>IPv6フラグを使用してAstra Tridentをインストールした場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、次のように角かっこで定義する必要があります。 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。</p> <p>シームレスなMetroClusterスイッチオーバーについては、を参照してください。 <a href="#">MetroClusterの例</a>。</p>	「 10.0.0.1 」、 「 [2001:1234:abcd::fefe] 」
dataLIF	<p>プロトコル LIF の IP アドレス。</p> <p>を指定することを推奨します dataLIF。指定しない場合は、Astra TridentがSVMからデータLIFを取得します。NFSマウント処理に使用するFully Qualified Domain Name (FQDN ; 完全修飾ドメイン名) を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。</p> <p>初期設定後に変更できます。を参照してください。</p> <p>IPv6フラグを使用してAstra Tridentをインストールした場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、次のように角かっこで定義する必要があります。 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。</p> <p>* MetroClusterの場合は省略してください。* <a href="#">MetroClusterの例</a>。</p>	指定されたアドレス、または指定されていない場合はSVMから取得されるアドレス (非推奨)

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

パラメータ	説明	デフォルト
debugTraceFlags	<p>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}</p> <p>使用しないでください debugTraceFlags トラブルシューティングを実行していて、詳細なログダンプが必要な場合を除きます。</p>	null
nasType	<p>NFSボリュームまたはSMBボリュームの作成を設定</p> <p>オプションはです nfs、 smb または null。 null に設定すると、デフォルトで NFS ボリュームが使用されます。</p>	nfs
nfsMountOptions	<p>NFSマウントオプションをカンマで区切ったリスト。</p> <p>Kubernetes永続ボリュームのマウントオプションは通常はストレージクラスで指定されますが、ストレージクラスでマウントオプションが指定されていない場合、Astra Tridentはストレージバックエンドの構成ファイルで指定されているマウントオプションを使用します。</p> <p>ストレージクラスや構成ファイルにマウントオプションが指定されていない場合、Astra Tridentは関連付けられた永続的ボリュームにマウントオプションを設定しません。</p>	""
qtreesPerFlexvol	<p>FlexVol あたりの最大 qtree 数。有効な範囲は [50、300] です。</p>	"200"
smbShare	<p>Microsoft管理コンソールまたはONTAP CLIを使用して作成されたSMB共有の名前、Astra TridentでSMB共有を作成できる名前、ボリュームへの共有アクセスを禁止する場合はパラメータを空白のままにすることができます。</p> <p>オンプレミスのONTAPでは、このパラメータはオプションです。</p> <p>このパラメータはAmazon FSx for ONTAPバックエンドで必須であり、空にすることはできません。</p>	smb-share

パラメータ	説明	デフォルト
useREST	<p>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 *</p> <p>useREST は、テクニカルプレビューとして提供されています。テスト環境では、本番環境のワークロードでは推奨されません。に設定すると true`Astra Tridentは、ONTAP REST APIを使用してバックエンドと通信します。この機能にはONTAP 9.11.1以降が必要です。また、使用するONTAP ログインロールにはへのアクセス権が必要です `ontap アプリケーション：これは事前定義されたによって満たされます vsadmin および cluster-admin ロール。</p> <p>useREST は、MetroCluster ではサポートされていません。</p>	いいえ

#### ボリュームのプロビジョニング用のバックエンド構成オプション

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

パラメータ	説明	デフォルト
spaceAllocation	space-allocation for LUN のコマンドを指定します	"正しい"
spaceReserve	スペースリザーベーションモード：「none」（シン）または「volume」（シック）	"なし"
snapshotPolicy	使用する Snapshot ポリシー	"なし"
qosPolicy	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプール/バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	""
adaptiveQosPolicy	<p>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。</p> <p>経済性に影響する ONTAP - NAS ではサポートされません。</p>	""
snapshotReserve	Snapshot 用にリザーブされているボリュームの割合	次の場合は「0」 snapshotPolicy は「none」、それ以外の場合は「」です。
splitOnClone	作成時にクローンを親からスプリットします	いいえ

パラメータ	説明	デフォルト
encryption	<p>新しいボリュームでNetApp Volume Encryption (NVE) を有効にします。デフォルトは <code>false</code>。このオプションを使用するには、クラスターで NVE のライセンスが設定され、有効になっている必要があります。</p> <p>NAEがバックエンドで有効になっている場合は、Astra TridentでプロビジョニングされたすべてのボリュームがNAEに有効になります。</p> <p>詳細については、以下を参照してください。"<a href="#">Astra TridentとNVEおよびNAEの相互運用性</a>"。</p>	いいえ
tieringPolicy	「none」を使用する階層化ポリシー	ONTAP 9.5より前のSVM-DR設定の場合は「snapshot-only」
unixPermissions	新しいボリュームのモード	NFSボリュームの場合は「777」、SMBボリュームの場合は空（該当なし）
snapshotDir	にアクセスする権限を管理します。 <code>.snapshot</code> ディレクトリ	いいえ
exportPolicy	使用するエクスポートポリシー	デフォルト
securityStyle	<p>新しいボリュームのセキュリティ形式。</p> <p>NFSのサポート <code>mixed</code> および <code>unix</code> セキュリティ形式</p> <p>SMBはをサポートします <code>mixed</code> および <code>ntfs</code> セキュリティ形式</p>	<p>NFSのデフォルトは <code>unix</code>。</p> <p>SMBのデフォルト： <code>ntfs</code>。</p>



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

#### ボリュームプロビジョニングの例

デフォルトが定義されている例を次に示します。

```

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

```

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

計算は次のとおりです。

```

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

```

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



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

2 entries were displayed.

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

#### 最小限の設定例

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



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

#### ONTAP NASエコノミーの例

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

#### ONTAP NAS FlexGroupの例

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

## MetroClusterの例

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

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

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

## SMBボリュームの例

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

## 証明書ベースの認証の例

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

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

## 自動エクスポートポリシーの例

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

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

## IPv6アドレスの例

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

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

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

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

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

## 仮想プールを使用するバックエンドの例

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

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

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

## ONTAP NASの例

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

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

## ONTAP NAS FlexGroupの例

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



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

```

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

```

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

バックエンドを **StorageClasses** にマッピングします

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

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

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

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

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

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

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

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

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

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

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

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

更新 dataLIF 初期設定後

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

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



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

## NetApp ONTAP 対応の Amazon FSX

Amazon FSX for NetApp ONTAP で Astra Trident を使用

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

### 概要

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

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

### 考慮事項

- SMBボリューム：
  - SMBボリュームは、を使用してサポートされます `ontap-nas` ドライバーのみ。
  - SMBボリュームはAstra Trident EKSアドオンではサポートされません。
  - Astra Tridentは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみをサポート
- Astra Trident 24.02より前のバージョンでは、自動バックアップが有効になっているAmazon FSxファイルシステム上に作成されたボリュームはTridentで削除できませんでした。Astra Trident 24.02以降でこの問題を無効にするには、`fsxFilesystemID`、`AWS apiRegion`、`AWS apikey`およびAWS `secretKey` AWS FSx for ONTAPのバックエンド構成ファイルに保存されます。



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

## FSx for ONTAPドライバの詳細

次のドライバを使用して、Astra TridentをAmazon FSX for NetApp ONTAP と統合できます。

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

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

### 認証

Astra Tridentは、2種類の認証モードを提供します。

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



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

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

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

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

### EKSのクラウドID

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

AWSでクラウドIDを利用するには、以下が必要です。

- EKSを使用して導入されるKubernetesクラスタ
- Astra Tridentをインストール（以下を含む） `cloudProvider` シティ `"AWS"` および `cloudIdentity`



## Trident オペレータ

Tridentオペレータを使用してAstra Tridentをインストールするには、tridentorchestrator\_cr.yaml をクリックして設定します cloudProvider 終了: "AWS" をクリックして設定します cloudIdentity をAWS IAMロールに割り当てます。

例:

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

## Helm

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

```
export CP="AWS"
export CI="'eks.amazonaws.com/role-arn:
arn:aws:iam::123456:role/astratrident-role'"
```

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

```
helm install trident trident-operator-100.2402.0.tgz --set
cloudProvider=$CP --set cloudIdentity=$CI
```

## <code>tridentctl</code>

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

```
export CP="AWS"
export CI="'eks.amazonaws.com/role-arn:
arn:aws:iam::123456:role/astratrident-role'"
```

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

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



詳細については、こちらをご覧ください

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

## NetApp ONTAP 向け Amazon FSX を統合します

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

### 要件

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

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



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

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

### ONTAP SAN と NAS ドライバの統合



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

### 手順

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

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

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



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

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

#### YAML

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

#### JSON

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

また、次の例に示すように、AWS Secret Managerに保存されているSVMのクレデンシャル（ユーザ名とパスワード）を使用してバックエンドファイルを作成することもできます。

## YAML

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: tbc-ontap-nas
  svm: svm-name
  aws:
    fsxFilesystemID: fs-xxxxxxxxxx
  managementLIF:
  credentials:
    name: "arn:aws:secretsmanager:us-west-2:xxxxxxxx:secret:secret-
name"
    type: awsarn
```

## JSON

```
{
  "apiVersion": "trident.netapp.io/v1",
  "kind": "TridentBackendConfig",
  "metadata": {
    "name": "backend-tbc-ontap-nas"
  },
  "spec": {
    "version": 1,
    "storageDriverName": "ontap-nas",
    "backendName": "tbc-ontap-nas",
    "svm": "svm-name",
    "aws": {
      "fsxFilesystemID": "fs-xxxxxxxxxx"
    },
    "managementLIF": null,
    "credentials": {
      "name": "arn:aws:secretsmanager:us-west-
2:xxxxxxxx:secret:secret-name",
      "type": "awsarn"
    }
  }
}
```

バックエンドの作成については、次のリンクを参照してください。

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

**SMBボリュームをプロビジョニングする準備をします**

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

作業を開始する前に

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

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

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

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

手順

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

- a. 必要に応じて、共有のディレクトリパス構造を作成します。

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

- b. 指定したSVMに関連付けられているSMB共有を作成します。

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

- c. 共有が作成されたことを確認します。

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



を参照してください ["SMB 共有を作成"](#) 詳細については、

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

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

## FSX (ONTAP の構成オプションと例)

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

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

パラメータ	説明	例
managementLIF	<p>クラスタ管理 LIF または SVM 管理 LIF の IP アドレス</p> <p>Fully Qualified Domain Name (FQDN ; 完全修飾ドメイン名) を指定できます。</p> <p>IPv6フラグを使用してAstra Tridentをインストールした場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、[28e8 : d9fb : a825 : b7bf : 69a8 : d02f : 9e7b : 3555]などの角かっこで定義する必要があります。</p>	<p>「 10.0.0.1 」、 「 [2001:1234:abcd::fefe] 」</p>
dataLIF	<p>プロトコル LIF の IP アドレス。</p> <p>* ONTAP NASドライバ* : データLIFを指定することを推奨します。指定しない場合は、Astra TridentがSVMからデータLIFを取得します。NFSマウント処理に使用するFully Qualified Domain Name (FQDN ; 完全修飾ドメイン名) を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。初期設定後に変更できます。を参照してください。</p> <p>* ONTAP SANドライバ* : iSCSIには指定しないでくださいTridentがONTAPの選択的LUNマップを使用して、マルチパスセッションの確立に必要なiSCSI LIFを検出します。データLIFが明示的に定義されている場合は警告が生成されません。</p> <p>IPv6フラグを使用してAstra Tridentをインストールした場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、[28e8 : d9fb : a825 : b7bf : 69a8 : d02f : 9e7b : 3555]などの角かっこで定義する必要があります。</p>	

パラメータ	説明	例
autoExportPolicy	<p>エクスポートポリシーの自動作成と更新を有効にします[ブーリアン]。</p> <p>を使用する autoExportPolicy および autoExportCIDRs ネットアップのAstra Tridentなら、エクスポートポリシーを自動的に管理できます。</p>	false
autoExportCIDRs	<p>KubernetesのノードIPをフィルタリングするCIDRのリスト autoExportPolicy が有効になります。</p> <p>を使用する autoExportPolicy および autoExportCIDRs ネットアップのAstra Tridentなら、エクスポートポリシーを自動的に管理できます。</p>	「[0.0.0.0/0]、 「::/0」」
labels	ボリュームに適用する任意のJSON形式のラベルのセット	""
clientCertificate	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""
clientPrivateKey	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""
trustedCACertificate	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます。	""
username	クラスタまたはSVMに接続するためのユーザ名。クレデンシャルベースの認証に使用されます。たとえば、vsadminのように指定します。	
password	クラスタまたはSVMに接続するためのパスワード。クレデンシャルベースの認証に使用されます。	
svm	使用する Storage Virtual Machine	SVM管理LIFが指定されている場合に生成されます。

パラメータ	説明	例
storagePrefix	<p>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。</p> <p>作成後に変更することはできません。このパラメータを更新するには、新しいバックエンドを作成する必要があります。</p>	trident
limitAggregateUsage	<p>* Amazon FSx for NetApp ONTAP には指定しないでください。*</p> <p>提供された fsxadmin および vsadmin アグリゲートの使用状況を取得し、Astra Tridentを使用して制限するために必要な権限が含まれていない。</p>	使用しないでください。
limitVolumeSize	<p>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。</p> <p>また、qtreeおよびLUN用に管理するボリュームの最大サイズも制限します qtreesPerFlexvol オプションを使用すると、FlexVol あたりの最大qtree数をカスタマイズできます。</p>	"" (デフォルトでは適用されません)
lunsPerFlexvol	<p>FlexVol あたりの最大LUN数。有効な範囲は50、200です。</p> <p>SANのみ。</p>	100
debugTraceFlags	<p>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}</p> <p>使用しないでください debugTraceFlags トラブルシューティングを実行していて、詳細なログダンプが必要な場合を除きます。</p>	null



パラメータ	説明	例
nfsMountOptions	<p>NFSマウントオプションをカンマで区切ったリスト。</p> <p>Kubernetes永続ボリュームのマウントオプションは通常はストレージクラスで指定されますが、ストレージクラスでマウントオプションが指定されていない場合、Astra Tridentはストレージバックエンドの構成ファイルで指定されているマウントオプションを使用します。</p> <p>ストレージクラスや構成ファイルにマウントオプションが指定されていない場合、Astra Tridentは関連付けられた永続的ボリュームにマウントオプションを設定しません。</p>	""
nasType	<p>NFSボリュームまたはSMBボリュームの作成を設定</p> <p>オプションはです <code>nfs</code>、<code>smb</code>、または<code>null</code>。</p> <p>*をに設定する必要があります <code>smb</code> SMBボリューム。*を<code>null</code>に設定すると、デフォルトでNFSボリュームが使用されます。</p>	<code>nfs</code>
qtreesPerFlexvol	<p>FlexVol あたりの最大 <code>qtree</code> 数。有効な範囲は [50、300] です。</p>	200
smbShare	<p>次のいずれかを指定できます。Microsoft管理コンソールまたはONTAP CLIを使用して作成されたSMB共有の名前、またはAstra TridentでSMB共有を作成できるようにする名前。</p> <p>このパラメータは、Amazon FSx for ONTAPバックエンドに必要です。</p>	<code>smb-share</code>

パラメータ	説明	例
useREST	<p>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 *</p> <p>useREST は、テクニカルプレビューとして提供されています。テスト環境では、本番環境のワークロードでは推奨されません。に設定すると true`Astra Trident は、ONTAP REST APIを使用してバックエンドと通信します。</p> <p>この機能にはONTAP 9.11.1以降が必要です。また、使用するONTAP ログインロールにはへのアクセス権が必要です `ontap アプリケーション：これは事前定義されたによって満たされます vsadmin および cluster-admin ロール。</p>	false
aws	<p>AWS FSx for ONTAPの構成ファイルでは、次の項目を指定できます。</p> <ul style="list-style-type: none"> <li>- fsxFilesystemID：AWS FSx ファイルシステムのIDを指定します。</li> <li>- apiRegion：AWS APIリージョン名。</li> <li>- apikey：AWS APIキー。</li> <li>- secretKey：AWSシークレットキー。</li> </ul>	"" "" ""
credentials	<p>AWS Secret Managerに保存するFSx SVMのクレデンシャルを指定します。</p> <ul style="list-style-type: none"> <li>- name：シークレットのAmazon Resource Name (ARN) 。SVMのクレデンシャルが含まれています。</li> <li>- type:に設定 awsarn。 を参照してください "<a href="#">AWS Secrets Managerシークレットの作成</a>" を参照してください。</li> </ul>	

## 更新 dataLIF 初期設定後

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

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



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

#### ボリュームのプロビジョニング用のバックエンド構成オプション

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

パラメータ	説明	デフォルト
spaceAllocation	space-allocation for LUN のコマンドを指定します	true
spaceReserve	スペースリザーベーションモード： 「none」（シン）または「volume」（シック）	none
snapshotPolicy	使用する Snapshot ポリシー	none
qosPolicy	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプールまたはバックエンドごとに、QOSPolicyまたはadaptiveQosPolicyのいずれかを選択します。  Trident が Astra で QoS ポリシーグループを使用するには、ONTAP 9.8 以降が必要です。  非共有のQoSポリシーグループを使用して、各コンスチチュエントに個別にポリシーグループを適用することを推奨します。共有 QoS ポリシーグループにより、すべてのワークロードの合計スループットに対して上限が適用されます。	「」
adaptiveQosPolicy	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプールまたはバックエンドごとに、QOSPolicyまたはadaptiveQosPolicyのいずれかを選択します。  経済性に影響する ONTAP - NAS ではサポートされません。	「」
snapshotReserve	Snapshot 「0」 用にリザーブされているボリュームの割合	状況 snapshotPolicy はです none、else 「」

パラメータ	説明	デフォルト
splitOnClone	作成時にクローンを親からスプリットします	false
encryption	<p>新しいボリュームでNetApp Volume Encryption (NVE) を有効にします。デフォルトはです false。このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。</p> <p>NAEがバックエンドで有効になっている場合は、Astra TridentでプロビジョニングされたすべてのボリュームがNAEに有効になります。</p> <p>詳細については、以下を参照してください。"<a href="#">Astra TridentとNVEおよびNAEの相互運用性</a>"。</p>	false
luksEncryption	<p>LUKS暗号化を有効にします。を参照してください "<a href="#">Linux Unified Key Setup (LUKS ; 統合キーセットアップ)</a> を使用"。</p> <p>SANのみ。</p>	""
tieringPolicy	使用する階層化ポリシー none	snapshot-only ONTAP 9.5より前のSVM-DR構成の場合
unixPermissions	<p>新しいボリュームのモード。</p> <p>* SMBボリュームは空にしておきます。*</p>	「」
securityStyle	<p>新しいボリュームのセキュリティ形式。</p> <p>NFSのサポート mixed および unix セキュリティ形式</p> <p>SMBはをサポートします mixed および ntfs セキュリティ形式</p>	<p>NFSのデフォルトはです unix。</p> <p>SMBのデフォルト： ntfs。</p>

構成例

## SMBボリュームノストレエシクラスノセツテイ

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

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

## Secret Managerを使用したAWS FSx for ONTAPの設定

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: tbc-ontap-nas
  svm: svm-name
  aws:
    fsxFileSystemID: fs-xxxxxxxxxx
  managementLIF:
  credentials:
    name: "arn:aws:secretsmanager:us-west-2:xxxxxxx:secret:secret-name"
    type: awsarn
```

## EKSクラスタでのAstra Trident EKSアドオンバージョン23.10の設定

Astra Tridentは、KubernetesでのAmazon FSx for NetApp ONTAPストレージ管理を合理化し、開発者や管理者がアプリケーションの導入に集中できるようにします。Astra Trident EKSアドオンには、最新のセキュリティパッチ、バグ修正が含まれており、AWSによってAmazon EKSとの連携が検証されています。EKSアドオンを使用する

と、Amazon EKSクラスタの安全性と安定性を一貫して確保し、アドオンのインストール、構成、更新に必要な作業量を削減できます。

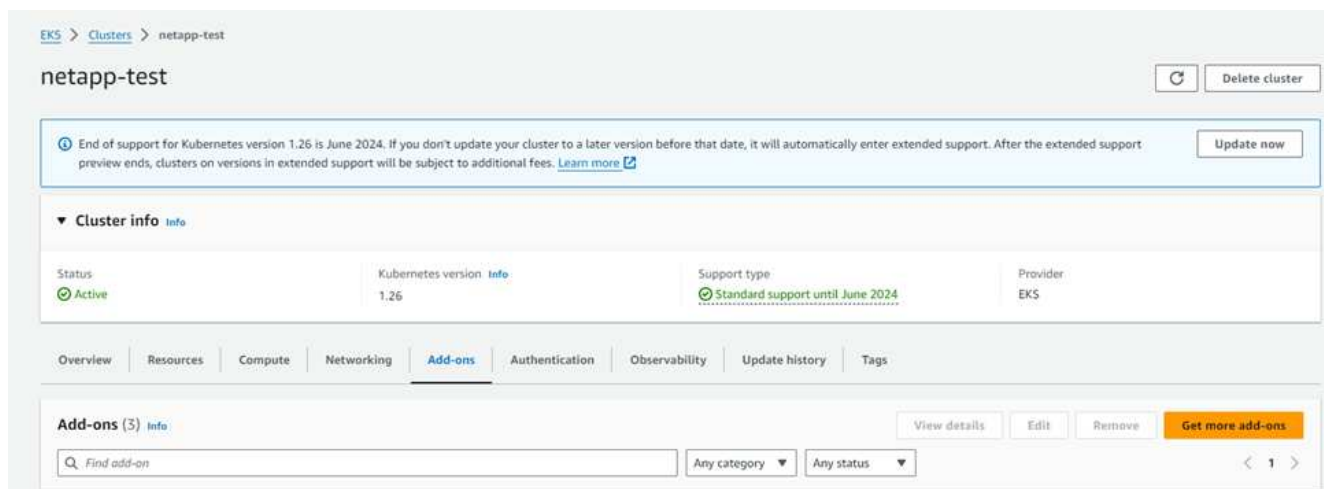
#### 前提条件

AWS EKS用のAstra Tridentアドオンを設定する前に、次の条件を満たしていることを確認してください。

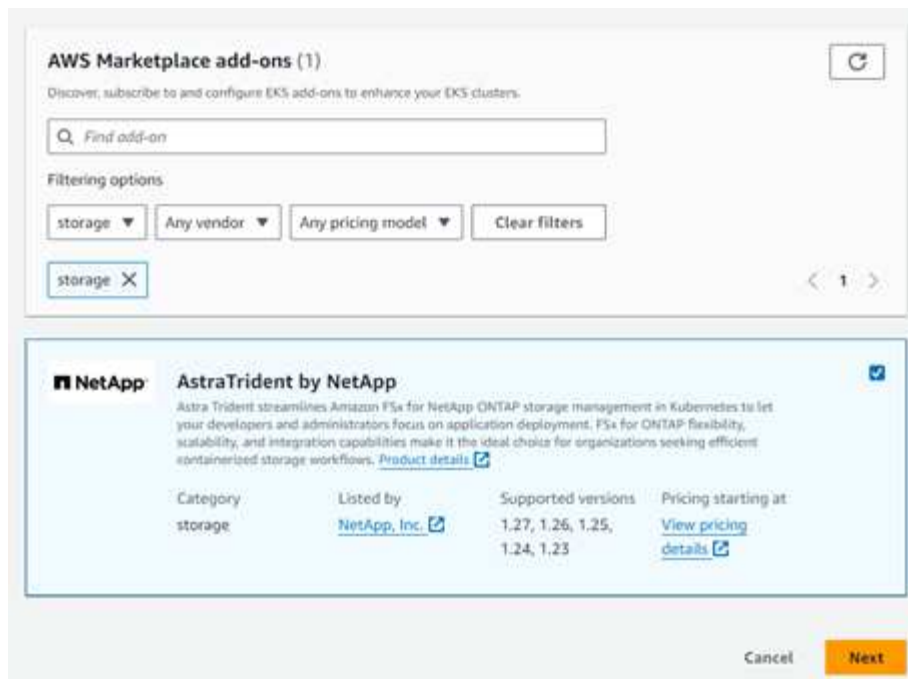
- アドオンサブスクリプションがあるAmazon EKSクラスタアカウント
- AWS MarketplaceへのAWS権限：  
"aws-marketplace:ViewSubscriptions",  
"aws-marketplace:Subscribe",  
"aws-marketplace:Unsubscribe
- AMIタイプ：Amazon Linux 2 (AL2\_x86\_64) またはAmazon Linux 2 Arm (AL2\_ARM\_64)
- ノードタイプ：AMDまたはARM
- 既存のAmazon FSx for NetApp ONTAPファイルシステム

#### 手順

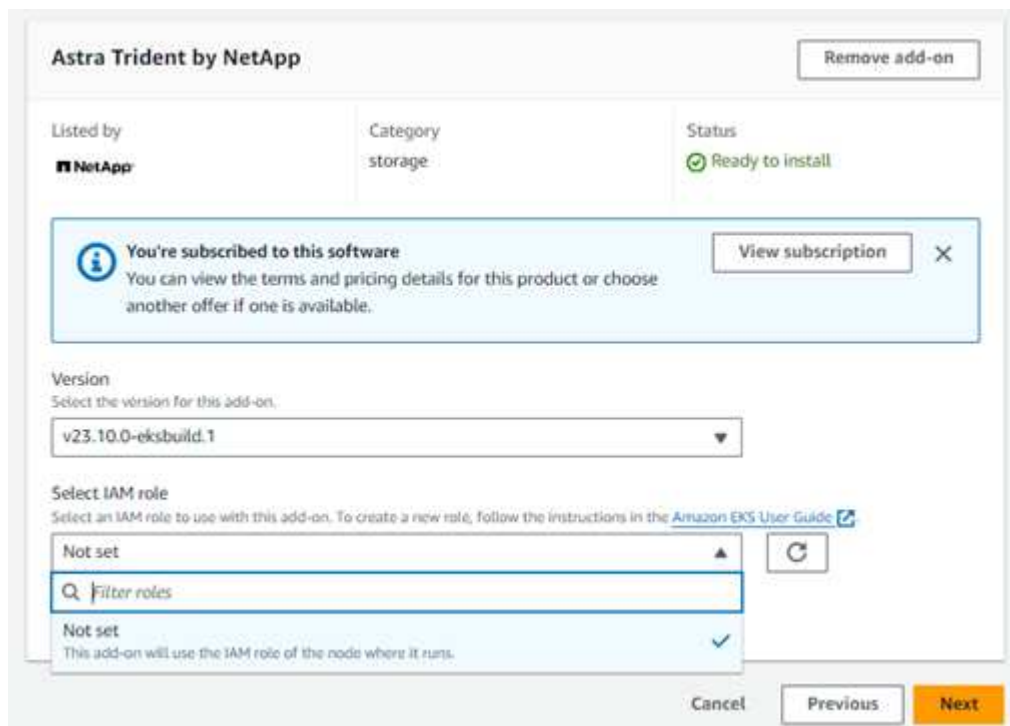
1. EKS Kubernetesクラスタで、\*アドオン\*タブに移動します。



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



3. [AstraTrident by NetApp \*]を探し、Astra Tridentアドオンのチェックボックスを選択します。
4. 必要なアドオンのバージョンを選択します。



5. ノードから継承するIAMロールオプションを選択します。
6. 必要に応じてオプションの設定を行い、\* Next \*を選択します。

### Review and add

**Step 1: Select add-ons** Edit

Selected add-ons

Find add-on

Add-on name	Type	Status
netapp_trident-operator	storage	Ready to install

**Step 2: Configure selected add-ons settings** Edit

Selected add-ons version

Add-on name	Version	IAM role
netapp_trident-operator	v23.10.0-eksbuild.1	Inherit from node

Cancel Previous Create

- 「\* Create \*」を選択します。
- アドオンのステータスが `_Active_` であることを確認します。

Add-ons (1) View details Edit Remove Get more add-ons

Find add-on

Any category

Any status

Category	Status	Version	IAM role	Created by
Storage	Active	v23.10.0-eksbuild.1	Inherited from node	NetApp, Inc.

#### CLIを使用したAstra Trident EKSアドオンのインストールとアンインストール

CLIを使用してAstra Trident EKSアドオンをインストールします。

次のコマンド例は、Astra Trident EKSアドオンをインストールします。

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

CLIを使用してAstra Trident EKSアドオンをアンインストールします。

次のコマンドは、Astra Trident EKSアドオンをアンインストールします。

```
eksctl delete addon --cluster K8s-arm --name netapp_trident-operator
```

#### kubectl を使用してバックエンドを作成します

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



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

TridentBackendConfig

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

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

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

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



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

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

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

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

。spec バックエンド固有の設定パラメータを使用します。この例では、バックエンドはを使用します ontap-san storage driverおよびでは、に示す構成パラメータを使用します。ご使用のストレージドライバの

設定オプションのリストについては、"[ストレージドライバのバックエンド設定情報](#)".

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

- credentials：このパラメータは必須フィールドで、ストレージシステム/サービスとの認証に使用されるクレデンシャルが含まれています。ユーザが作成した Kubernetes Secret に設定されます。クレデンシャルをプレーンテキストで渡すことはできないため、エラーになります。
- deletionPolicy:このフィールドは、がどうなるかを定義します TridentBackendConfig が削除されました。次の 2 つの値のいずれかを指定できます。
  - delete:この結果、両方が削除されます TridentBackendConfig CRとそれに関連付けられたバックエンド。これがデフォルト値です。
  - retain:時 TridentBackendConfig CRが削除され、バックエンド定義は引き続き存在し、で管理できます tridentctl。削除ポリシーをに設定しています retain 以前のリリース (21.04より前) にダウングレードし、作成されたバックエンドを保持することができます。このフィールドの値は、のあとに更新できます TridentBackendConfig が作成されます。



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



で作成されたバックエンド tridentctl が関連付けられていません TridentBackendConfig オブジェクト。このようなバックエンドの管理は、で選択できます kubect1 を作成します TridentBackendConfig CR。同一の設定パラメータ (など) を指定するように注意する必要があります spec.backendName、spec.storagePrefix、spec.storageDriverName` など)。新しく作成したTridentがAstraに自動的にバインドされる `TridentBackendConfig 既存のバックエンドを使用します。

## 手順の概要

を使用して新しいバックエンドを作成します `kubect1` では、次の操作を実行する必要があります。

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

バックエンドを作成したら、を使用してそのステータスを確認できます kubect1 get tbc <tbc-name> -n <trident-namespace> 追加の詳細情報を収集します。

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

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

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

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

ストレージプラットフォームシークレットフィールドの説明	秘密	Fields概要
Azure NetApp Files の特長	ClientID	アプリケーション登録からのクライアント ID
Cloud Volumes Service for GCP	private_key_id です	秘密鍵の ID。CVS 管理者ロールを持つ GCP サービスアカウントの API キーの一部
Cloud Volumes Service for GCP	private_key を使用します	秘密鍵CVS 管理者ロールを持つ GCP サービスアカウントの API キーの一部
Element ( NetApp HCI / SolidFire )	エンドポイント	テナントのクレデンシャルを使用する SolidFire クラスタの MVIP
ONTAP	ユーザ名	クラスタ / SVM に接続するためのユーザ名。クレデンシャルベースの認証に使用されます
ONTAP	パスワード	クラスタ / SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます
ONTAP	clientPrivateKey	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます
ONTAP	chapUsername のコマンド	インバウンドユーザ名。useCHAP = true の場合は必須。の場合 ontap-san および ontap-san-economy

ストレージプラットフォームシークレットフィールドの説明	秘密	Fields概要
ONTAP	chapInitiatorSecret	CHAP イニシエータシークレット。useCHAP = true の場合は必須。の場合 ontap-san および ontap-san-economy
ONTAP	chapTargetUsername のコマンド	ターゲットユーザ名。useCHAP = true の場合は必須。の場合 ontap-san および ontap-san-economy
ONTAP	chapTargetInitiatorSecret	CHAP ターゲットイニシエータシークレット。useCHAP = true の場合は必須。の場合 ontap-san および ontap-san-economy

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

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

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

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

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

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

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

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、にバインドされました TridentBackendConfig CR。

フェーズには次のいずれかの値を指定できます。

- Bound: TridentBackendConfig CRはバックエンドに関連付けられており、そのバックエンドには含まれています configRef をに設定します TridentBackendConfig crのuid
- Unbound:を使用して表されます ""。 TridentBackendConfig オブジェクトがバックエンドにバインドされていません。新しく作成されたすべてのファイル TridentBackendConfig CRSはデフォルトでこのフェーズになっています。フェーズが変更された後、再度 Unbound に戻すことはできません。
- Deleting: TridentBackendConfig CR deletionPolicy が削除対象に設定されました。をクリックします TridentBackendConfig CRが削除され、削除状態に移行します。
  - バックエンドに永続ボリューム要求 (PVC) が存在しない場合は、を削除します TridentBackendConfig その結果、Astra Tridentによってバックエンドとが削除されます TridentBackendConfig CR。
  - バックエンドに 1 つ以上の PVC が存在する場合は、削除状態になります。 TridentBackendConfig CRはその後、削除フェーズにも入ります。バックエンドと TridentBackendConfig は、すべてのPVCが削除されたあとにのみ削除されます。
- Lost:に関連付けられているバックエンド TridentBackendConfig CRが誤って削除されたか、故意に削除された TridentBackendConfig CRには削除されたバックエンドへの参照があります。 TridentBackendConfig CRは、に關係なく削除できます deletionPolicy 値。
- Unknown : Astra Tridentは、に関連付けられているバックエンドの状態または存在を特定できません TridentBackendConfig CR。たとえば、APIサーバが応答していない場合や、が応答していない場合などです tridentbackends.trident.netapp.io CRDがありません。これには介入が必要な場合があります

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

(オプション) 手順 4 : 詳細を確認します

バックエンドに関する詳細情報を確認するには、次のコマンドを実行します。

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

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

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

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

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

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

TridentBackendConfig CRとバックエンド。上記の例では、phase 値はバインドされています。これは、を意味します TridentBackendConfig CRはバックエンドに関連付けられています。

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



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

## バックエンドの管理

**kubectl** を使用してバックエンド管理を実行します

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

バックエンドを削除します

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

次のコマンドを実行します。

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

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

既存のバックエンドを表示します

次のコマンドを実行します。

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

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

バックエンドを更新します

バックエンドを更新する理由はいくつかあります。

- ストレージシステムのクレデンシャルが変更されている。クレデンシャルを更新する場合、で使用されるKubernetes Secret TridentBackendConfig オブジェクトを更新する必要があります。Astra Trident

が、提供された最新のクレデンシャルでバックエンドを自動的に更新次のコマンドを実行して、Kubernetes Secret を更新します。

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

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

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

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

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



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

**tridentctl** を使用してバックエンド管理を実行します

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

バックエンドを作成します

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

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

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

```
tridentctl logs -n trident
```

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

バックエンドを削除します

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



## 1. バックエンド名を取得します。

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

## 2. バックエンドを削除します。

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



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

既存のバックエンドを表示します

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

- 概要を取得するには、次のコマンドを実行します。

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

- すべての詳細を確認するには、次のコマンドを実行します。

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

バックエンドを更新します

新しいバックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs -n trident
```

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

バックエンドを使用するストレージクラスを特定します

以下は、回答 できるJSON形式の質問の例です tridentctl バックエンドオブジェクトの出力。これには

を使用します `jq` ユーティリティをインストールする必要があります。

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

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

バックエンド管理オプション間を移動します

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

バックエンドを管理するためのオプション

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

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

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

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

これは、次のシナリオに該当します。

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

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

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

を使用して、既存のバックエンドを管理します `kubectl` を作成する必要があります

`TridentBackendConfig` これは既存のバックエンドにバインドします。その仕組みの概要を以下に示します。

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

## 手順 0 : バックエンドを特定します

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

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

```
cat ontap-nas-backend.json

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

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

```
    "labels":{"app":"mysqldb", "cost":"25"},
    "zone":"us_east_1d",
    "defaults": {
      "spaceReserve": "volume",
      "encryption": "false",
      "unixPermissions": "0775"
    }
  }
]
}
```

### 手順 1 : Kubernetes Secret を作成します

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

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

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

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

### 手順2 : を作成します TridentBackendConfig CR

次の手順では、を作成します TridentBackendConfig 既存のに自動的にバインドされるCR ontap-nas-backend (この例のように)。次の要件が満たされていることを確認します。

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

この場合は、を参照してください TridentBackendConfig 次のようになります。

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

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

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

のあとに入力します TridentBackendConfig が作成されている必要があります Bound。また、既存のバックエンドと同じバックエンド名と UUID が反映されている必要があります。

```

kubect1 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、バックエンドはを使用して完全に管理されます tbc-ontap-nas-backend TridentBackendConfig オブジェクト。

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

``tridentctl`` を使用して、を使用して作成されたバックエンドを表示できます ``TridentBackendConfig``。また、管理者は、を使用してこのようなバックエンドを完全に管理することもできます ``tridentctl`` 削除します ``TridentBackendConfig`` そして確かめなさい ``spec.deletionPolicy`` がに設定されます ``retain``。

手順 0 : バックエンドを特定します

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

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

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

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

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

では、の値を見てみましょう deletionPolicy。これはに設定する必要があります retain。これにより、が確実に実行されます TridentBackendConfig CRが削除され、バックエンド定義は引き続き存在し、で管理できます tridentctl。

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



それ以外の場合は、次の手順に進まないでください `deletionPolicy` がに設定されます `retain`。

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

最後の手順は、を削除することです `TridentBackendConfig CR`。確認が完了したら `deletionPolicy` がに設定されます `retain` をクリックすると、次のように削除されます。

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

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

が削除されたとき `TridentBackendConfig Astra Trident`は、実際にバックエンド自体を削除することなく、単にオブジェクトを削除します。

## ストレージクラスの実成と管理

ストレージクラスを生成する。

Kubernetes `StorageClass` オブジェクトを設定してストレージクラスを生成し、`Astra Trident` でボリュームのプロビジョニング方法を指定

### Kubernetes `StorageClass` オブジェクトの設定

。 "[Kubernetes StorageClass オブジェクト](#)" そのクラスで使用するプロビジョニングツールとして `Astra Trident` を特定し、`Astra Trident` にボリュームのプロビジョニング方法を指示します。例：



```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

を参照してください "[Kubernetes オブジェクトと Trident オブジェクト](#)" ストレージクラスとの連携の詳細については、[こちら](#)を参照してください。PersistentVolumeClaim とパラメータを使用して、Astra Trident でポリシーをプロビジョニングする方法を制御します。

ストレージクラスを作成する。

StorageClass オブジェクトを作成したら、ストレージクラスを作成できます。 [[ストレージクラスノサンプル](#)] に、使用または変更できる基本的なサンプルを示します。

手順

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

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

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

```

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

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

```

ストレージクラスノサンプル

Astra Tridentの特長 ["特定のバックエンド向けのシンプルなストレージクラス定義"](#)。

または、sample-input/storage-class-csi.yaml.templ インストーラに付属しており、`BACKEND_TYPE` ストレージドライバの名前を指定します。

```

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

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

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

```

## ストレージクラスを管理する

既存のストレージクラスを表示したり、デフォルトのストレージクラスを設定したり、ストレージクラスバックエンドを識別したり、ストレージクラスを削除したりできます。

既存のストレージクラスを表示します

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

```
kubectl get storageclass
```

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

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

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

```
tridentctl get storageclass
```

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

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

## デフォルトのストレージクラスを設定する

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

- アノテーションを設定してデフォルトのストレージクラスを定義します  
storageclass.kubernetes.io/is-default-class をtrueに設定してストレージクラスの定義に追加します。仕様に応じて、それ以外の値やアノテーションがない場合は false と解釈されます。
- 次のコマンドを使用して、既存のストレージクラスをデフォルトのストレージクラスとして設定できます。

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

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

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

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



クラスタ内のデフォルトのストレージクラスは一度に1つだけにしてください。Kubernetes では、技術的に複数のストレージを使用することはできますが、デフォルトのストレージクラスがまったくない場合と同様に動作します。

## ストレージクラスのバックエンドを特定します

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

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

## ストレージクラスを削除する

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

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

<storage-class> は、ストレージクラスで置き換える必要があります。

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



Astra Tridentでは空白が強制される fsType を作成します。iSCSIバックエンドの場合は、適用することを推奨します parameters.fsType ストレージクラス。既存のストレージクラスを削除して、で再作成する必要があります parameters.fsType 指定された。

## ボリュームのプロビジョニングと管理

### ボリュームをプロビジョニングする

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

#### 概要

A "[永続ボリューム](#)" (PV) は、Kubernetesクラスタ上のクラスタ管理者がプロビジョニングする物理ストレージリソースです。。 "[PersistentVolumeClaim](#)" (PVC) は、クラスタ上のPersistentVolumeへのアクセス要求です。

PVCは、特定のサイズまたはアクセスモードのストレージを要求するように設定できます。クラスタ管理者は、関連付けられているStorageClassを使用して、PersistentVolumeのサイズとアクセスモード（パフォーマンスやサービスレベルなど）以上を制御できます。

PVとPVCを作成したら、ポッドにボリュームをマウントできます。

#### マニフェストの例

## PersistentVolumeサンプルマニフェスト

このサンプルマニフェストは、StorageClassに関連付けられた10Giの基本PVを示しています。basic-csi。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
  labels:
    type: local
spec:
  storageClassName: basic-csi
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/my/host/path"
```

## PersistentVolumeClaim サンプルマニフェスト

次に、基本的なPVC設定オプションの例を示します。

### RWOアクセスを備えたPVC

次の例は、という名前のStorageClassに関連付けられた、RWOアクセスが設定された基本的なPVCを示しています。basic-csi。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

### NVMe / TCP対応PVC

この例は、という名前のStorageClassに関連付けられたNVMe/TCPの基本的なPVCとRWOアクセスを示しています。protection-gold。

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-san-nvme
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 300Mi
  storageClassName: protection-gold
```

## PODマニフェストのサンプル

次の例は、PVCをポッドに接続するための基本的な設定を示しています。

キホンセット

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: pv-storage
      persistentVolumeClaim:
        claimName: basic
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: pv-storage
```



## NVMe/TCPの基本構成

```
---
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    resources: {}
    volumeMounts:
    - mountPath: "/usr/share/nginx/html"
      name: task-pv-storage
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  volumes:
  - name: task-pv-storage
    persistentVolumeClaim:
      claimName: pvc-san-nvme
```

## PVおよびPVCの作成

### 手順

1. PVを作成します。

```
kubectl create -f pv.yaml
```

2. PVステータスを確認します。

```
kubectl get pv
NAME          CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM
STORAGECLASS  REASON    AGE
pv-storage    4Gi       RWO           Retain          Available
7s
```

3. PVCを作成します。

```
kubectl create -f pvc.yaml
```

4. PVCステータスを確認します。

```
kubectl get pvc
NAME                STATUS VOLUME          CAPACITY ACCESS MODES STORAGECLASS AGE
pvc-storage        Bound  pv-name          2Gi      RWO                5m
```

5. ボリュームをポッドにマウントします。

```
kubectl create -f pv-pod.yaml
```



進捗状況は次を使用して監視できます。 `kubectl get pod --watch`。

6. ボリュームがマウントされていることを確認します。 `/my/mount/path`。

```
kubectl exec -it task-pv-pod -- df -h /my/mount/path
```

7. ポッドを削除できるようになりました。Podアプリケーションは存在しなくなりますが、ボリュームは残ります。

```
kubectl delete pod task-pv-pod
```

を参照してください ["Kubernetes オブジェクトと Trident オブジェクト"](#) ストレージクラスとの連携の詳細については、[を参照してください](#)。PersistentVolumeClaim とパラメータを使用して、Astra Trident でボリュームをプロビジョニングする方法を制御します。

## ボリュームを展開します

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

### iSCSI ボリュームを展開します

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



iSCSI ボリューム拡張は、`ontap-san`、`ontap-san-economy`、`solidfire-san` ドライバには Kubernetes 1.16 以降が必要です。

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

StorageClass定義を編集して allowVolumeExpansion フィールドからに移動します true。

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

既存のストレージクラスの場合は、編集してを追加します allowVolumeExpansion パラメータ

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

PVC定義を編集し、 spec.resources.requests.storage 新たに必要となったサイズを反映するには、元のサイズよりも大きくする必要があります。

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

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

```

kubect1 get pvc
NAME          STATUS      VOLUME                                     CAPACITY
ACCESS MODES  STORAGECLASS  AGE
san-pvc      Bound       pvc-8a814d62-bd58-4253-b0d1-82f2885db671  1Gi
RWO          ontap-san    8s

kubect1 get pv
NAME          CAPACITY  ACCESS MODES
RECLAIM POLICY  STATUS    CLAIM          STORAGECLASS  REASON    AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671  1Gi      RWO
Delete        Bound     default/san-pvc  ontap-san    10s

```

手順 3 : PVC を接続するポッドを定義します

サイズを変更するポッドにPVを接続します。iSCSI PV のサイズ変更には、次の 2 つのシナリオがあります。

- PV がポッドに接続されている場合、Astra Trident はストレージバックエンドのボリュームを拡張し、デバイスを再スキャンし、ファイルシステムのサイズを変更します。
- 未接続の PV のサイズを変更しようとする、Astra Trident がストレージバックエンドのボリュームを拡張します。PVC がポッドにバインドされると、Trident はデバイスを再スキャンし、ファイルシステムのサイズを変更します。展開操作が正常に完了すると、Kubernetes は PVC サイズを更新します。

この例では、を使用するポッドが作成されます san-pvc。

```

kubect1 get pod
NAME          READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0           65s

kubect1 describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
               csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod

```

#### ステップ 4 : PV を展開します

1Giから2Giに作成されたPVのサイズを変更するには、PVCの定義を編集してを更新します  
spec.resources.requests.storage 2Giへ。

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

#### 手順 5 : 拡張を検証する

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

```

kubect1 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
kubect1 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、でプロビジョニングしたNFS PVSのボリューム拡張をサポートしています ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、gcp-cvs`および `azure-netapp-files バックエンド

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

NFS PVのサイズを変更するには、管理者はまず、を設定してボリュームを拡張できるようにストレージクラスを構成する必要があります allowVolumeExpansion フィールドからに移動します true :

```

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

```

このオプションを指定せずにストレージクラスを作成済みの場合は、を使用して既存のストレージクラスを編集するだけです kubect1 edit storageclass ボリュームを拡張できるようにするため。

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

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

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

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

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

ステップ3 : **PV**を拡張する

新しく作成した20MiBのPVのサイズを1GiBに変更するには、そのPVCを編集してを設定します  
spec.resources.requests.storage 1GiBへ :

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

#### 手順4：拡張を検証する

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



```

kubect1 get pvc ontapnas20mb
NAME                STATUS      VOLUME
CAPACITY  ACCESS MODES  STORAGECLASS  AGE
ontapnas20mb  Bound        pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7  1Gi
RWO                ontapnas          4m44s

kubect1 get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                CAPACITY  ACCESS MODES
RECLAIM POLICY     STATUS    CLAIM                STORAGECLASS  REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7  1Gi      RWO
Delete            Bound     default/ontapnas20mb  ontapnas
5m35s

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

```

## ボリュームをインポート

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

### 概要と考慮事項

Astra Tridentにボリュームをインポートすると、次のことが可能になります。

- アプリケーションをコンテナ化し、既存のデータセットを再利用する
- 一時的なアプリケーションにはデータセットのクローンを使用
- 障害が発生したKubernetesクラスタを再構築します
- ディザスタリカバリ時にアプリケーションデータを移行

### 考慮事項

ボリュームをインポートする前に、次の考慮事項を確認してください。

- Astra TridentでインポートできるのはRW（読み取り/書き込み）タイプのONTAPボリュームのみです。DP（データ保護）タイプのボリュームはSnapMirrorデスティネーションボリュームです。ボリュームをAstra

Tridentにインポートする前に、ミラー関係を解除する必要があります。

- アクティブな接続がないボリュームをインポートすることを推奨します。アクティブに使用されているボリュームをインポートするには、ボリュームのクローンを作成してからインポートを実行します。



Kubernetesは以前の接続を認識せず、アクティブなボリュームをポッドに簡単に接続できるため、これはブロックボリュームで特に重要です。その結果、データが破損する可能性があります。

- でもね StorageClass PVCに対して指定する必要があります。Astra Tridentはインポート時にこのパラメータを使用しません。ストレージクラスは、ボリュームの作成時に、ストレージ特性に基づいて使用可能なプールから選択するために使用されます。ボリュームはすでに存在するため、インポート時にプールを選択する必要はありません。そのため、PVCで指定されたストレージクラスと一致しないバックエンドまたはプールにボリュームが存在してもインポートは失敗しません。
- 既存のボリュームサイズはPVCで決定され、設定されます。ストレージドライバによってボリュームがインポートされると、PVはClaimRefを使用してPVCに作成されます。
  - 再利用ポリシーは、最初から設定されています retain PVにあります。KubernetesがPVCとPVを正常にバインドすると、再利用ポリシーがストレージクラスの再利用ポリシーに合わせて更新されます。
  - ストレージクラスの再利用ポリシーが`delete`にすると、PVが削除されるとストレージボリュームが削除されます。
- デフォルトでは、Astra TridentがPVCを管理し、バックエンドでFlexVolとLUNの名前を変更します。を渡すことができます `--no-manage` 管理対象外のボリュームをインポートするフラグ。を使用する場合 `--no-manage` Astra Tridentは、オブジェクトのライフサイクルを通じてPVCやPVに対して追加の処理を実行することはありません。PVが削除されてもストレージボリュームは削除されず、ボリュームのクローンやボリュームのサイズ変更などのその他の処理も無視されます。`



このオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、Kubernetes 以外でストレージボリュームのライフサイクルを管理する場合に便利です。

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

## ボリュームをインポートします

を使用できます `tridentctl import` をクリックしてボリュームをインポートします。

### 手順

1. Persistent Volume Claim (PVC; 永続的ボリューム要求) ファイルを作成します (例: `pvc.yaml`) をクリックします。PVCファイルには、が含まれている必要があります `name`、`namespace`、`accessModes` および `storageClassName`。必要に応じて、を指定できます `unixPermissions` 定義されています。

最小仕様の例を次に示します。

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



PV名やボリュームサイズなどの追加のパラメータは指定しないでください。これにより原因、インポートコマンドが失敗する可能性があります。

2. を使用します `tridentctl import` コマンドを使用して、ボリュームを含むAstra Tridentバックエンドの名前と、ストレージ上のボリュームを一意に識別する名前（ONTAP FlexVol、Elementボリューム、Cloud Volumes Serviceパスなど）を指定します。。 `-f` PVCファイルへのパスを指定するには、引数が必要です。

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

## 例

サポートされているドライバについて、次のボリュームインポートの例を確認してください。

### ONTAP NASおよびONTAP NAS FlexGroup

Astra Tridentでは、を使用したボリュームインポートがサポートされます `ontap-nas` および `ontap-nas-flexgroup` ドライバ。



- `ontap-nas-economy` ドライバで`qtree`をインポートおよび管理できない。
- `ontap-nas` および `ontap-nas-flexgroup` ドライバでボリューム名の重複が許可されていません。

を使用して作成した各ボリューム `ontap-nas driver`はONTAP クラスタ上のFlexVol です。を使用してFlexVolをインポートする `ontap-nas` ドライバも同じように動作します。ONTAP クラスタにすでに存在するFlexVolは、としてインポートできます `ontap-nas PVC`。同様に、FlexGroup ボリュームはとしてインポートできません `ontap-nas-flexgroup PVC`

### ONTAP NASの例

次の例は、管理対象ボリュームと管理対象外ボリュームのインポートを示しています。

## 管理対象ボリューム

次の例は、という名前のボリュームをインポートします managed\_volume という名前のバックエンドで ontap\_nas :

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

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

## 管理対象外のボリューム

を使用する場合 --no-manage 引数に指定します。Astra Tridentはボリュームの名前を変更しません。

次に、をインポートする例を示します unmanaged\_volume をクリックします ontap\_nas バックエンド:

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

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

## ONTAP SAN

Astra Tridentでは、を使用したボリュームインポートがサポートされます ontap-san ドライバ。を使用したボリュームインポートはサポートされていません ontap-san-economy ドライバ。

Astra Tridentでは、単一のLUNを含むONTAP SAN FlexVolをインポートできます。これはと同じです ontap-

san ドライバ。FlexVol 内の各PVCおよびLUNにFlexVol を作成します。Astra TridentがFlexVolをインポートし、PVCの定義に関連付けます。

## ONTAP SANの例

次の例は、管理対象ボリュームと管理対象外ボリュームのインポートを示しています。

### 管理対象ボリューム

管理対象ボリュームの場合、Astra TridentはFlexVolの名前をに変更します `pvc-<uuid>` およびFlexVol内のLUNをからにフォーマットします `lun0`。

次の例は、をインポートします `ontap-san-managed` にあるFlexVol `ontap_san_default` バックエンド：

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

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

### 管理対象外のボリューム

次に、をインポートする例を示します `unmanaged_example_volume` をクリックします `ontap_san` バックエンド：

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
```

```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
|          NAME          | SIZE  | STORAGE CLASS |
PROTOCOL |          BACKEND UUID          | STATE | MANAGED |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228 | 1.0 GiB | san-blog      |
block    | e3275890-7d80-4af6-90cc-c7a0759f555a | online | false      |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

次の例に示すように、KubernetesノードのIQNとIQNを共有するigroupにLUNをマッピングすると、エラーが表示されます。LUN already mapped to initiator(s) in this group。ボリュームをインポートするには、イニシエータを削除するか、LUNのマッピングを解除する必要があります。

Vserver	Igroup	Protocol	OS Type	Initiators
svm0	k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3	iscsi	linux	iqn.1994-05.com.redhat:4c2e1cf35e0
svm0	unmanaged-example-igroup	mixed	linux	iqn.1994-05.com.redhat:4c2e1cf35e0

### 要素 (Element)

Astra Tridentでは、を使用したNetApp ElementソフトウェアとNetApp HCIボリュームのインポートがサポートされず solidfire-san ドライバ。



Element ドライバではボリューム名の重複がサポートされます。ただし、ボリューム名が重複している場合はAstra Tridentからエラーが返されます。回避策としてボリュームをクローニングし、一意のボリューム名を指定して、クローンボリュームをインポートします。

### 要素の例

次に、をインポートする例を示します element-managed バックエンドのボリューム element\_default。

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

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

### Google Cloud Platform の 1 つです

Astra Tridentでは、を使用したボリュームインポートがサポートされます gcp-cvs ドライバ。



NetApp Cloud Volumes Serviceから作成されたボリュームをGoogle Cloud Platformにインポートするには、ボリュームパスでボリュームを特定します。ボリュームパスは、ボリュームのエクスポートパスのの続く部分です :/。たとえば、エクスポートパスがの場合などです 10.0.0.1:/adroit-jolly-swift、ボリュームのパスはです adroit-jolly-swift。

### Google Cloud Platformの例

次に、をインポートする例を示します gcp-cvs バックエンドのボリューム gcpcvs\_YEppr を指定します adroit-jolly-swift。

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

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

### Azure NetApp Files の特長

Astra Tridentでは、を使用したボリュームインポートがサポートされます azure-netapp-files ドライバ。



Azure NetApp Filesボリュームをインポートするには、ボリュームパスでボリュームを特定します。ボリュームパスは、ボリュームのエクスポートパスのの続く部分です :/。たとえば、マウントパスがの場合などです 10.0.0.2:/importvoll1、ボリュームのパスはです importvoll1。

### Azure NetApp Filesの例

次に、をインポートする例を示します azure-netapp-files バックエンドのボリューム azurenetappfiles\_40517 を指定します importvoll1。

```
tridentctl import volume azurenetappfiles_40517 importvoll1 -f <path-to-pvc-file> -n trident
```

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

## ネームスペース間で**NFS**ボリュームを共有します

Tridentを使用すると、プライマリネームスペースにボリュームを作成し、1つ以上のセカンダリネームスペースで共有できます。

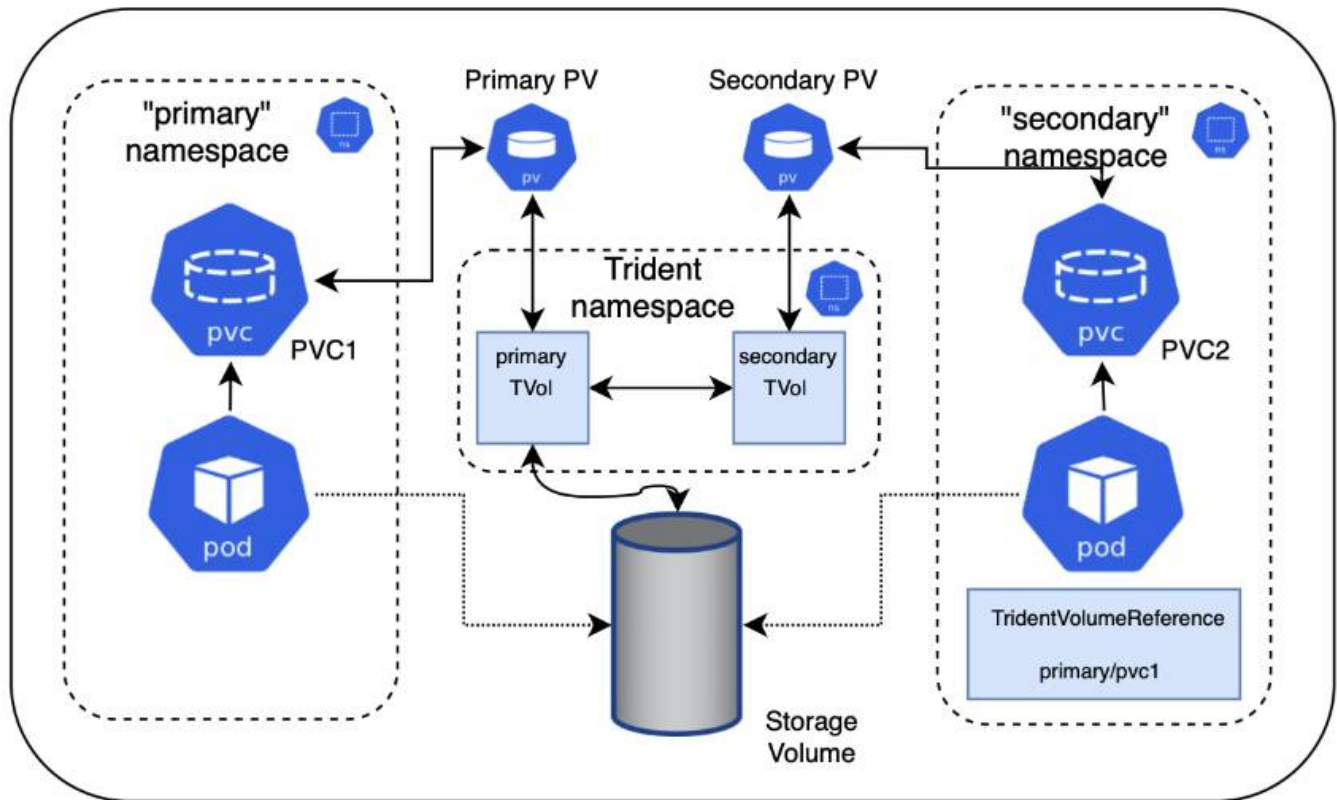
### の機能

Astra TridentVolumeReference CRを使用すると、1つ以上のKubernetesネームスペース間でReadWriteMany (RWX) NFSボリュームをセキュアに共有できます。このKubernetesネイティブ解決策には、次のようなメリットがあります。

- セキュリティを確保するために、複数のレベルのアクセス制御が可能です
- すべてのTrident NFSボリュームドライバで動作
- tridentctlやその他の非ネイティブのKubernetes機能に依存しません

この図は、2つのKubernetesネームスペース間でのNFSボリュームの共有を示しています。





## クイックスタート

NFSボリューム共有はいくつかの手順で設定できます。

- 1** ボリュームを共有するようにソースPVCを設定します  
 ソース名前空間の所有者は、ソースPVCのデータにアクセスする権限を付与します。
- 2** デスティネーション名前空間にCRを作成する権限を付与します  
 クラスタ管理者が、デスティネーション名前空間の所有者にTridentVolumeReference CRを作成する権限を付与します。
- 3** デスティネーション名前空間にTridentVolumeReferenceを作成します  
 宛先名前空間の所有者は、送信元PVCを参照するためにTridentVolumeReference CRを作成します。
- 4** 宛先名前空間に下位PVCを作成します  
 宛先名前空間の所有者は、送信元PVCからのデータソースを使用する下位PVCを作成します。

ソース名前空間とデスティネーション名前空間を設定します

セキュリティを確保するために、名前空間間共有では、ソース名前空間の所有者、クラスタ管理

者、および宛先名前空間の所有者によるコラボレーションとアクションが必要です。ユーザーロールは各手順で指定します。

#### 手順

1. ソース名前空間の所有者：PVCを作成します (pvc1) をソース名前空間に追加し、デスティネーション名前空間との共有権限を付与します (namespace2)を使用します shareToNamespace アノテーション

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/shareToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

#### Astra TridentがPVとバックエンドのNFSストレージボリュームを作成



- カンマ区切りリストを使用して、複数の名前空間にPVCを共有できます。例：  
trident.netapp.io/shareToNamespace:  
namespace2, namespace3, namespace4。
- \*を使用して、すべての名前空間に共有できます \*。例：  
trident.netapp.io/shareToNamespace: \*
- PVCを更新してを含めることができます shareToNamespace アノテーションはいつでも作成できます。

2. \*クラスタ管理者：\*カスタムロールとkubecfgを作成して、デスティネーション名前空間の所有者にTridentVolumeReference CRを作成する権限を付与します。
3. \*デスティネーション名前空間所有者：\*ソース名前空間を参照するデスティネーション名前空間にTridentVolumeReference CRを作成します pvc1。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

- 宛先名前空間の所有者：PVCを作成します (pvc2) をデスティネーション名前スペースに展開します (namespace2)を使用します shareFromPVC 送信元PVCを指定する注釈。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```



宛先PVCのサイズは、送信元PVCのサイズ以下である必要があります。

## 結果

Astra Tridentがを読み取り shareFromPVC デスティネーションPVCにアノテーションを設定し、ソースPVを参照するストレージリソースを持たない下位のボリュームとしてデスティネーションPVを作成し、ソースPVストレージリソースを共有します。宛先PVCとPVは、通常どおりバインドされているように見えます。

## 共有ボリュームを削除

複数の名前スペースで共有されているボリュームは削除できます。Tridentが、ソース名前スペースのボリュームへのアクセスを削除し、ボリュームを共有する他の名前スペースへのアクセスを維持します。ボリュームを参照するすべての名前スペースが削除されると、Astra Tridentによってボリュームが削除されます。

## 使用 tridentctl get 下位のボリュームを照会する

を使用する[tridentctl コマンド]を使用すると、を実行できます get コマンドを使用して下位のボリュームを取得します。詳細については、リンク:[./trident-reference/tridentctl.html](#)を参照してください [tridentctl コマンドとオプション]。

Usage:

```
tridentctl get [option]
```

フラグ:

- `-h, --help`: ボリュームのヘルプ。
- `--parentOfSubordinate string`: クエリを下位のソースボリュームに制限します。
- `--subordinateOf string`: クエリをボリュームの下位に制限します。

制限

- Astra Tridentでは、デスティネーション名前スペースが共有ボリュームに書き込まれるのを防ぐことはできません。共有ボリュームのデータの上書きを防止するには、ファイルロックなどのプロセスを使用する必要があります。
- を削除しても、送信元PVCへのアクセスを取り消すことはできません `shareToNamespace` または `shareFromNamespace` 注釈またはを削除します `TridentVolumeReference CR`。アクセスを取り消すには、下位PVCを削除する必要があります。
- Snapshot、クローン、およびミラーリングは下位のボリュームでは実行できません。

を参照してください。

名前スペース間のボリュームアクセスの詳細については、次の資料を参照してください。

- にアクセスします "名前スペース間でのボリュームの共有: 名前スペース間のボリュームアクセスを許可する場合は「Hello」と入力します"。
- のデモをご覧ください "ネットアップTV"。

## CSI トポロジを使用します

Astra Trident では、を使用して、Kubernetes クラスタ内にあるノードにボリュームを選択的に作成して接続できます **"CSI トポロジ機能"**。

概要

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



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

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

- を使用 `VolumeBindingMode` をに設定します `Immediate` トポロジを認識することなくボリュームを作成できます。ボリュームバインディングと動的プロビジョニングは、PVC が作成されるときに処理され

ます。これがデフォルトです `VolumeBindingMode` また、トポロジの制約を適用しないクラスタにも適しています。永続ボリュームは、要求元ポッドのスケジュール要件に依存することなく作成されます。

- を使用 `VolumeBindingMode` をに設定します `WaitForFirstConsumer` PVCの永続的ボリュームの作成とバインディングは、PVCを使用するポッドがスケジュールされて作成されるまで遅延されます。これにより、トポロジの要件に応じたスケジュールの制約を満たすようにボリュームが作成されます。



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

必要なもの

CSI トポロジを使用するには、次のものがが必要です。

- を実行するKubernetesクラスタ "[サポートされるKubernetesバージョン](#)"

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeaafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeaafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- クラスタ内のノードには、トポロジを認識するためのラベルが必要です (`topology.kubernetes.io/region` および `topology.kubernetes.io/zone`)。このラベル \* は、Astra Trident をトポロジ対応としてインストールする前に、クラスタ内のノードに存在する必要があります。

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

#### 手順 1 : トポロジ対応バックエンドを作成する

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

バックエンド定義の例を次に示します。

## YAML

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

## JSON

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



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

を定義できます supportedTopologies ストレージプールごとに作成することもできます。次の例を参照してください。

```

---
version: 1
storageDriverName: ontap-nas
backendName: nas-backend-us-centrall
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
- topology.kubernetes.io/region: us-centrall
  topology.kubernetes.io/zone: us-centrall-a
- topology.kubernetes.io/region: us-centrall
  topology.kubernetes.io/zone: us-centrall-b
storage:
- labels:
    workload: production
    region: Iowa-DC
    zone: Iowa-DC-A
    supportedTopologies:
    - topology.kubernetes.io/region: us-centrall
      topology.kubernetes.io/zone: us-centrall-a
- labels:
    workload: dev
    region: Iowa-DC
    zone: Iowa-DC-B
    supportedTopologies:
    - topology.kubernetes.io/region: us-centrall
      topology.kubernetes.io/zone: us-centrall-b

```

この例では、を使用しています region および zone ラベルはストレージプールの場所を表します。 topology.kubernetes.io/region および topology.kubernetes.io/zone ストレージプールの使用場所を指定します。

## 手順 2 : トポロジを認識するストレージクラスを定義する

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

次の例を参照してください。



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

上記のStorageClass定義で、volumeBindingMode がに設定されます WaitForFirstConsumer。このStorageClass で要求された PVC は、ポッドで参照されるまで処理されません。および、allowedTopologies 使用するゾーンとリージョンを提供します。。netapp-san-us-east1 StorageClassがにPVCを作成します san-backend-us-east1 上で定義したバックエンド。

### ステップ 3 : PVC を作成して使用する

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

例を参照 spec 下記：

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

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

```

kubect1 create -f pvc.yaml
persistentvolumeclaim/pvc-san created
kubect1 get pvc
NAME          STATUS      VOLUME      CAPACITY      ACCESS MODES      STORAGECLASS
AGE
pvc-san      Pending
2s
kubect1 describe pvc
Name:          pvc-san
Namespace:     default
StorageClass:  netapp-san-us-east1
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Mounted By:    <none>
Events:
  Type          Reason              Age   From
  ----          -
  Normal        WaitForFirstConsumer 6s    persistentvolume-controller
waiting
for first consumer to be created before binding

```

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: voll
        persistentVolumeClaim:
          claimName: pvc-san
    containers:
      - name: sec-ctx-demo
        image: busybox
        command: [ "sh", "-c", "sleep 1h" ]
        volumeMounts:
          - name: voll
            mountPath: /data/demo
        securityContext:
          allowPrivilegeEscalation: false

```

このpodSpecにより、Kubernetesは、にあるノードにPODをスケジュールするように指示されます us-east1 リージョンを選択し、にある任意のノードから選択します us-east1-a または us-east1-b ゾーン。

次の出力を参照してください。

```

kubect1 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>
kubect1 get pvc -o wide
NAME          STATUS   VOLUME                                     CAPACITY
ACCESS MODES  STORAGECLASS          AGE   VOLUMEMODE
pvc-san      Bound    pvc-ecb1e1a0-840c-463b-8b65-b3d033e2e62b 300Mi
RWO           netapp-san-us-east1  48s   Filesystem

```

## バックエンドを更新して追加 supportedTopologies

既存のバックエンドを更新して、のリストを追加することができます supportedTopologies を使用します tridentctl backend update。これは、すでにプロビジョニングされているボリュームには影響せず、以降の PVC にのみ使用されます。

詳細については、こちらをご覧ください

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

## スナップショットを操作します

永続ボリューム（PV）のKubernetesボリュームSnapshotを使用すると、ボリュームのポイントインタイムコピーを作成できます。Astra Tridentを使用して作成したボリュームのSnapshotの作成、Astra Trident外で作成したSnapshotのインポート、既存のSnapshotから新しいボリュームの作成、Snapshotからボリュームデータをリカバリできます。

### 概要

ボリュームSnapshotは、でサポートされます ontap-nas、ontap-nas-flexgroup、ontap-san、ontap-san-economy、solidfire-san、gcp-cvs`および`azure-netapp-files ドライバ。

作業を開始する前に

スナップショットを操作するには、外部スナップショットコントローラとカスタムリソース定義（CRD）が必要です。Kubernetesオーケストレーションツール（例：Kubeadm、GKE、OpenShift）の役割を担っています。

KubernetesディストリビューションにスナップショットコントローラとCRDが含まれていない場合は、を参照してください [ボリュームSnapshotコントローラの導入](#)。



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

## ボリューム **Snapshot** を作成します

### 手順

1. を作成します VolumeSnapshotClass。詳細については、を参照してください "[ボリュームSnapshotクラス](#)"。
  - driver Astra Trident CSIドライバを指します。
  - deletionPolicy は、です Delete または Retain。に設定すると Retain`を使用すると、ストレージクラスタの基盤となる物理Snapshotが、の場合でも保持されます `VolumeSnapshot オブジェクトが削除された。

### 例

```
cat snap-sc.yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

2. 既存のPVCのスナップショットを作成します。

### 例

- 次に、既存のPVCのスナップショットを作成する例を示します。

```
cat snap.yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: pvc1-snap
spec:
  volumeSnapshotClassName: csi-snapclass
  source:
    persistentVolumeClaimName: pvc1
```

- 次の例は、という名前のPVCのボリュームSnapshotオブジェクトを作成します。pvc1 Snapshotの名前には設定されます pvc1-snap。ボリュームSnapshotはPVCに似ており、に関連付けられています VolumeSnapshotContent 実際のスナップショットを表すオブジェクト。

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

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

- 次の情報を確認できます。VolumeSnapshotContent のオブジェクト pvc1-snap ボリュームSnapshot。ボリュームSnapshotの詳細を定義します。。 Snapshot Content Name このSnapshotを提供するVolumeSnapshotContentオブジェクトを特定します。。 Ready To Use パラメータは、スナップショットを使用して新しいPVCを作成できることを示します。

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

## ボリュームSnapshotからPVCを作成

使用できます dataSource という名前のVolumeSnapshotを使用してPVCを作成するには <pvc-name> データのソースとして。作成された PVC は、ポッドに接続して、他の PVC と同様に使用できます。



PVCはソースボリュームと同じバックエンドに作成されます。を参照してください ["KB : Trident PVCスナップショットからPVCを作成することは代替バックエンドではできない"](#)。

次に、を使用してPVCを作成する例を示します。 pvc1-snap をデータソースとして使用します。

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

## ボリュームSnapshotのインポート

Astra Tridentは以下をサポートします。"[Kubernetesの事前プロビジョニングされたSnapshotプロセス](#)" クラスタ管理者が VolumeSnapshotContent Astra Tridentの外部で作成されたオブジェクトとSnapshotをインポート

作業を開始する前に

Astra TridentでSnapshotの親ボリュームが作成またはインポートされている必要があります。

手順

1. クラスタ管理者： VolumeSnapshotContent バックエンドスナップショットを参照するオブジェクト。これにより、Astra TridentでSnapshotワークフローが開始されます。
  - バックエンドスナップショットの名前を annotations として  
trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">。
  - を指定します <name-of-parent-volume-in-trident>/<volume-snapshot-content-name>  
インテリ snapshotHandle。Astra Tridentに提供される唯一の情報は、ListSnapshots 電話だ



◦ <volumeSnapshotContentName> CRの命名規則のため、バックエンドスナップショット名が常に一致するとは限りません。

例

次の例では、VolumeSnapshotContent バックエンドスナップショットを参照するオブジェクト snap-01。

```

apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: import-snap-content
  annotations:
    trident.netapp.io/internalSnapshotName: "snap-01" # This is the
name of the snapshot on the backend
spec:
  deletionPolicy: Retain
  driver: csi.trident.netapp.io
  source:
    snapshotHandle: pvc-f71223b5-23b9-4235-bbfe-e269ac7b84b0/import-
snap-content # <import PV name or source PV name>/<volume-snapshot-
content-name>

```

2. クラスタ管理者：VolumeSnapshot を参照するCR VolumeSnapshotContent オブジェクト。これにより、VolumeSnapshot 指定された名前空間内。

例

次の例では、VolumeSnapshot CR名 import-snap を参照しています。VolumeSnapshotContent 名前付き import-snap-content。

```

apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: import-snap
spec:
  # volumeSnapshotClassName: csi-snapclass (not required for pre-
provisioned or imported snapshots)
  source:
    volumeSnapshotContentName: import-snap-content

```

3. \*内部処理（アクション不要）：\*外部スナップショットは、新しく作成されたスナップショットを認識します。VolumeSnapshotContent を実行します。ListSnapshots 電話だAstra Tridentが TridentSnapshot。
  - 外部スナップショットは、VolumeSnapshotContent 終了：readyToUse および VolumeSnapshot 終了：true。
  - Tridentのリターン readyToUse=true。
4. 任意のユーザー：PersistentVolumeClaim 新しい VolumeSnapshot` を参照してください `spec.dataSource`（または spec.dataSourceRef）nameは VolumeSnapshot 名前。

例

次に、を参照するPVCを作成する例を示します。VolumeSnapshot 名前付き import-snap。



```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: simple-sc
  resources:
    requests:
      storage: 1Gi
  dataSource:
    name: import-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

## Snapshotを使用したボリュームデータのリカバリ

Snapshotディレクトリは、を使用してプロビジョニングされるボリュームの互換性を最大限に高めるため、デフォルトでは非表示になっています。ontap-nas および ontap-nas-economy ドライバ。を有効にします .snapshot スナップショットからデータを直接リカバリするディレクトリ。

ボリュームを以前のSnapshotに記録されている状態にリストアするには、ボリュームSnapshotリストアONTAP CLIを使用します。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```



Snapshotコピーをリストアすると、既存のボリューム設定が上書きされます。Snapshotコピーの作成後にボリュームデータに加えた変更は失われます。

## Snapshotが関連付けられているPVを削除する

スナップショットが関連付けられている永続ボリュームを削除すると、対応する Trident ボリュームが「削除状態」に更新されます。ボリュームSnapshotを削除してAstra Tridentボリュームを削除します。

## ボリュームSnapshotコントローラの導入

KubernetesディストリビューションにスナップショットコントローラとCRDが含まれていない場合は、次のように導入できます。

### 手順

1. ボリュームのSnapshot作成

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

## 2. スナップショットコントローラを作成します。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```



必要に応じて、を開きます `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` およびを更新します namespace に移動します。

### 関連リンク

- ["ボリューム Snapshot"](#)
- ["ボリュームSnapshotクラス"](#)

## 著作権に関する情報

Copyright © 2024 NetApp, Inc. All Rights Reserved. Printed in the U.S.このドキュメントは著作権によって保護されています。著作権所有者の書面による事前承諾がある場合を除き、画像媒体、電子媒体、および写真複写、記録媒体、テープ媒体、電子検索システムへの組み込みを含む機械媒体など、いかなる形式および方法による複製も禁止します。

ネットアップの著作物から派生したソフトウェアは、次に示す使用許諾条項および免責条項の対象となります。

このソフトウェアは、ネットアップによって「現状のまま」提供されています。ネットアップは明示的な保証、または商品性および特定目的に対する適合性の暗示的保証を含み、かつこれに限定されないいかなる暗示的な保証も行いません。ネットアップは、代替品または代替サービスの調達、使用不能、データ損失、利益損失、業務中断を含み、かつこれに限定されない、このソフトウェアの使用により生じたすべての直接的損害、間接的損害、偶発的損害、特別損害、懲罰的損害、必然的損害の発生に対して、損失の発生の可能性が通知されていたとしても、その発生理由、根拠とする責任論、契約の有無、厳格責任、不法行為（過失またはそうでない場合を含む）にかかわらず、一切の責任を負いません。

ネットアップは、ここに記載されているすべての製品に対する変更を随時、予告なく行う権利を保有します。ネットアップによる明示的な書面による合意がある場合を除き、ここに記載されている製品の使用により生じる責任および義務に対して、ネットアップは責任を負いません。この製品の使用または購入は、ネットアップの特許権、商標権、または他の知的所有権に基づくライセンスの供与とはみなされません。

このマニュアルに記載されている製品は、1つ以上の米国特許、その他の国の特許、および出願中の特許によって保護されている場合があります。

権利の制限について：政府による使用、複製、開示は、DFARS 252.227-7013（2014年2月）およびFAR 5252.227-19（2007年12月）のRights in Technical Data -Noncommercial Items（技術データ - 非商用品目に関する諸権利）条項の(b)(3)項、に規定された制限が適用されます。

本書に含まれるデータは商用製品および / または商用サービス（FAR 2.101の定義に基づく）に関係し、データの所有権はNetApp, Inc.にあります。本契約に基づき提供されるすべてのネットアップの技術データおよびコンピュータソフトウェアは、商用目的であり、私費のみで開発されたものです。米国政府は本データに対し、非独占的かつ移転およびサブライセンス不可で、全世界を対象とする取り消し不能の制限付き使用权を有し、本データの提供の根拠となった米国政府契約に関連し、当該契約の裏付けとする場合にのみ本データを使用できます。前述の場合を除き、NetApp, Inc.の書面による許可を事前に得ることなく、本データを使用、開示、転載、改変するほか、上演または展示することはできません。国防総省にかかる米国政府のデータ使用权については、DFARS 252.227-7015(b)項（2014年2月）で定められた権利のみが認められます。

## 商標に関する情報

NetApp、NetAppのロゴ、<http://www.netapp.com/TM>に記載されているマークは、NetApp, Inc.の商標です。その他の会社名と製品名は、それを所有する各社の商標である場合があります。