使用 FSxN 在 AWS 上提供 Red Hat OpenShift 服务

NetApp container solutions

NetApp
January 21, 2026

目录

使用 FSxN 在 AWS 上提供 Red Hat OpenShift 服务

借助NetApp ONTAP在 AWS 上提供 Red Hat OpenShift 服务

概述

在本节中，我们将展示如何利用 FSx for ONTAP作为在 ROSA 上运行的应用程序的持久存储层。它将展示在 ROSA 集群上安装NetApp Trident CSI 驱动程序、为ONTAP文件系统配置 FSx 以及部署示例有状态应用程序。它还将显示备份和恢复应用程序数据的策略。通过此集成解决方案，您可以建立一个可轻松跨可用区扩展的共享存储框架，从而简化使用Trident CSI 驱动程序扩展、保护和恢复数据的过程。

前提条件

- "AWS 账户"
- "Red Hat 帐户"
- IAM 用户"具有适当的权限"创建并访问ROSA集群
- "AWS CLI"
- "ROSA CLI"
- "OpenShift 命令行界面"（oc）
- Helm 3"文档"
- "HCP ROSA 集群"
- "访问 Red Hat OpenShift Web 控制台"

该图显示了部署在多个 AZ 中的 ROSA 集群。 ROSA集群的主节点、基础设施节点位于Red Hat的VPC中，而工作节点位于客户账户中的VPC中。我们将在同一个 VPC 内创建一个 FSx for ONTAP文件系统，并在 ROSA 集群中安装Trident驱动程序，以允许该 VPC 的所有子网连接到该文件系统。

初始设置

1.为**NetApp ONTAP**配置 **FSx**

在与 ROSA 集群相同的 VPC 中为NetApp ONTAP创建多可用区 FSx。可通过多种方法实现此操作。提供了使用 CloudFormation Stack 创建 FSxN 的详细信息

a.克隆 **GitHub** 存储库

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

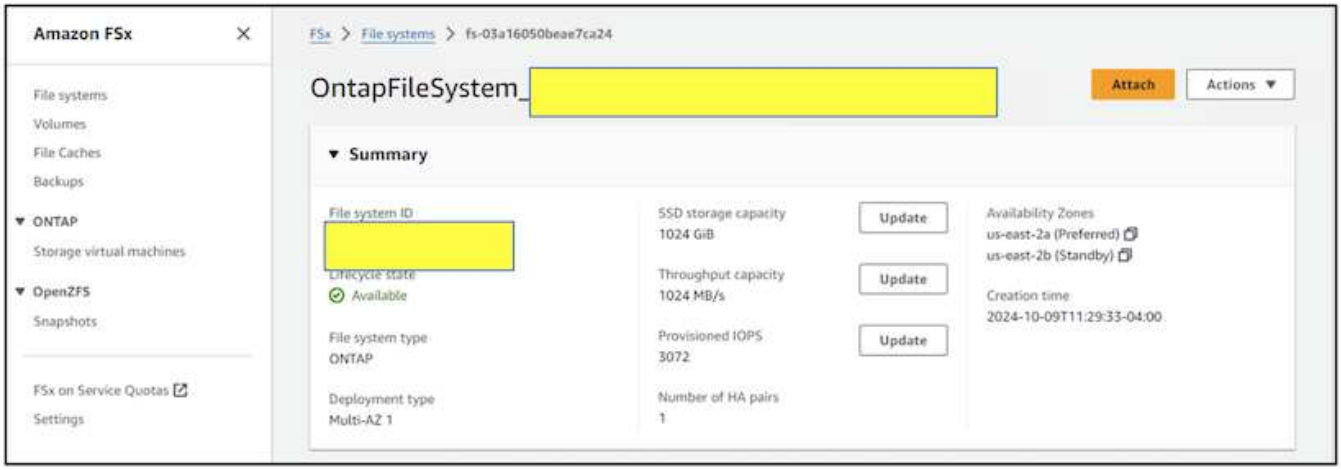
b.运行 **CloudFormation** 堆栈 通过将参数值替换为您自己的值来运行以下命令：

```
$ 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
```

其中： region-name：与部署 ROSA 集群的区域相同 subnet1_ID：FSxN 的首选子网的 id subnet2_ID：FSxN 的备用子网的 id VPC_ID：部署 ROSA 集群的 VPC 的 id routetable1_ID、routetable2_ID：与上面选择的子网关联的路由表的 id your_allowed_CIDR：FSx for ONTAP安全组入口规则允许的 CIDR 范围，用于控制访问。您可以使用 0.0.0.0/0 或任何适当的 CIDR 来允许所有流量访问 FSx for ONTAP的特定端口。定义管理员密码：登录 FSxN 的密码 定义 SVM 密码：登录将要创建的 SVM 的密码。

使用Amazon FSx控制台验证您的文件系统和存储虚拟机 (SVM) 是否已创建，如下所示：



2.为 **ROSA** 集群安装并配置**Trident CSI** 驱动程序

b.安装**Trident**

ROSA 集群工作节点预先配置了 nfs 工具，使您能够使用 NAS 协议进行存储配置和访问。

如果您想使用 iSCSI，则需要为 iSCSI 准备工作节点。从Trident 25.02 版本开始，您可以轻松准备 ROSA 集群（或任何 OpenShift 集群）的工作节点以在 FSxN 存储上执行 iSCSI 操作。有两种简单的方法可以安装Trident 25.02（或更高版本），以自动为 iSCSI 准备工作节点。 1. 使用 tridentctl 工具从命令行使用 node-prep-flag。2. 使用来自操作员中心的 Red Hat 认证的Trident操作员并对其进行自定义。 3.使用 Helm。

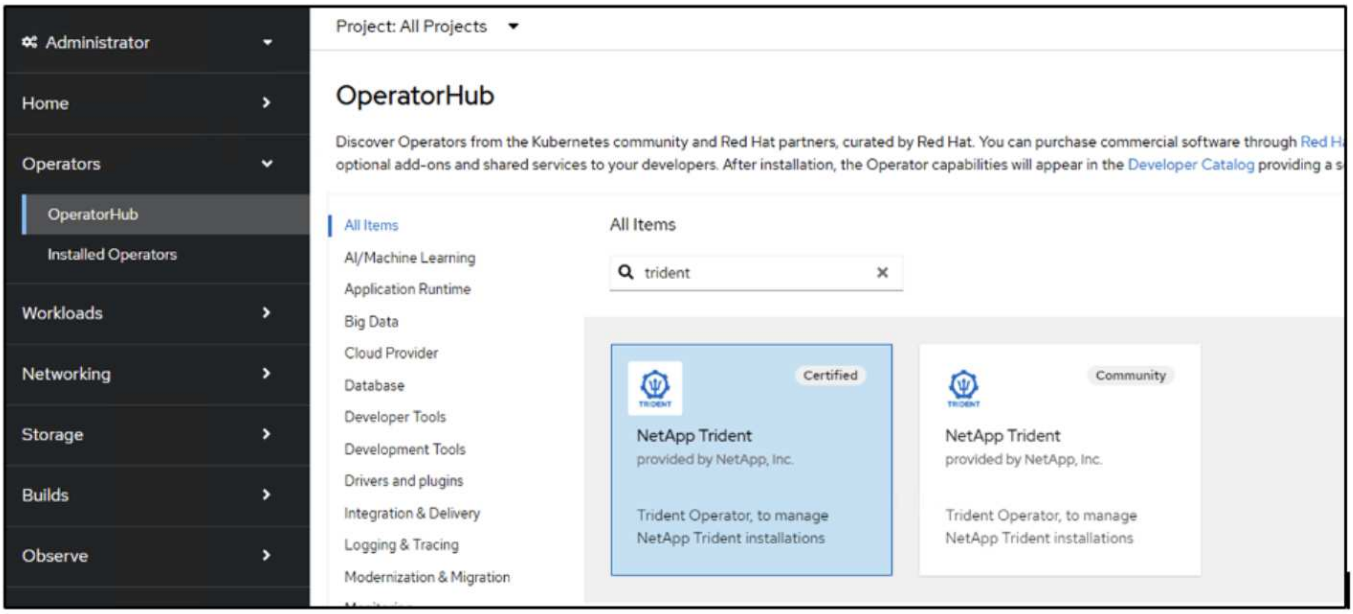
使用上述任何一种方法而不启用节点准备将允许您仅使用 NAS 协议在 FSxN 上配置存储。

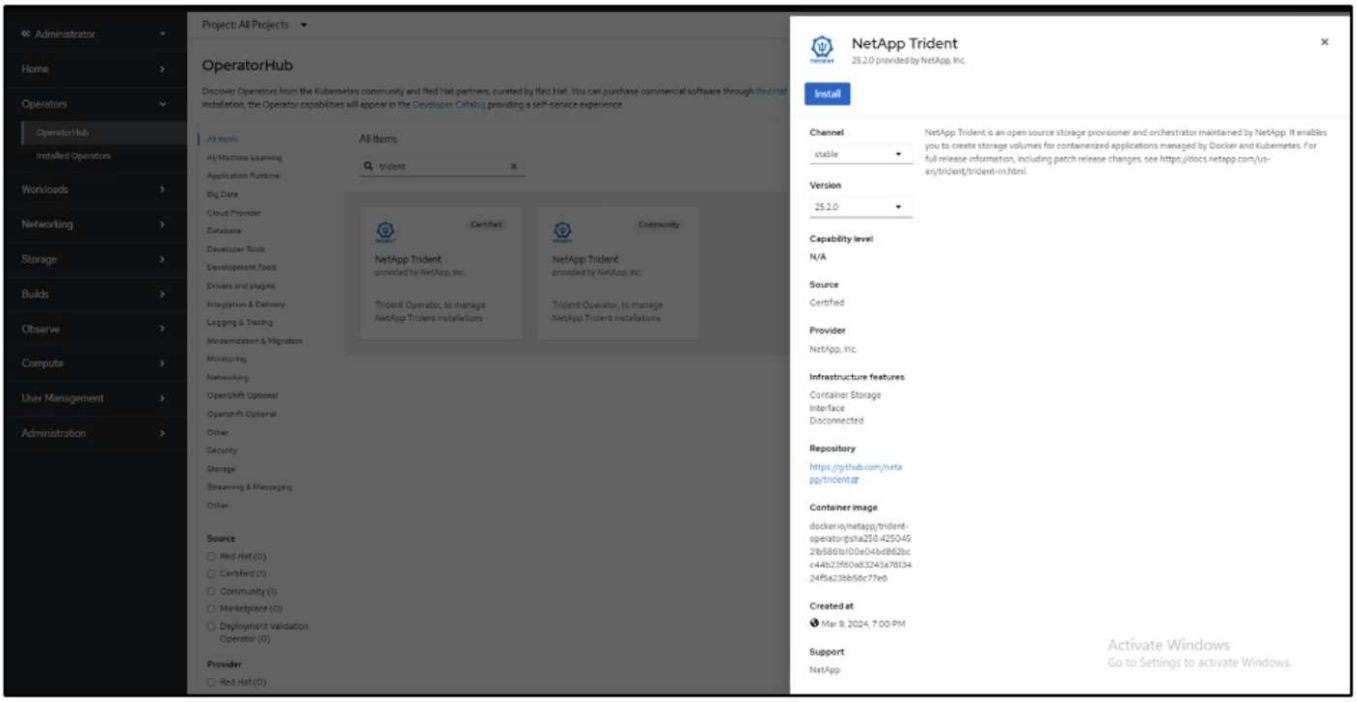
方法一：使用 **tridentctl** 工具

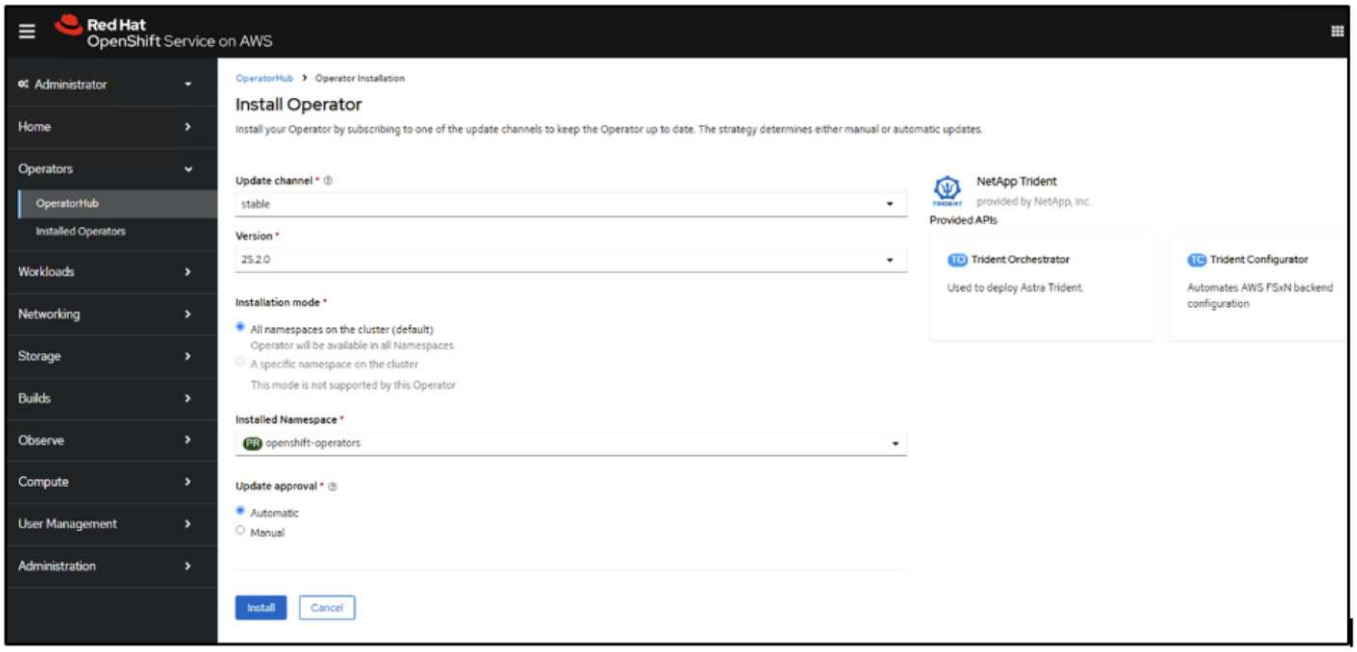
使用 node-prep 标志并安装Trident ，如图所示。在发出安装命令之前，您应该已经下载了安装程序包。请参阅"此处的文档" 。

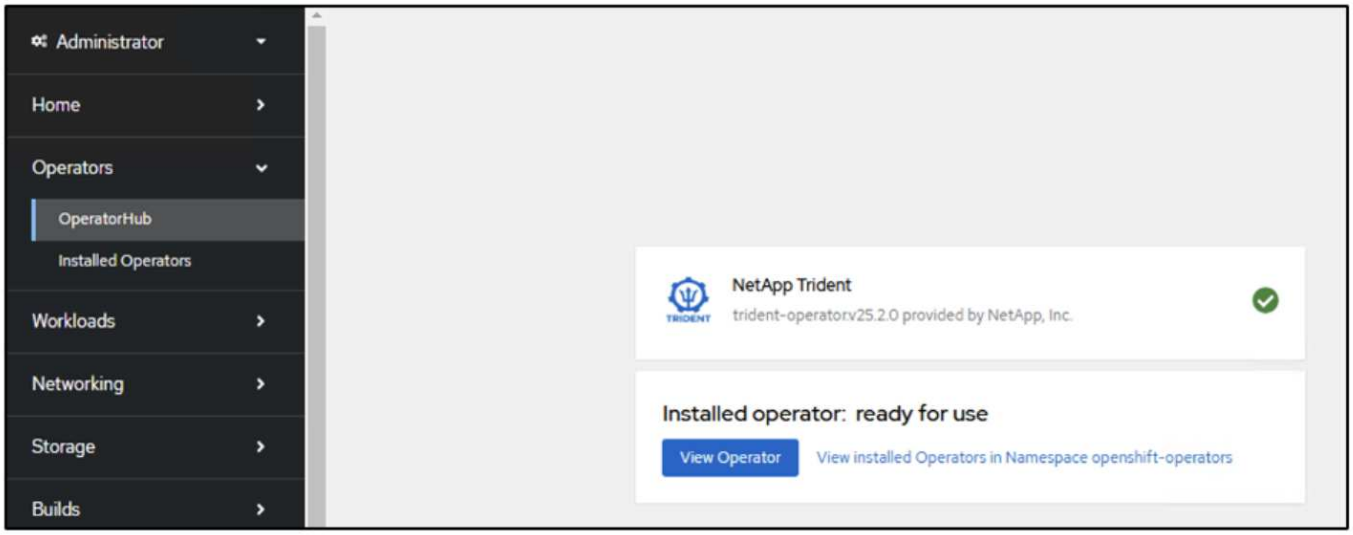
```
#./tridentctl install trident -n trident --node-prep=iscsi
```

方法 **2**：使用 **Red Hat** 认证的**Trident Operator** 并进行自定义 从 OperatorHub 中找到 Red Hat 认证的Trident Operator 并安装它。

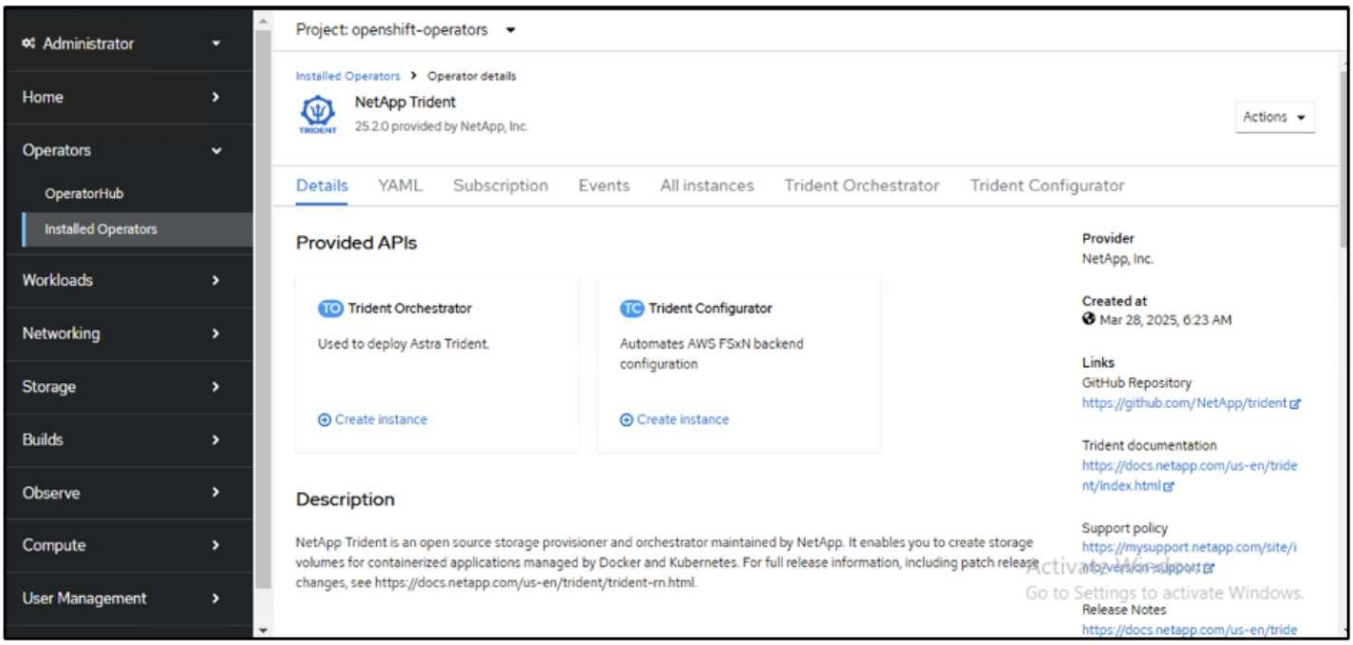

Administrator
Home
Operators
OperatorHub
Installed Operators
Workloads
Networking
Storage
Builds
Observe
Project: All Projects
OperatorHub
Discover Operators from the Kubernetes community and Red Hat partners, curated by Red Hat. You can purchase commercial software through Red H
optional add-ons and shared services to your developers. After installation, the Operator capabilities will appear in the Developer Catalog providing a s
All Items
AI/Machine Learning
Application Runtime
Big Data
Cloud Provider
Database
Developer Tools
Development Tools
Drivers and plugins
Integration & Delivery
Logging & Tracing
Modernization & Migration
All Items
trident
Certified
NetApp Trident
provided by NetApp, Inc.
Trident Operator, to manage NetApp Trident installations
Community
NetApp Trident
provided by NetApp, Inc.
Trident Operator, to manage NetApp Trident installations

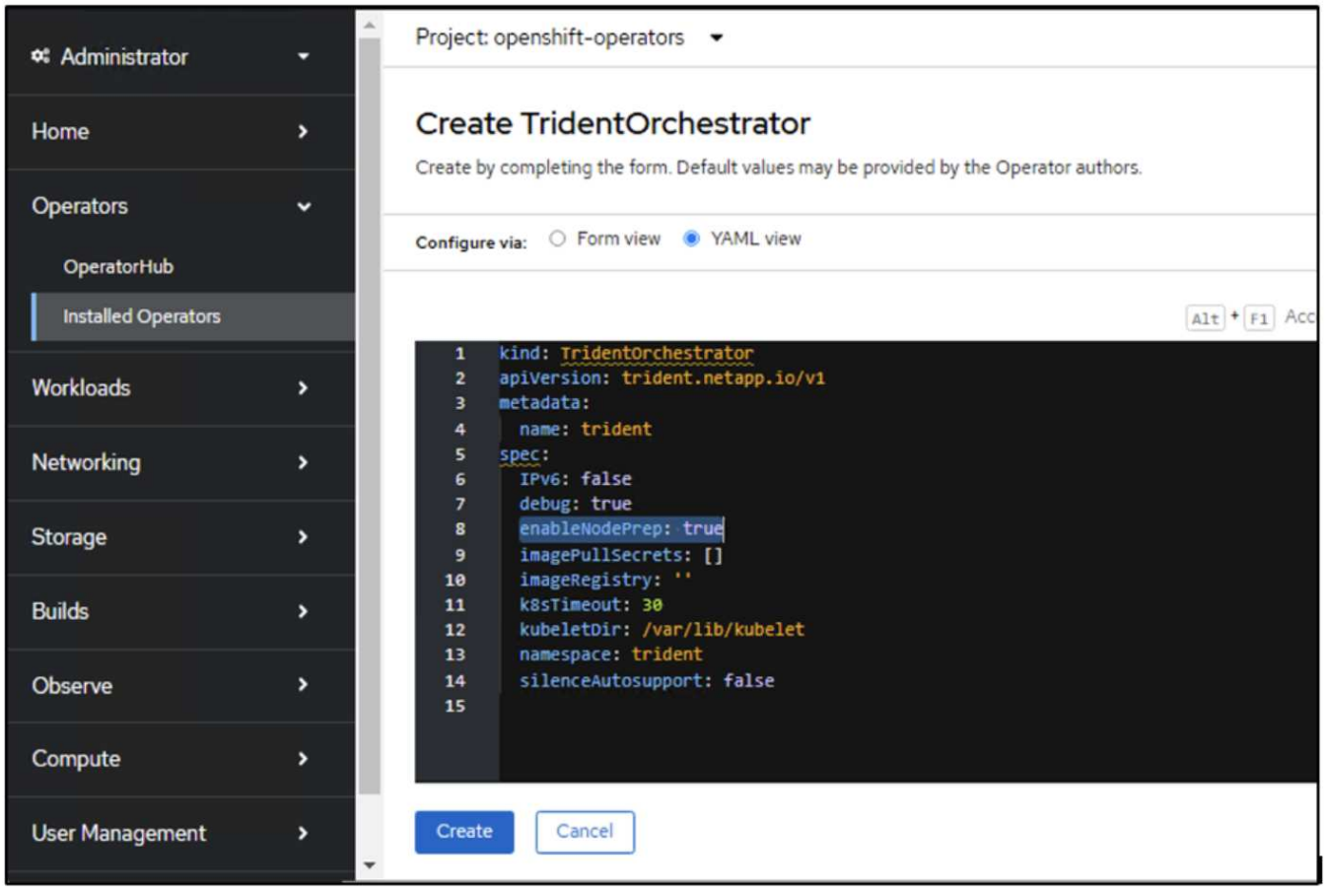

NetApp Trident
Install
Channel
stable
Version
25.2.0
Capability level
N/A
Source
Certified
Provider
NetApp, Inc.
Infrastructure features
Container Storage Interface
Disconnected
Repository
Container image
Created at
Mar 9, 2024, 7:00 PM
Support
NetApp
Activate Windows

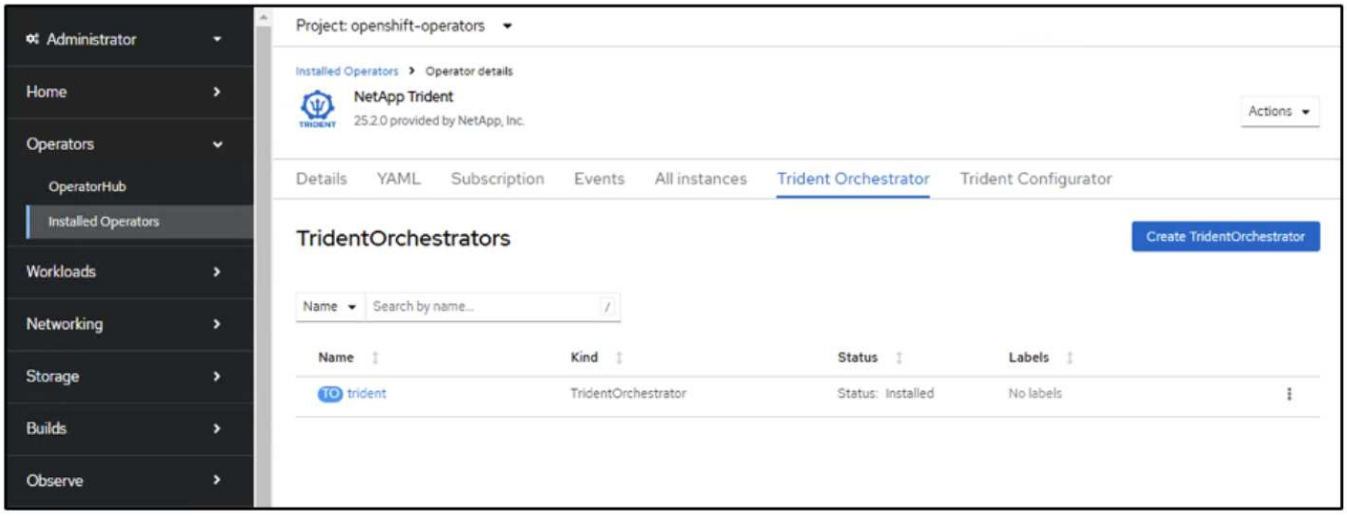




接下来，创建Trident Orchestrator 实例。使用 YAML 视图设置任何自定义值或在安装期间启用 iscsi 节点准备。


Administrator
Home
Operators
OperatorHub
Installed Operators
Workloads
Networking
Storage
Builds
Observe
Compute
User Management
Project: openshift-operators
Installed Operators > Operator details
NetApp Trident
25.2.0 provided by NetApp, Inc.
Actions
Details
YAML
Subscription
Events
All instances
Trident Orchestrator
Trident Configurator
Provided APIs
Trident Orchestrator
Used to deploy Astra Trident.
Create instance
Trident Configurator
Automates AWS FSxN backend configuration
Create instance
Provider
NetApp, Inc.
Created at
Mar 28, 2025, 6:23 AM
Links
GitHub Repository
https://github.com/NetApp/trident
Trident documentation
https://docs.netapp.com/us-en/trident/index.html
Support policy
Release Notes
Description
NetApp Trident is an open source storage provisioner and orchestrator maintained by NetApp. It enables you to create storage volumes for containerized applications managed by Docker and Kubernetes. For full release information, including patch release changes, see https://docs.netapp.com/us-en/trident/trident-rn.html.


Administrator
Home
Operators
OperatorHub
Installed Operators
Workloads
Networking
Storage
Builds
Observe
Compute
User Management
Project: openshift-operators
Create TridentOrchestrator
Create by completing the form. Default values may be provided by the Operator authors.
Configure via:
Form view
YAML view
Alt + F1
kind: TridentOrchestrator
apiVersion: trident.netapp.io/v1
metadata:
name: trident
spec:
IPv6: false
debug: true
enableNodePrep: true
imagePullSecrets: []
imageRegistry: ''
k8sTimeout: 30
kubeletDir: /var/lib/kubelet
namespace: trident
silenceAutosupport: false
Create
Cancel




```
[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]#
```

使用上述任何一种方法安装Trident都将通过启动 iscsid 和 multipathd 服务并在 /etc/multipath.conf 文件中设置以下内容，为 iSCSI 准备 ROSA 集群工作节点

```
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.验证所有**Trident pod** 是否处于运行状态

```
[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]#
```

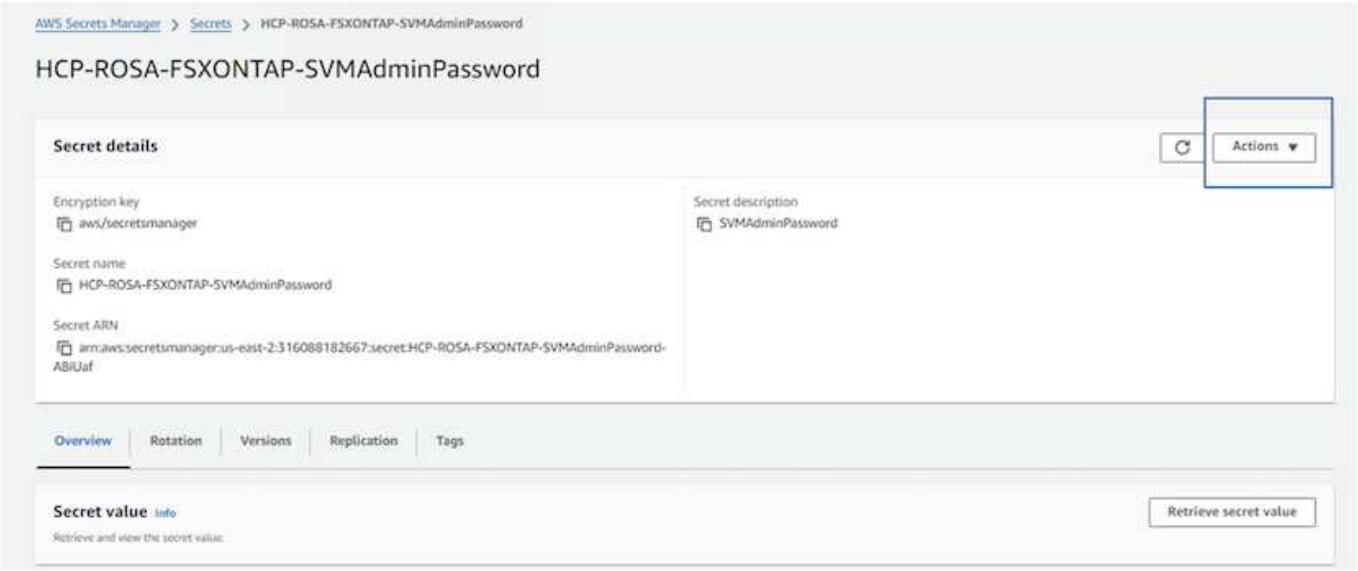
3.配置Trident CSI 后端以使用 FSx for ONTAP (ONTAP NAS)

Trident后端配置告诉Trident如何与存储系统（在本例中为 FSx for ONTAP）通信。为了创建后端，我们将提供要连接的存储虚拟机的凭据，以及集群管理和 NFS 数据接口。我们将使用"ontap-nas驱动程序"在 FSx 文件系统中配置存储卷。

一个。首先，使用以下 **yaml** 为 SVM 凭证创建密钥

```
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>
```

您还可以从 AWS Secrets Manager 检索为 FSxN 创建的 SVM 密码，如下所示。



b.接下来，使用以下命令将 **SVM** 凭证的密钥添加到 **ROSA** 集群

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

您可以使用以下命令验证该秘密是否已添加到 trident 命名空间中

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

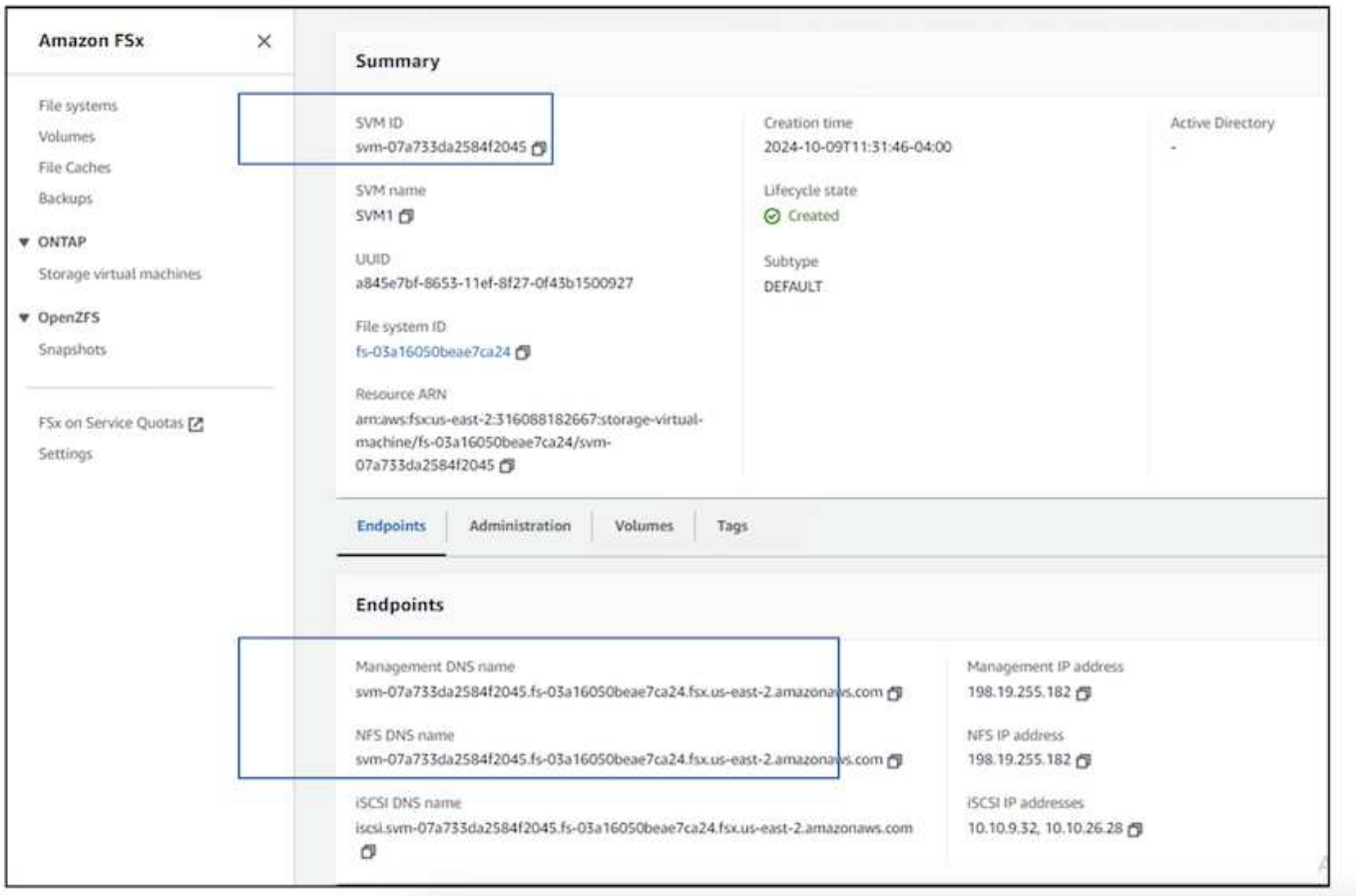
```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# oc get secrets -n trident | grep backend-fsx-ontap-nas-secret
backend-fsx-ontap-nas-secret         Opaque                  2      21h
[root@localhost hcp-testing]#
```

c.接下来，创建后端对象为此，请进入克隆的 Git 存储库的 **fsx** 目录。打开文件 backend-ontap-nas.yaml。替换以下内容：将 **managementLIF** 替换为管理 DNS 名称，将 **dataLIF** 替换为Amazon FSx SVM 的 NFS DNS 名称，将 **svm** 替换为 SVM 名称。使用以下命令创建后端对象。

使用以下命令创建后端对象。

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

您可以从Amazon FSx控制台获取管理 DNS 名称、NFS DNS 名称和 SVM 名称，如下面的屏幕截图所示



d.现在，运行以下命令来验证后端对象是否已创建，并且阶段显示为绑定且状态为成功

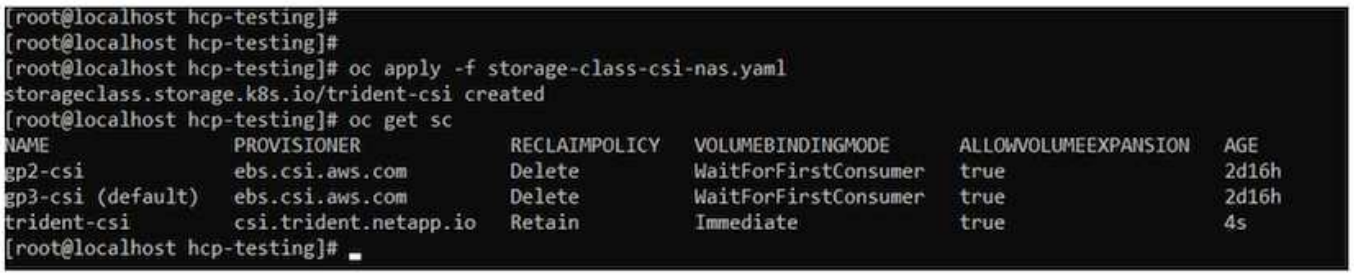
```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# oc apply -f backend-ontap-nas.yaml
tridentbackendconfig.trident.netapp.io/backend-fsx-ontap-nas created
[root@localhost hcp-testing]# oc get tbc -n trident
NAME                   BACKEND NAME   BACKEND UUID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

4.创建存储类 现在已经配置了Trident后端，您可以创建一个 Kubernetes 存储类来使用该后端。存储类是集群可用的资源对象。它描述并分类了您可以为应用程序请求的存储类型。

一个。查看 **fsx** 文件夹中的文件 **storage-class-csi-nas.yaml**。

```
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.在 **ROSA** 集群中创建存储类并验证 **trident-csi** 存储类是否已创建。



这完成了Trident CSI 驱动程序的安装及其与 FSx for ONTAP文件系统的连接。现在，您可以使用 FSx for ONTAP上的文件卷在 ROSA 上部署示例 Postgresql 有状态应用程序。

c.验证没有使用 **trident-csi** 存储类创建的 **PVC** 和 **PV**。

d.验证应用程序是否可以使用**Trident CSI** 创建 **PV**。

使用 **fsx** 文件夹中提供的 pvc-trident.yaml 文件创建 PVC。

```
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["使用Trident创建测试 PVC"]
```

要使用 iSCSI，您应该已经在工作节点上启用了 iSCSI（如前所示），并且需要创建 iSCSI 后端和存储类。以下是一些示例 yaml 文件。

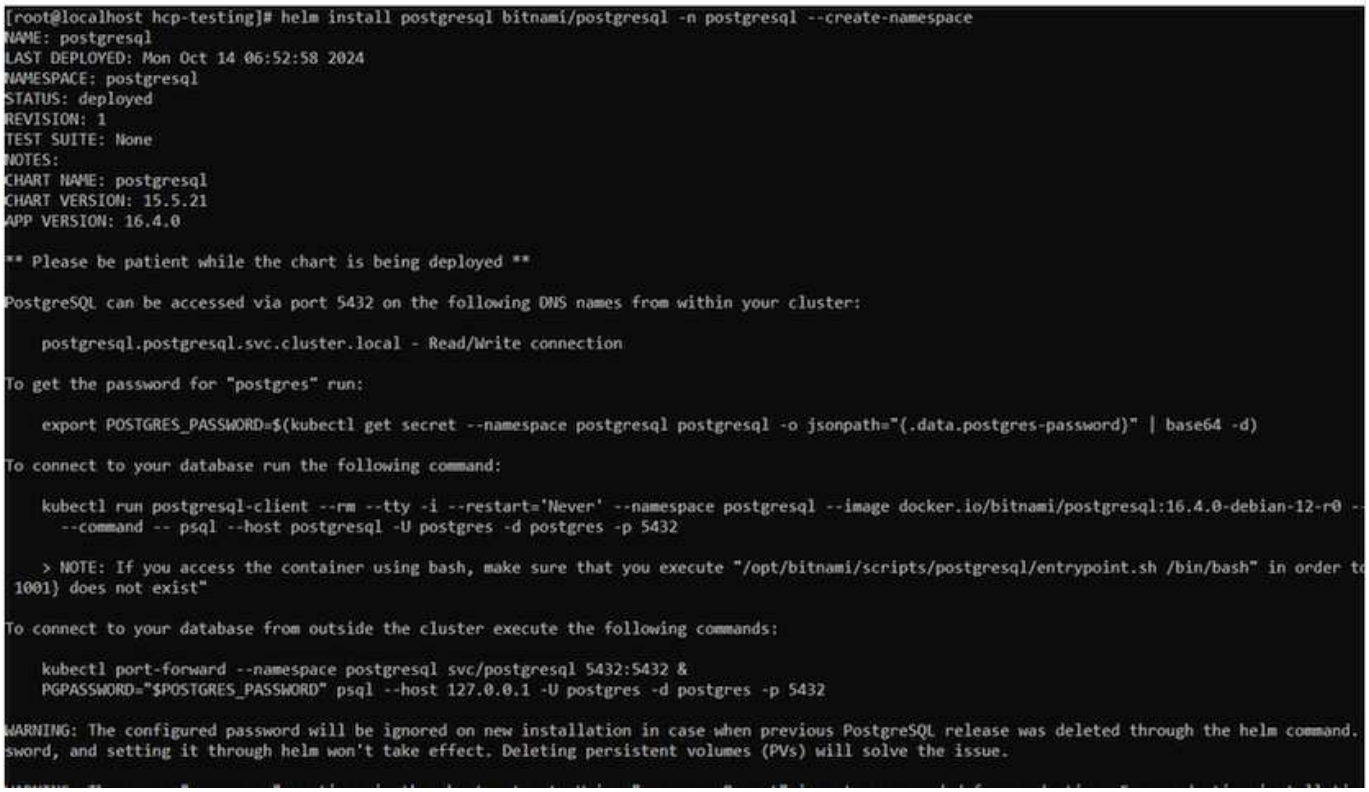
```
cat tbc.yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-san-secret
type: Opaque
stringData:
  username: fsxadmin
  password: <password for the fsxN filesystem>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 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.部署示例 **Postgresql** 有状态应用程序

一个。使用 **helm** 安装 **postgresql**

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



b.验证应用程序 **pod** 是否正在运行，以及是否为该应用程序创建了 **PVC** 和 **PV**。

c.部署 **Postgresql** 客户端

使用以下命令获取已安装的 **postgresql** 服务器的密码。

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

使用以下命令运行 **postgresql** 客户端并使用密码连接到服务器

```
$ 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.创建一个数据库和一个表。为表创建一个模式，并在表中插入 **2** 行数据。

```
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)
```

借助NetApp ONTAP在 AWS 上提供 Red Hat OpenShift 服务

本文档将概述如何将NetApp ONTAP与 AWS 上的 Red Hat OpenShift 服务 (ROSA) 结合使用。

创建卷快照

1.创建应用程序卷的快照 在本节中，我们将展示如何创建与应用程序关联的卷的 trident 快照。这将是应用程序数据的时间点副本。如果应用程序数据丢失，我们可以从该时间点的副本恢复数据。注意：此快照与ONTAP中的原始卷存储在同一个聚合中（本地或云端）。因此，如果ONTAP存储聚合丢失，我们就无法从其快照中恢复应用程序数据。

**一个。创建 VolumeSnapshotClass 将以下清单保存在名为volume-snapshot-class.yaml的文件中

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

使用上述清单创建快照。

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

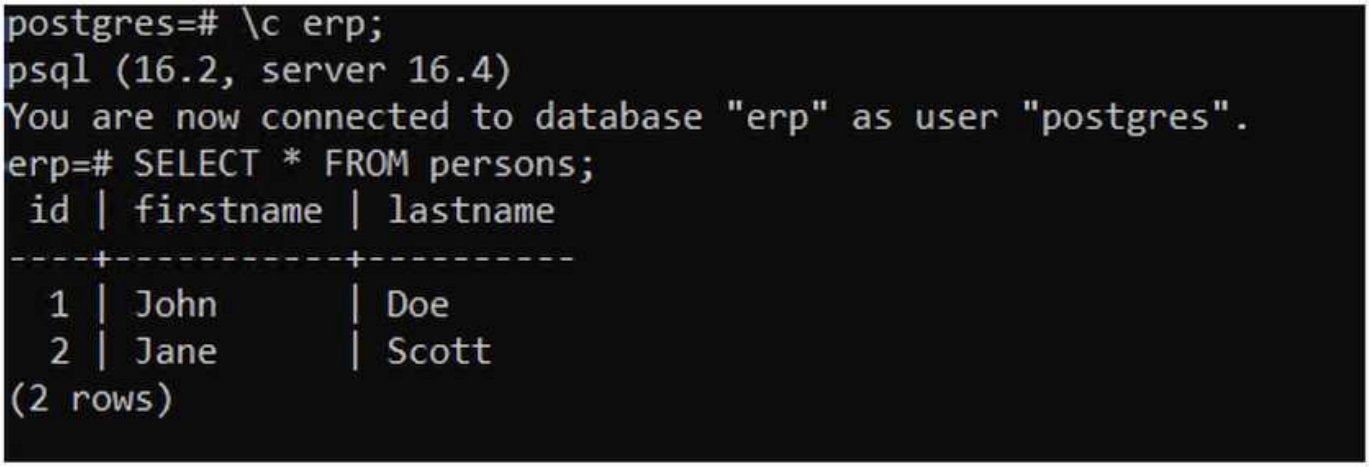
b.接下来，创建一个快照通过创建 VolumeSnapshot 来创建现有 PVC 的快照，以获取 Postgresql 数据的时间点副本。这会创建一个几乎不占用文件系统后端空间的 FSx 快照。将以下清单保存在名为volume-snapshot.yaml的文件中：

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

c.创建卷快照并确认已创建

删除数据库模拟数据丢失（数据丢失可能由于多种原因发生，这里我们只是通过删除数据库来模拟）

d.删除数据库以模拟数据丢失（数据丢失可能由于多种原因而发生，这里我们只是通过删除数据库来模拟）



从卷快照还原

1.从快照恢复 在本节中，我们将展示如何从应用程序卷的 trident 快照恢复应用程序。

一个。从快照创建卷克隆

要将卷恢复到以前的状态，您必须根据所拍摄快照中的数据创建一个新的 PVC。为此，请将以下清单保存在名为 pvc-clone.yaml 的文件中

```
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
```

使用上述清单，通过使用快照作为源创建 PVC 来创建卷的克隆。应用清单并确保创建克隆。

```
[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.删除原始的 **postgresql** 安装

```
[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.使用新的克隆 **PVC** 创建新的 **postgresql** 应用程序

```
$ 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.验证应用程序 **pod** 处于运行状态

```
[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.验证 **Pod** 是否使用克隆作为其 **PVC**

```
[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) 要验证数据库是否已按预期恢复，请返回容器控制台并显示现有数据库

```
[root@localhost hcp-testing]# kubectl run postgresql-client --rm --tty -i --restart='Never' --namespace postgresql --image docker.io/bitnami/postgresql:
$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 securityContext.allowPri
 capabilities (container "postgresql-client" must set securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod or container "postgresql-clie
Root=true), seccompProfile (pod or container "postgresql-client" must set securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost")
If you don't see a command prompt, try pressing enter.

postgres=# \l
                                                List of databases
   Name    |  Owner   | Encoding | Locale Provider |   Collate   |    Ctype    | ICU Locale | ICU Rules |   Access privileges
-----------+----------+----------+-----------------+-------------+-------------+------------+-----------+-----------------------
 erp       | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           |
 postgres  | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           |
 template0 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           | =c/postgres          +
           |          |          |                 |             |             |            |           | postgres=CTc/postgres
 template1 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           | =c/postgres          +
           |          |          |                 |             |             |            |           | postgres=CTc/postgres
(4 rows)

postgres=# \c erp;
psql (16.2, server 16.4)
You are now connected to database "erp" as user "postgres".
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
  2 | Jane      | Scott
(2 rows)
```

演示视频

使用托管控制平面在 AWS 上将Amazon FSx for NetApp ONTAP与 Red Hat OpenShift 服务结合使用

可以找到更多关于 Red Hat OpenShift 和 OpenShift 解决方案的视频"此处"。