FSxN を使用した AWS 上の Red Hat OpenShift サービス

NetApp container solutions

NetApp
January 21, 2026

目次

FSxN を使用した AWS 上の Red Hat OpenShift サービス

NetApp ONTAPを使用した AWS 上の Red Hat OpenShift サービス

概要

このセクションでは、ROSA 上で実行されるアプリケーションの永続ストレージ レイヤーとして FSx for ONTAPを活用する方法を説明します。 ROSA クラスターへのNetApp Trident CSI ドライバーのインストール、FSx for ONTAPファイルシステムのプロビジョニング、サンプルのステートフル アプリケーションの展開について説明します。また、アプリケーション データのバックアップと復元の戦略も示します。この統合ソリューションを使用すると、AZ 間で簡単に拡張できる共有ストレージ フレームワークを確立でき、Trident CSI ドライバーを使用してデータの拡張、保護、および復元のプロセスを簡素化できます。

前提条件

- "AWSアカウント"
- "Red Hatアカウント"
- IAMユーザー"適切な権限を持つ"ROSAクラスタを作成してアクセスする
- "AWS CLI"
- "ローザCLI"
- "OpenShift コマンドラインインターフェース"（oc）
- ヘルム3"ドキュメント"
- "HCP ROSAクラスター"
- "Red Hat OpenShift Webコンソールへのアクセス"

この図は、複数の AZ に展開された ROSA クラスターを示しています。 ROSA クラスターのマスターノード、インフラストラクチャノードは Red Hat の VPC 内にありますが、ワーカーノードは顧客のアカウントの VPC 内にあります。同じ VPC 内に FSx for ONTAPファイル システムを作成し、ROSA クラスターにTridentドライバーをインストールして、この VPC のすべてのサブネットがファイル システムに接続できるようにします。

初期セットアップ

1. NetApp ONTAP用の **FSx** のプロビジョニング

ROSA クラスターと同じ VPC にNetApp ONTAP用のマルチ AZ FSx を作成します。これにはいくつかの方法があります。 CloudFormationスタックを使用してFSxNを作成する詳細が提供されています

a.GitHubリポジトリをクローンする

```
$ git clone https://github.com/aws-samples/rosa-fsx-netapp-ontap.git
```

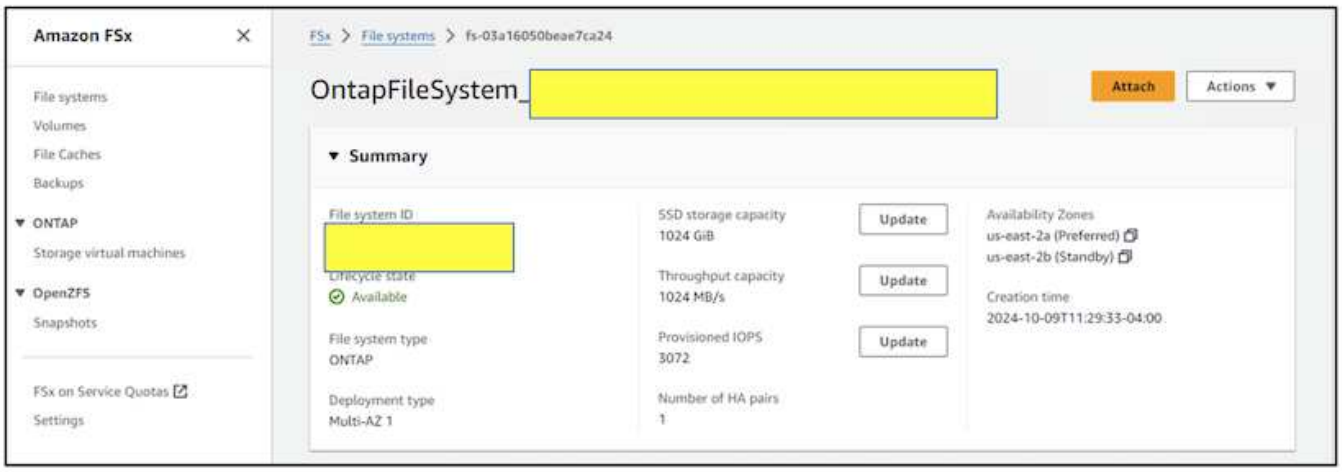
b. CloudFormation スタックを実行する パラメータ値を独自の値に置き換えて、以下のコマンドを実行します。

```
$ cd rosa-fsx-netapp-ontap/fsx
```

```
$ aws cloudformation create-stack \
  --stack-name ROSA-FSXONTAP \
  --template-body file://./FSxONTAP.yaml \
  --region <region-name> \
  --parameters \
  ParameterKey=Subnet1ID,ParameterValue=[subnet1_ID] \
  ParameterKey=Subnet2ID,ParameterValue=[subnet2_ID] \
  ParameterKey=myVpc,ParameterValue=[VPC_ID] \
ParameterKey=FSxONTAPRouteTable,ParameterValue=[routetable1_ID,routetable2
_ID] \
  ParameterKey=FileSystemName,ParameterValue=ROSA-myFSxONTAP \
  ParameterKey=ThroughputCapacity,ParameterValue=1024 \
  ParameterKey=FSxAllowedCIDR,ParameterValue=[your_allowed_CIDR] \
  ParameterKey=FsxAdminPassword,ParameterValue=[Define Admin password] \
  ParameterKey=SvmAdminPassword,ParameterValue=[Define SVM password] \
  --capabilities CAPABILITY_NAMED_IAM
```

ここで、region-name: ROSA クラスターがデプロイされているリージョンと同じ、subnet1_ID: FSxN の優先サブネットの ID、subnet2_ID: FSxN のスタンバイ サブネットの ID、VPC_ID: ROSA クラスターがデプロイされている VPC の ID、routetable1_ID、routetable2_ID: 上記で選択したサブネットに関連付けられているルート テーブルの ID、your_allowed_CIDR: アクセスを制御するための FSx for ONTAPセキュリティ グループの入力ルールで許可される CIDR 範囲。 0.0.0.0/0 または適切な CIDR を使用して、すべてのトラフィックが FSx for ONTAPの特定のポートにアクセスできるようにすることができます。管理者パスワードの定義: FSxN にログインするためのパスワード。SVM パスワードの定義: 作成される SVM にログインするためのパスワード。

以下に示すように、 Amazon FSxコンソールを使用してファイルシステムとストレージ仮想マシン (SVM) が作成されたことを確認します。



2. ROSA クラスタ用の**Trident CSI** ドライバをインストールして構成する

- Tridentをインストールする**

ROSA クラスター ワーカー ノードには、ストレージのプロビジョニングとアクセスに NAS プロトコルを使

用できるようにする NFS ツールが事前構成されています。

代わりに iSCSI を使用する場合は、iSCSI 用にワーカーノードを準備する必要があります。 Trident 25.02 リリース以降では、ROSA クラスター (または任意の OpenShift クラスター) のワーカー ノードを簡単に準備して、FSxN ストレージで iSCSI 操作を実行できます。 iSCSI のワーカー ノードの準備を自動化するTrident 25.02 (またはそれ以降) をインストールするには、2 つの簡単な方法があります。 1. tridentctl ツールを使用して、コマンド ラインから node-prep-flag を使用する。2.オペレーターハブから Red Hat 認定Tridentオペレーターを使用してカスタマイズします。 3.Helm を使用する。

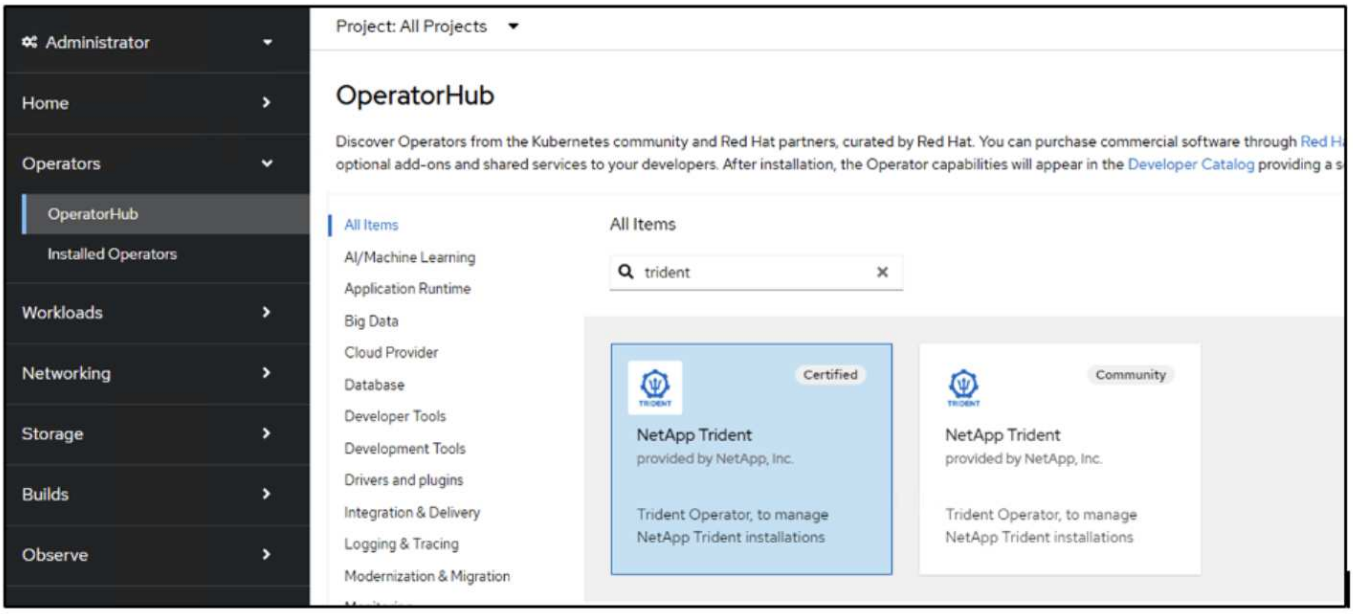
node-prep を有効にせずに上記のいずれかの方法を使用すると、FSxN 上のストレージのプロビジョニングに NAS プロトコルのみを使用できるようになります。

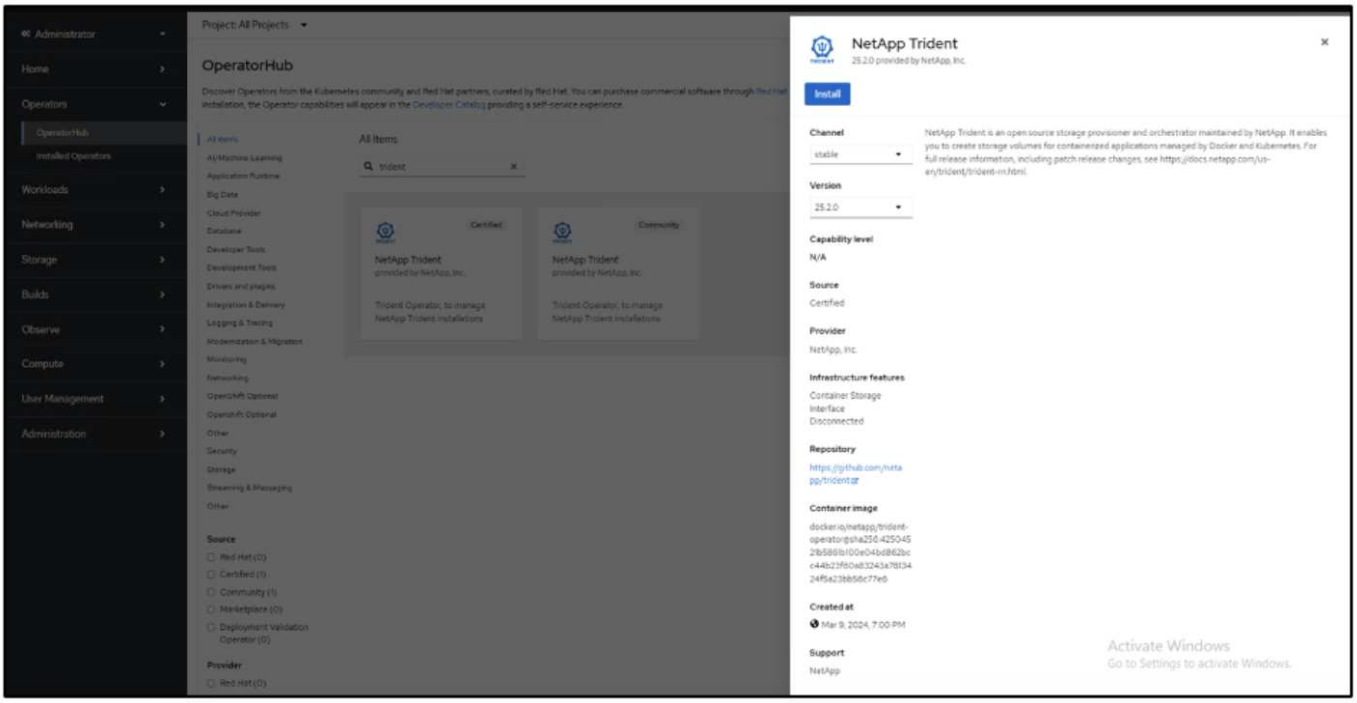
方法 **1: tridentctl** ツールを使用する

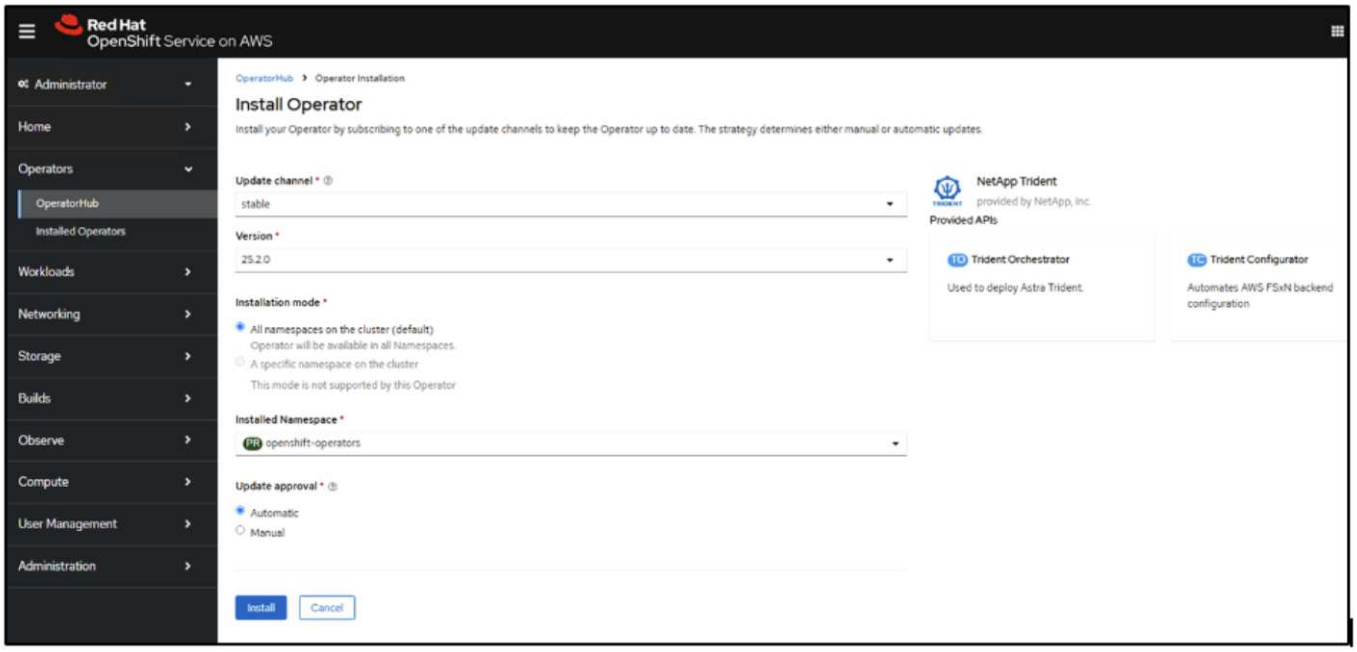
node-prep フラグを使用して、示されているようにTridentをインストールします。インストール コマンドを発行する前に、インストーラ パッケージをダウンロードしておく必要があります。。 "ドキュメントはこちら" 。

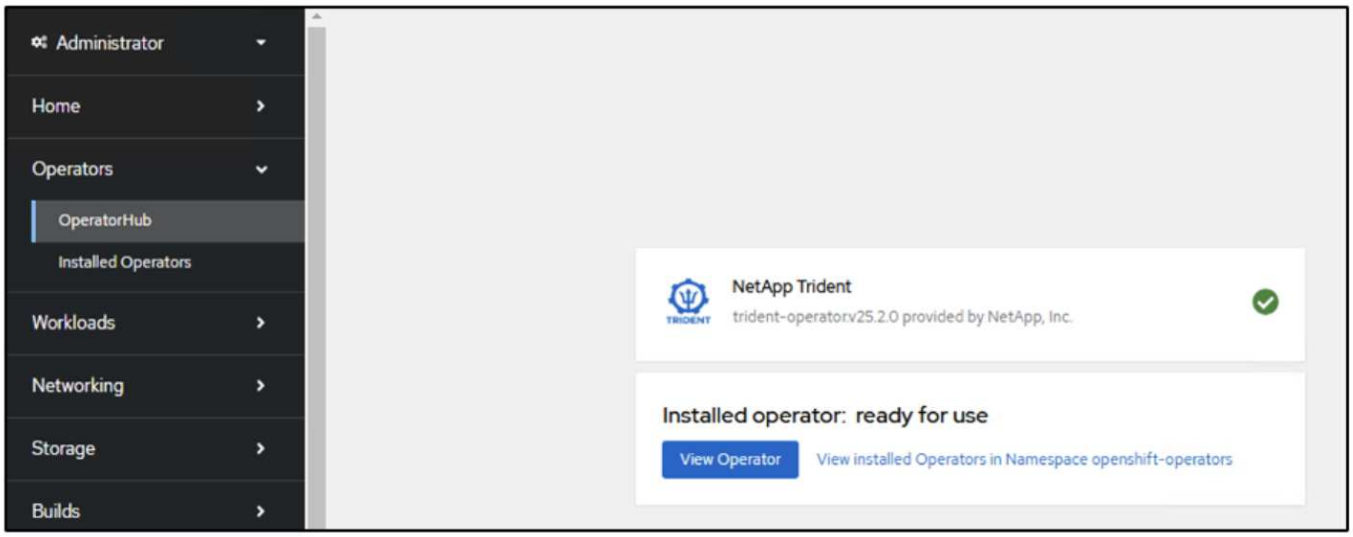
```
#./tridentctl install trident -n trident --node-prep=iscsi
```

方法 **2: Red Hat** 認定**Trident Operator** を使用してカスタマイズする OperatorHub から、Red Hat 認定Trident Operator を見つけてインストールします。

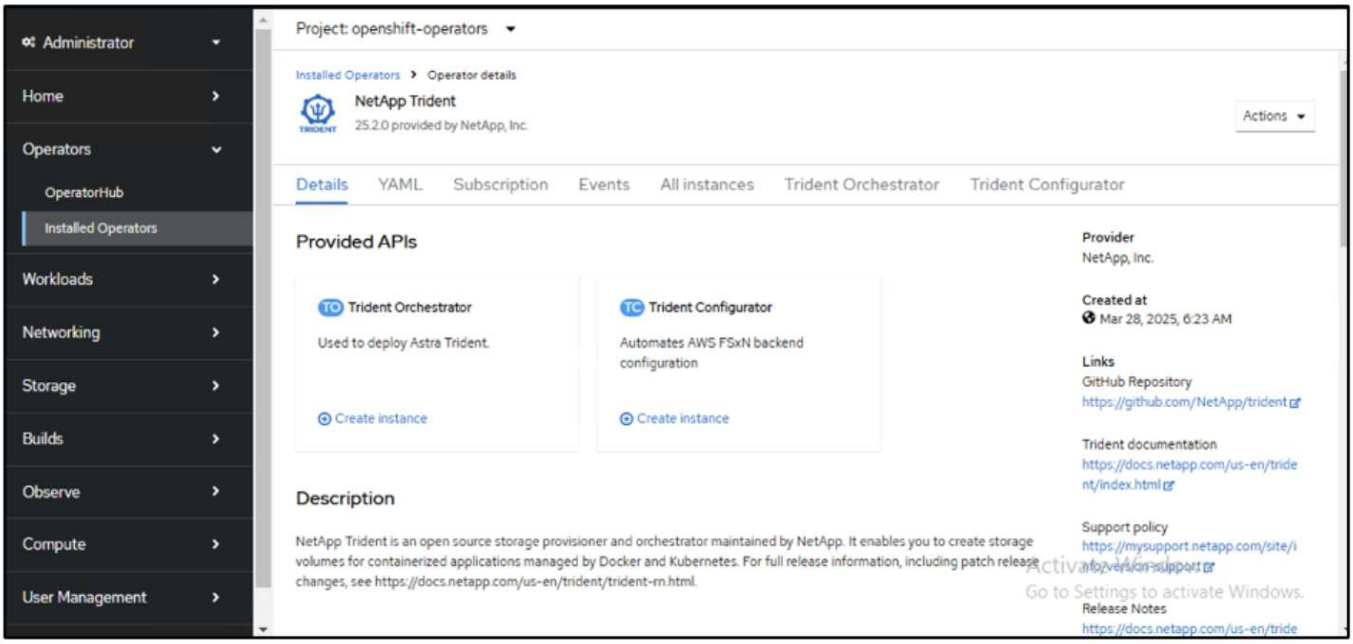


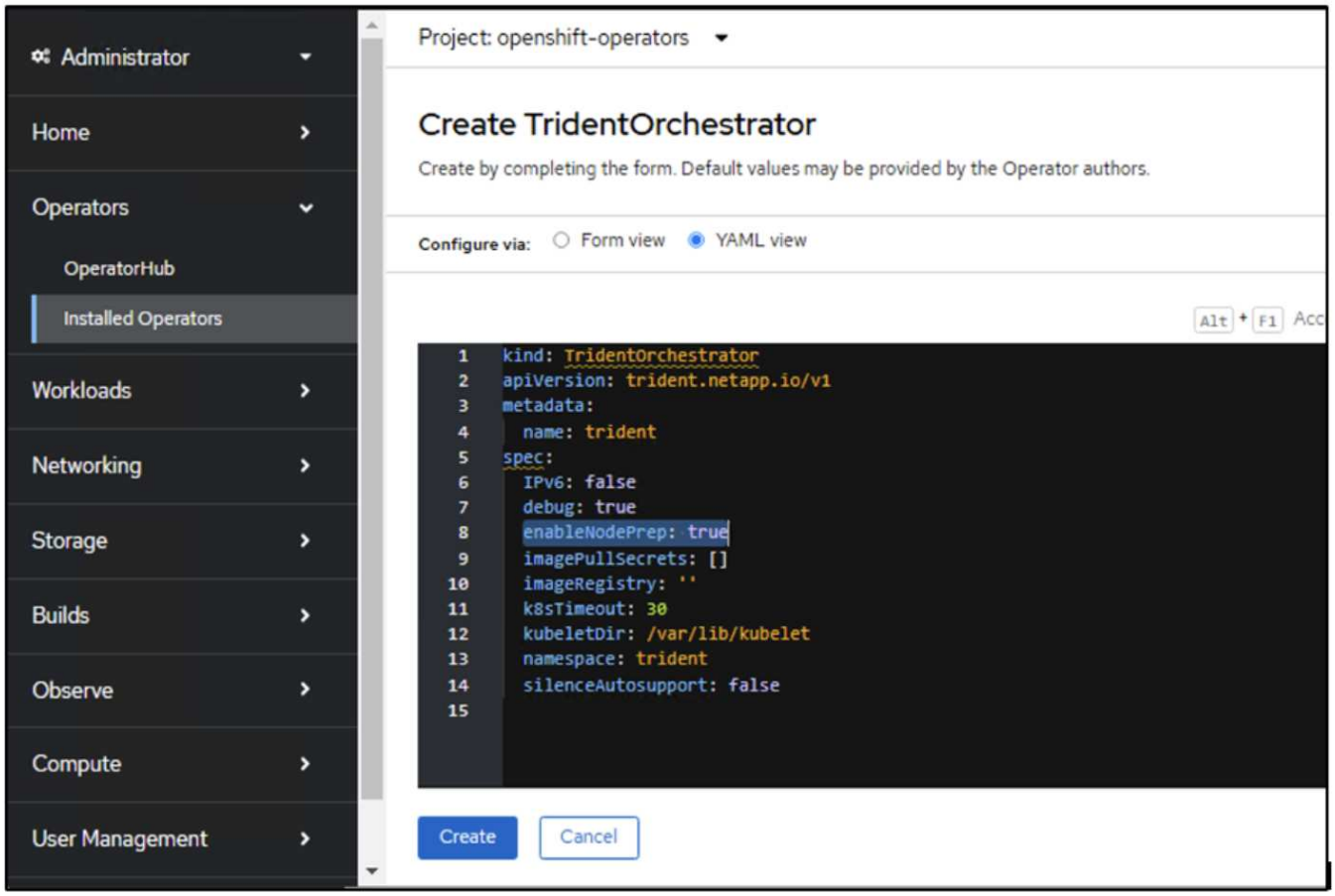

Administrator
Home
Operators
OperatorHub
Installed Operators
Workloads
Networking
Storage
Builds
Observe
Compute
User Management
Administration
Project: All Projects
OperatorHub
All Items
trident
NetApp Trident
provided by NetApp, Inc.
Trident Operator, to manage NetApp Trident installations
Certified
Community
NetApp Trident
25.2.0 provided by NetApp, Inc.
Install
Channel
stable
Version
25.2.0
Capability level
N/A
Source
Certified
Provider
NetApp, Inc.
Infrastructure features
Container Storage Interface
Disconnected
Repository
Container image
Created at
Mar 9, 2024, 7:00 PM
Support
NetApp
Activate Windows
Go to Settings to activate Windows.

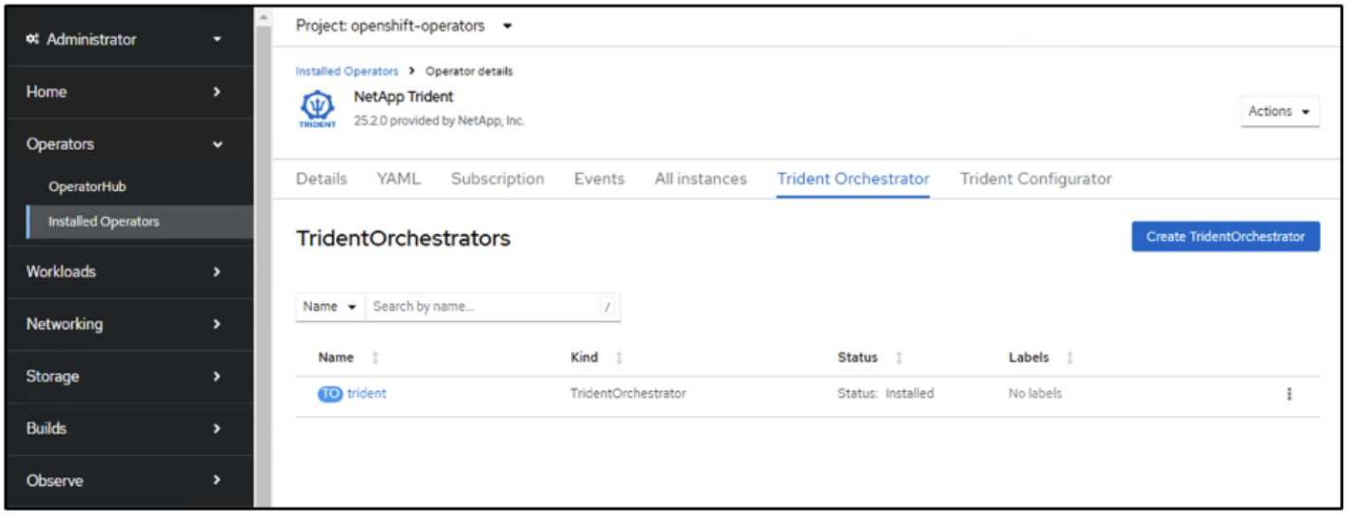

Red Hat
OpenShift Service on AWS
Administrator
Home
Operators
OperatorHub
Installed Operators
Workloads
Networking
Storage
Builds
Observe
Compute
User Management
Administration
OperatorHub > Operator Installation
Install Operator
Install your Operator by subscribing to one of the update channels to keep the Operator up to date. The strategy determines either manual or automatic updates.
Update channel *
stable
Version *
25.2.0
Installation mode *
All namespaces on the cluster (default)
Operator will be available in all Namespaces.
A specific namespace on the cluster
This mode is not supported by this Operator
Installed Namespace *
openshift-operators
Update approval *
Automatic
Manual
Install
Cancel
NetApp Trident
provided by NetApp, Inc.
Provided APIs
Trident Orchestrator
Used to deploy Astra Trident.
Trident Configurator
Automates AWS FSxN backend configuration



次に、 Trident Orchestrator インスタンスを作成します。 YAML ビューを使用してカスタム値を設定したり、インストール中に iscsi ノードの準備を有効にしたりします。








```
[root@localhost RedHat]# oc get pods -n trident
NAME                                  READY   STATUS    RESTARTS   AGE
trident-controller-86f89c855d-8w2jx   6/6     Running   0          38s
trident-node-linux-rnrnn              2/2     Running   0          38s
trident-node-linux-t9bxj              2/2     Running   0          38s
trident-node-linux-vqvl9              2/2     Running   0          38s
[root@localhost RedHat]#
```

上記のいずれかの方法でTridentをインストールすると、iscsidおよびmultipathdサービスを開始し、/etc/multipath.confファイルに次の設定を行うことで、ROSAクラスタワーカーノードがiSCSI用に準備されます。

```
sh-5.1#
sh-5.1# systemctl status iscsid
● iscsid.service - Open-iSCSI
     Loaded: loaded (/usr/lib/systemd/system/iscsid.service; enabled; preset: disabled)
     Active: active (running) since Fri 2025-03-21 18:28:13 UTC; 3 days ago
TriggeredBy: ● iscsid.socket
       Docs: man:iscsid(8)
             man:iscsiuio(8)
             man:iscsiadm(8)
   Main PID: 23224 (iscsid)
     Status: "Ready to process requests"
      Tasks: 1 (limit: 1649420)
     Memory: 3.2M
        CPU: 109ms
     CGroup: /system.slice/iscsid.service
             └─23224 /usr/sbin/iscsid -f
sh-5.1#
```

```
sh-5.1#
sh-5.1# systemctl status multipathd
● multipathd.service - Device-Mapper Multipath Device Controller
     Loaded: loaded (/usr/lib/systemd/system/multipathd.service; enabled; preset: enabled)
     Active: active (running) since Fri 2025-03-21 18:20:50 UTC; 3 days ago
TriggeredBy: ● multipathd.socket
   Main PID: 1565 (multipathd)
     Status: "up"
      Tasks: 7
     Memory: 62.4M
        CPU: 33min 51.363s
     CGroup: /system.slice/multipathd.service
             └─1565 /sbin/multipathd -d -s
```

```
sh-5.1#
sh-5.1# cat /etc/multipath.conf
defaults {
    find_multipaths     no
    user_friendly_names yes
}
blacklist {
}
blacklist_exceptions {
    device {
        vendor  NETAPP
        product LUN
    }
}
sh-5.1#
```

c.すべての**Trident**ポッドが実行状態であることを確認します

```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# oc get pods -n trident
NAME                                READY   STATUS    RESTARTS   AGE
trident-controller-f5f6796f-vd2sk   6/6     Running   0          19h
trident-node-linux-4svgz            2/2     Running   0          19h
trident-node-linux-dj9j4            2/2     Running   0          19h
trident-node-linux-jlshh            2/2     Running   0          19h
trident-node-linux-sqthw            2/2     Running   0          19h
trident-node-linux-ttj9c            2/2     Running   0          19h
trident-node-linux-vmjr5            2/2     Running   0          19h
trident-node-linux-wvqsf            2/2     Running   0          19h
trident-operator-545869857c-kgc7p   1/1     Running   0          19h
[root@localhost hcp-testing]#
```

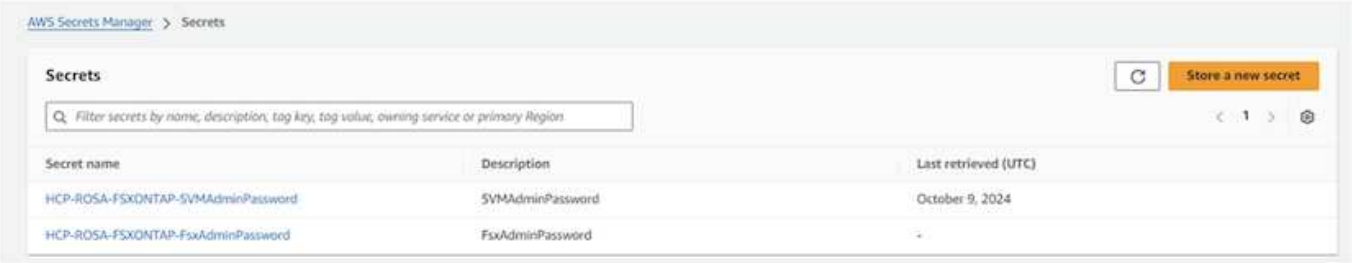
3. FSx for ONTAP (ONTAP NAS) を使用するように**Trident CSI** バックエンドを構成する

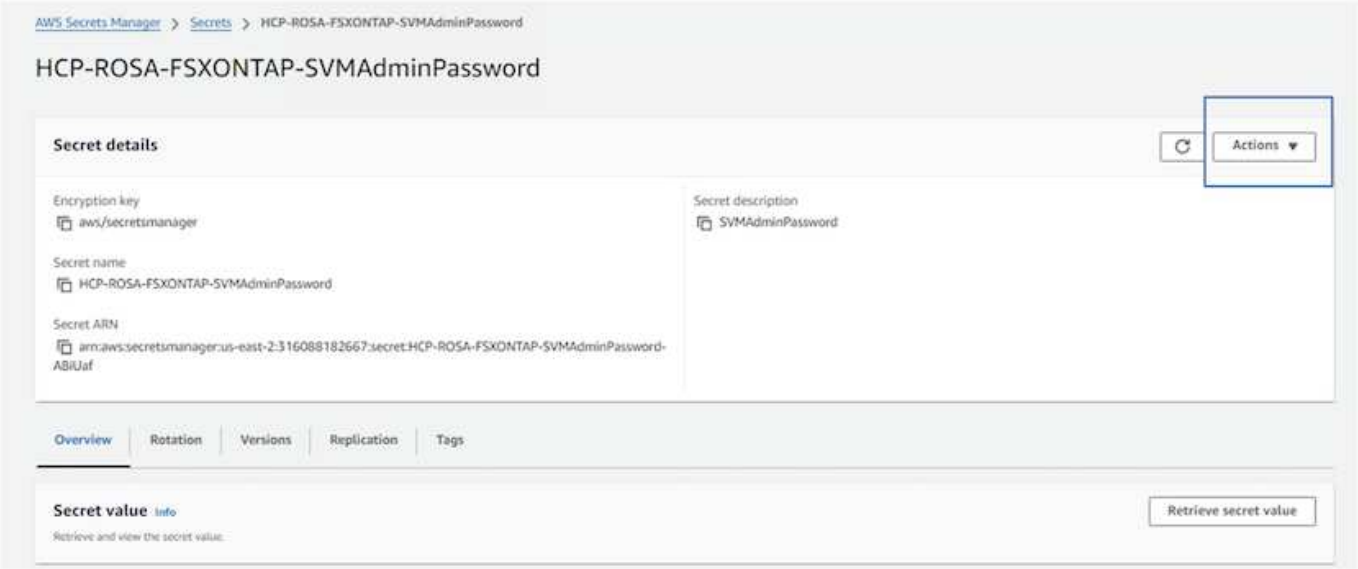
Tridentバックエンド構成は、 Trident にストレージ システム (この場合は FSx for ONTAP) と通信する方法を指示します。バックエンドを作成するには、クラスター管理および NFS データ インターフェイスとともに、接続するストレージ仮想マシンの資格情報を提供します。私たちは"ontap-nas ドライバー"FSx ファイル システムにストレージ ボリュームをプロビジョニングします。

a.まず、次の **yaml** を使用して **SVM** 認証情報のシークレットを作成します

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-fsx-ontap-nas-secret
  namespace: trident
type: Opaque
stringData:
  username: vsadmin
  password: <value provided for Define SVM password as a parameter to the
Cloud Formation Stack>
```

以下に示すように、AWS Secrets Manager から FSxN 用に作成された SVM パスワードを取得することもできます。





b.次に、次のコマンドを使用して、**SVM**認証情報のシークレットを**ROSA**クラスターに追加します

```
$ oc apply -f svm_secret.yaml
```

次のコマンドを使用して、シークレットがトライデント名前空間に追加されたことを確認できます。

```
$ oc get secrets -n trident |grep backend-fsx-ontap-nas-secret
```

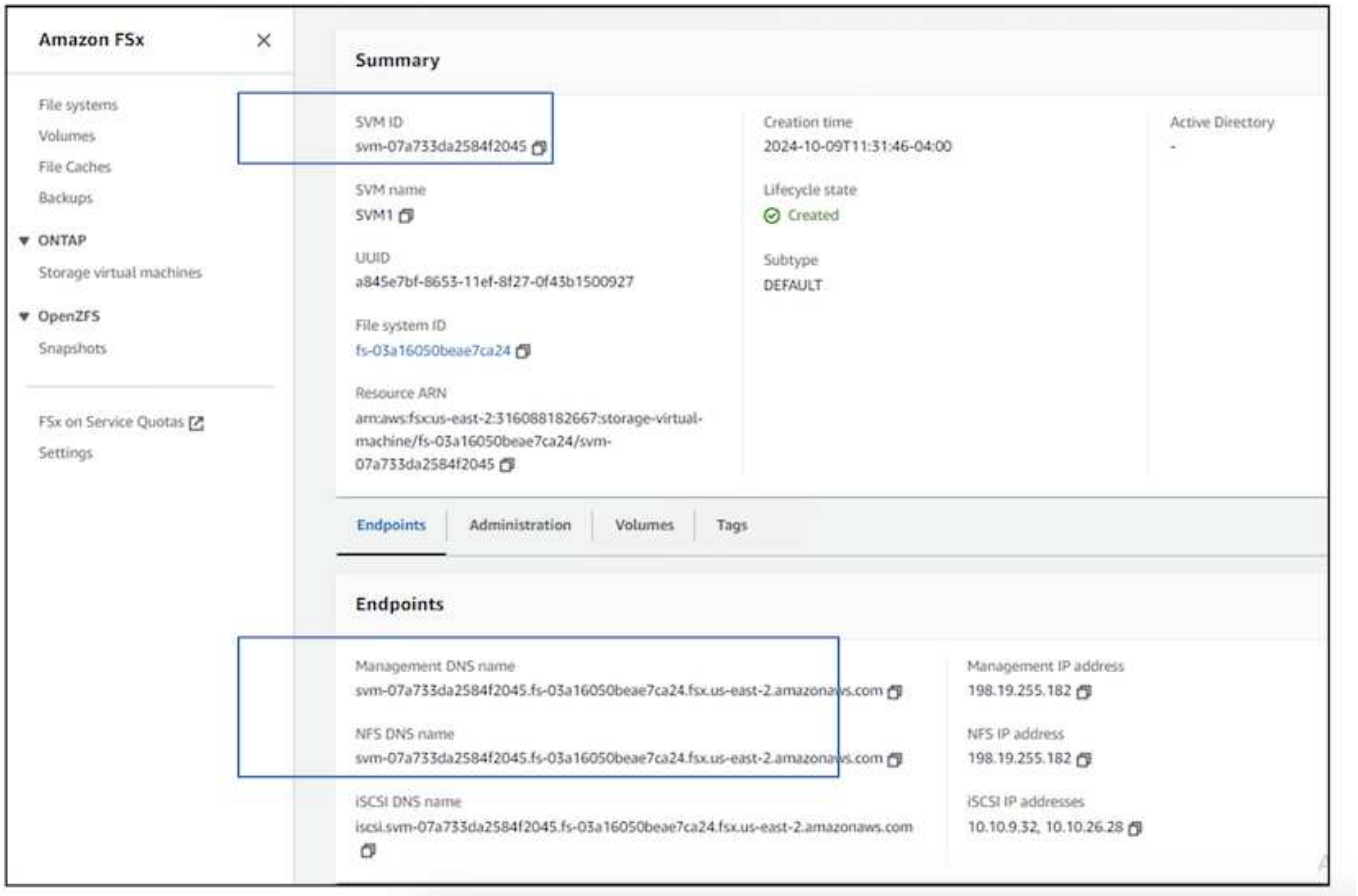
```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# oc get secrets -n trident | grep backend-fsx-ontap-nas-secret
backend-fsx-ontap-nas-secret         Opaque                  2      21h
[root@localhost hcp-testing]#
```

c.次に、バックエンド オブジェクトを作成します そのために、クローンした Git リポジトリの **fsx** ディレクトリに移動します。ファイル backend-ontap-nas.yaml を開きます。以下を置き換えます: **managementLIF** を管理 DNS 名に、**dataLIF** をAmazon FSx SVM の NFS DNS 名に、**svm** を SVM 名に置き換えます。次のコマンドを使用してバックエンド オブジェクトを作成します。

次のコマンドを使用してバックエンド オブジェクトを作成します。

```
$ oc apply -f backend-ontap-nas.yaml
```

以下のスクリーンショットに示すように、 Amazon FSxコンソールから管理 DNS 名、NFS DNS 名、および SVM 名を取得できます。



d.次に、次のコマンドを実行して、バックエンド オブジェクトが作成され、フェーズがバインドされ、ステータスが成功になっていることを確認します

```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# oc apply -f backend-ontap-nas.yaml
tridentbackendconfig.trident.netapp.io/backend-fsx-ontap-nas created
[root@localhost hcp-testing]# oc get tbc -n trident
NAME                  BACKEND NAME   BACKEND UUID                           PHASE   STATUS
backend-fsx-ontap-nas   fsx-ontap      acc65405-56be-4719-999d-27b448a50e29   Bound   Success
[root@localhost hcp-testing]#
```

4.ストレージ クラスの作成 Tridentバックエンドが構成されたので、バックエンドを使用するための Kubernetes ストレージ クラスを作成できます。ストレージ クラスは、クラスターで使用できるリソース オブジェクトです。アプリケーションに要求できるストレージの種類を説明および分類します。

a. fsx フォルダー内のファイル **storage-class-csi-nas.yaml** を確認します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: trident-csi
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: True
reclaimPolicy: Retain
```

b. ROSA クラスターにストレージ クラスを作成し、**trident-csi** ストレージ クラスが作成されたことを確認します。

```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# oc apply -f storage-class-csi-nas.yaml
storageclass.storage.k8s.io/trident-csi created
[root@localhost hcp-testing]# oc get sc
NAME              PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
gp2-csi           ebs.csi.aws.com         Delete          WaitForFirstConsumer   true                   2d16h
gp3-csi (default)   ebs.csi.aws.com         Delete          WaitForFirstConsumer   true                   2d16h
trident-csi       csi.trident.netapp.io   Retain          Immediate              true                   4s
[root@localhost hcp-testing]#
```

これにより、 Trident CSI ドライバーのインストールと FSx for ONTAPファイル システムへの接続が完了します。これで、FSx for ONTAPのファイルボリュームを使用して、サンプルの PostgreSQL ステートフル アプリケーションを ROSA にデプロイできるようになりました。

c. trident-csi ストレージ クラスを使用して作成された **PVC** および **PV** がないことを確認します。

d.アプリケーションが**Trident CSI** を使用して **PV** を作成できることを確認します。

fsx フォルダーに提供されている pvc-trident.yaml ファイルを使用して PVC を作成します。

```
pvc-trident.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: basic
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  storageClassName: trident-csi
```

```
 You can issue the following commands to create a pvc and verify that it
has been created.
image:redhat-openshift-container-rosa-011.png["Tridentを使用してテスト PVC
を作成する"]
```

iSCSI を使用するには、前述のようにワーカー ノードで iSCSI を有効にし、iSCSI バックエンドとストレージ クラスを作成する必要があります。以下に、サンプルの yaml ファイルをいくつか示します。

```
cat tbc.yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-san-secret
type: Opaque
stringData:
  username: fsxadmin
  password: <password for the fsxN filesystem>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  storageDriverName: ontap-san
  managementLIF: <management lif of fsxN filesystem>
  backendName: backend-tbc-ontap-san
  svm: svm_FSxNForROSAiSCSI
  credentials:
    name: backend-tbc-ontap-san-secret

cat sc.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: trident-csi
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
allowVolumeExpansion: true
```

5.サンプルの **PostgreSQL** ステートフル アプリケーションをデプロイする

a. helm を使用して **postgresql** をインストールします

```
$ helm install postgresql bitnami/postgresql -n postgresql --create
-namespace
```

```
[root@localhost hcp-testing]# helm install postgresql bitnami/postgresql -n postgresql --create-namespace
NAME: postgresql
LAST DEPLOYED: Mon Oct 14 06:52:58 2024
NAMESPACE: postgresql
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: postgresql
CHART VERSION: 15.5.21
APP VERSION: 16.4.0

** Please be patient while the chart is being deployed **

PostgreSQL can be accessed via port 5432 on the following DNS names from within your cluster:

    postgresql.postgresql.svc.cluster.local - Read/Write connection

To get the password for "postgres" run:

    export POSTGRES_PASSWORD=$(kubectl get secret --namespace postgresql postgresql -o jsonpath="{.data.postgres-password}" | base64 -d)

To connect to your database run the following command:

    kubectl run postgresql-client --rm --tty -i --restart='Never' --namespace postgresql --image docker.io/bitnami/postgresql:16.4.0-debian-12-r0 --
      --command -- psql --host postgresql -U postgres -d postgres -p 5432

    > NOTE: If you access the container using bash, make sure that you execute "/opt/bitnami/scripts/postgresql/entrypoint.sh /bin/bash" in order to
 1001) does not exist"

To connect to your database from outside the cluster execute the following commands:

    kubectl port-forward --namespace postgresql svc/postgresql 5432:5432 &
    PGPASSWORD="$POSTGRES_PASSWORD" psql --host 127.0.0.1 -U postgres -d postgres -p 5432

WARNING: The configured password will be ignored on new installation in case when previous PostgreSQL release was deleted through the helm command.
sword, and setting it through helm won't take effect. Deleting persistent volumes (PVs) will solve the issue.
```

b.アプリケーション ポッドが実行中であり、アプリケーションに対して **PVC** と **PV** が作成されていることを確認します。

```
[root@localhost hcp-testing]# oc get pods -n postgresql
NAME           READY   STATUS    RESTARTS   AGE
postgresql-0   1/1     Running   0          29m
```

```
[root@localhost hcp-testing]# oc get pvc -n postgresql
NAME                STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
data-postgresql-0   Bound    pvc-e3ddd9bd-e6a7-4a4a-b935-f1c090fd8db6   8Gi        RWO            trident-csi
```

```
[root@localhost hcp-testing]# oc get pv | grep postgresql
pvc-e3ddd9bd-e6a7-4a4a-b935-f1c090fd8db6   8Gi        RWO            Retain           Bound    postgresql/data-postgresql-0
csi    <unset>                          4h20m
[root@localhost hcp-testing]#
```

c. PostgreSQLクライアントをデプロイする

インストールされた **postgresql** サーバーのパスワードを取得するには、次のコマンドを使用します。

```
$ export POSTGRES_PASSWORD=$(kubectl get secret --namespace postgresql
postgresql -o jsoata.postgres-password}" | base64 -d)
```

以下のコマンドを使用して **postgresql** クライアントを実行し、パスワードを使用してサーバーに接続します

```
$ kubectl run postgresql-client --rm --tty -i --restart='Never'
--namespace postgresql --image docker.io/bitnami/postgresql:16.2.0-debian-
11-r1 --env="PGPASSWORD=$POSTGRES_PASSWORD" \
> --command -- psql --host postgresql -U postgres -d postgres -p 5432
```

```
[root@localhost hcp-testing]# kubectl run postgresql-client --rm --tty -i --restart='Never' --namespace postgresql --image docker.io/bitna
$POSTGRES_PASSWORD" \
> --command -- psql --host postgresql -U postgres -d postgres -p 5432
Warning: would violate PodSecurity "restricted:v1.24": allowPrivilegeEscalation != false (container "postgresql-client" must set securityC
 capabilities (container "postgresql-client" must set securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod or container "
Root=true), seccompProfile (pod or container "postgresql-client" must set securityContext.seccompProfile.type to "RuntimeDefault" or "Loca
If you don't see a command prompt, try pressing enter.
```

d.データベースとテーブルを作成します。テーブルのスキーマを作成し、テーブルに **2** 行のデータを挿入します。

```
postgres=# CREATE DATABASE erp;
CREATE DATABASE
postgres=# \c erp
psql (16.2, server 16.4)
You are now connected to database "erp" as user "postgres".
erp=# CREATE TABLE PERSONS(ID INT PRIMARY KEY NOT NULL, FIRSTNAME TEXT NOT NULL, LASTNAME TEXT NOT NULL);
CREATE TABLE
erp=# INSERT INTO PERSONS VALUES(1,'John','Doe');
INSERT 0 1
erp=# \dt
         List of relations
 Schema |  Name   | Type  |  Owner
--------+---------+-------+----------
 public | persons | table | postgres
(1 row)
```

```
erp=# SELECT * FROM PERSONS;
 id | firstname | lastname
----+-----------+----------
  1 | John      | Doe
(1 row)
```

```
erp=# INSERT INTO PERSONS VALUES(2,'Jane','Scott');
INSERT 0 1
erp=# SELECT * from PERSONS;
 id | firstname | lastname
----+-----------+----------
  1 | John      | Doe
  2 | Jane      | Scott
(2 rows)
```

NetApp ONTAPを使用した AWS 上の Red Hat OpenShift サービス

このドキュメントでは、 NetApp ONTAP をRed Hat OpenShift Service on AWS (ROSA) で使用する方法について概説します。

ボリュームスナップショットの作成

1.アプリ ボリュームのスナップショットを作成する このセクションでは、アプリに関連付けられたボリュームのトライデント スナップショットを作成する方法を説明します。これは、アプリ データの特定時点のコピーになります。アプリケーション データが失われた場合、この時点のコピーからデータを回復できます。注: このスナップショットは、 ONTAP(オンプレミスまたはクラウド) の元のボリュームと同じアグリゲートに保存されます。したがって、 ONTAPストレージ アグリゲートが失われた場合、そのスナップショットからアプリケーション データを回復することはできません。

**a. VolumeSnapshotClass を作成し、次のマニフェストを volume-snapshot-class.yaml というファイルに保存します。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
 name: fsx-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

上記のマニフェストを使用してスナップショットを作成します。

```
[root@localhost hcp-testing]# oc create -f volume-snapshot-class.yaml
volumesnapshotclass.snapshot.storage.k8s.io/fsx-snapclass created
[root@localhost hcp-testing]#
```

b.次に、スナップショットを作成します VolumeSnapshot を作成して、Postgresql データのポイントインタイム コピーを取得し、既存の PVC のスナップショットを作成します。これにより、ファイルシステムのバックエンドでほとんどスペースを占有しない FSx スナップショットが作成されます。次のマニフェストを volume-snapshot.yaml というファイルに保存します。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
 name: postgresql-volume-snap-01
spec:
 volumeSnapshotClassName: fsx-snapclass
 source:
   persistentVolumeClaimName: data-postgresql-0
```

c.ボリュームスナップショットを作成し、作成されたことを確認します

データベースを削除して、データの損失をシミュレートします (データの損失はさまざまな理由で発生する可能性がありますが、ここではデータベースを削除することでシミュレートしています)

```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# oc create -f postgresql-volume-snapchot.yaml -n postgresql
volumesnapshot.snapshot.storage.k8s.io/postgresql-volume-snap-01 created
[root@localhost hcp-testing]# oc get VolumeSnapShot -n postgresql
NAME                       READYTOUSE   SOURCEPVC          SOURCESNAPSHOTCONTENT   RESTORESIZE   SNAPSHOTCLASS   SNAPSHOTCONTENT
postgresql-volume-snap-01   true        data-postgresql-0                          41500Ki       fsx-snapclass   snapcontent-5baf4337-922e-4318-be82-6db822082339
[root@localhost hcp-testing]#
```

d.データベースを削除して、データの損失をシミュレートします **(**データの損失はさまざまな理由で発生する可能性がありますが、ここではデータベースを削除することでシミュレートしています**)**

```
postgres=# \c erp;
psql (16.2, server 16.4)
You are now connected to database "erp" as user "postgres".
erp=# SELECT * FROM persons;
 id | firstname | lastname
----+-----------+----------
  1 | John      | Doe
  2 | Jane      | Scott
(2 rows)
```

```
postgres=# DROP DATABASE erp;
DROP DATABASE
postgres=# \c erp;
connection to server at "postgresql" (172.30.103.67), port 5432 failed: FATAL:  database "erp" does not exist
Previous connection kept
postgres=#
```

ボリュームスナップショットからの復元

1.スナップショットからの復元 このセクションでは、アプリ ボリュームの Trident スナップショットからアプリケーションを復元する方法を説明します。

a.スナップショットからボリュームのクローンを作成する

ボリュームを以前の状態に復元するには、取得したスナップショットのデータに基づいて新しい PVC を作成する必要があります。これを行うには、次のマニフェストをpvc-clone.yamlというファイルに保存します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: postgresql-volume-clone
spec:
 accessModes:
   - ReadWriteOnce
 storageClassName: trident-csi
 resources:
   requests:
     storage: 8Gi
 dataSource:
   name: postgresql-volume-snap-01
   kind: VolumeSnapshot
   apiGroup: snapshot.storage.k8s.io
```

上記のマニフェストを使用して、スナップショットをソースとして PVC を作成し、ボリュームのクローンを作成します。マニフェストを適用し、クローンが作成されたことを確認します。

```
[root@localhost hcp-testing]# oc create -f postgresql-pvc-clone.yaml -n postgresql
persistentvolumeclaim/postgresql-volume-clone created
[root@localhost hcp-testing]# oc get pvc -n postgresql
NAME                     STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
data-postgresql-0        Bound    pvc-e3ddd9bd-e6a7-4a4a-b935-f1c090fd8db6   8Gi        RWO            trident-csi
postgresql-volume-clone  Bound    pvc-b38fbc54-55dc-47e8-934d-47f181fddac6   8Gi        RWO            trident-csi
[root@localhost hcp-testing]#
```

b.元の **postgresql** インストールを削除します

```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# helm uninstall postgresql -n postgresql
release "postgresql" uninstalled
[root@localhost hcp-testing]# oc get pods -n postgresql
No resources found in postgresql namespace.
[root@localhost hcp-testing]#
```

c.新しいクローン **PVC** を使用して新しい **postgresql** アプリケーションを作成します

```
$ helm install postgresql bitnami/postgresql --set
primary.persistence.enabled=true --set
primary.persistence.existingClaim=postgresql-volume-clone -n postgresql
```

```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# helm install postgresql bitnami/postgresql --set primary.persistence.enabled=true \
> --set primary.persistence.existingClaim=postgresql-volume-clone -n postgresql
NAME: postgresql
LAST DEPLOYED: Mon Oct 14 12:03:31 2024
NAMESPACE: postgresql
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: postgresql
CHART VERSION: 15.5.21
APP VERSION: 16.4.0

** Please be patient while the chart is being deployed **

PostgreSQL can be accessed via port 5432 on the following DNS names from within your cluster:

    postgresql.postgresql.svc.cluster.local - Read/Write connection

To get the password for "postgres" run:

    export POSTGRES_PASSWORD=$(kubectl get secret --namespace postgresql postgresql -o jsonpath="{.data.postgres-password}" | ba

To connect to your database run the following command:

    kubectl run postgresql-client --rm --tty -i --restart='Never' --namespace postgresql --image docker.io/bitnami/postgresql:16
      --command -- psql --host postgresql -U postgres -d postgres -p 5432

    > NOTE: If you access the container using bash, make sure that you execute "/opt/bitnami/scripts/postgresql/entrypoint.sh /b
 1001} does not exist"

To connect to your database from outside the cluster execute the following commands:

    kubectl port-forward --namespace postgresql svc/postgresql 5432:5432 &
    PGPASSWORD="$POSTGRES_PASSWORD" psql --host 127.0.0.1 -U postgres -d postgres -p 5432

WARNING: The configured password will be ignored on new installation in case when previous PostgreSQL release was deleted throug
sword, and setting it through helm won't take effect. Deleting persistent volumes (PVs) will solve the issue.

WARNING: There are "resources" sections in the chart not set. Using "resourcesPreset" is not recommended for production. For pro
ng to your workload needs:
  - primary.resources
  - readReplicas.resources
+info https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
[root@localhost hcp-testing]#
```

d.アプリケーションポッドが実行状態であることを確認します

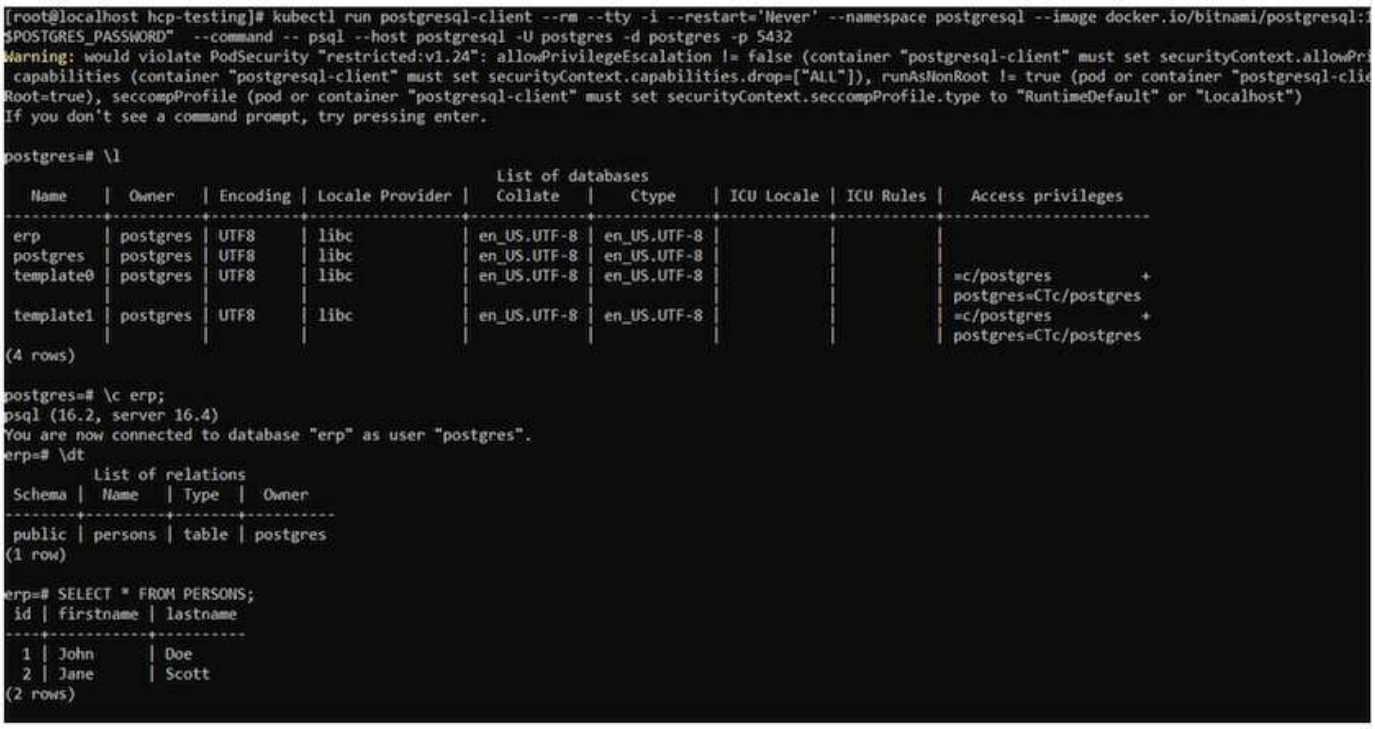
```
[root@localhost hcp-testing]# oc get pods -n postgresql
NAME           READY   STATUS    RESTARTS   AGE
postgresql-0   1/1     Running   0          2m1s
[root@localhost hcp-testing]#
```

e.ポッドがクローンを **PVC** として使用していることを確認します

```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# oc describe pod/postgresql-0 -n postgresql
```

```
  ContainersReady             True
  PodScheduled                True
Volumes:
  empty-dir:
    Type:       EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:
    SizeLimit:  <unset>
  dshm:
    Type:       EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:     Memory
    SizeLimit:  <unset>
  data:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  postgresql-volume-clone
    ReadOnly:   false
QoS Class:      Burstable
Node-Selectors: <none>
Tolerations:    node.kubernetes.io/memory-pressure:NoSchedule op=Exists
                node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason                  Age    From                     Message
  ----    ------                  ----   ----                     -------
  Normal  Scheduled               3m55s  default-scheduler        Successfully assigned postgresql/postgres
.us-east-2.compute.internal
  Normal  SuccessfulAttachVolume  3m54s  attachdetach-controller  AttachVolume.Attach succeeded for volume
8-934d-47f181fddac6"
  Normal  AddedInterface          3m43s  multus                   Add eth0 [10.129.2.126/23] from ovn-kuber
  Normal  Pulled                  3m43s  kubelet                  Container image "docker.io/bitnami/postgr
r0" already present on machine
  Normal  Created                 3m42s  kubelet                  Created container postgresql
  Normal  Started                 3m42s  kubelet                  Started container postgresql
[root@localhost hcp-testing]#
```

f) データベースが期待通りに復元されたことを確認するには、コンテナコンソールに戻り、既存のデータベースを表示します。



デモビデオ

ホスト型コントロールプレーンを使用した AWS 上の Red Hat OpenShift サービスとAmazon FSx for NetApp ONTAP

Red Hat OpenShiftとOpenShiftソリューションに関するその他のビデオは以下をご覧ください。"ここをクリックしてください。" 。

著作権に関する情報

商標に関する情報