Service Red Hat OpenShift sur AWS avec FSxN

NetApp container solutions

NetApp
January 25, 2026

Sommaire

Service Red Hat OpenShift sur AWS avec FSxN

Service Red Hat OpenShift sur AWS avec NetApp ONTAP

Aperçu

Dans cette section, nous montrerons comment utiliser FSx pour ONTAP comme couche de stockage persistante pour les applications exécutées sur ROSA. Il montrera l'installation du pilote NetApp Trident CSI sur un cluster ROSA, le provisionnement d'un système de fichiers FSx pour ONTAP et le déploiement d'un exemple d'application avec état. Il montrera également des stratégies pour sauvegarder et restaurer les données de votre application. Avec cette solution intégrée, vous pouvez établir une infrastructure de stockage partagée qui s'adapte sans effort à toutes les zones de disponibilité, simplifiant ainsi les processus de mise à l'échelle, de protection et de restauration de vos données à l'aide du pilote Trident CSI.

Prérequis

- "compte AWS"
- "Un compte Red Hat"
- Utilisateur IAM"avec les autorisations appropriées" pour créer et accéder au cluster ROSA
- "AWS CLI"
- "ROSA CLI"
- "Interface de ligne de commande OpenShift"(oc)
- Casque 3"documentation"
- "Un cluster HCP ROSA"
- "Accès à la console Web Red Hat OpenShift"

Ce diagramme montre le cluster ROSA déployé dans plusieurs AZ. Les nœuds maîtres du cluster ROSA et les nœuds d'infrastructure se trouvent dans le VPC de Red Hat, tandis que les nœuds de travail se trouvent dans un VPC du compte client. Nous allons créer un système de fichiers FSx pour ONTAP dans le même VPC et installer le pilote Trident dans le cluster ROSA, permettant à tous les sous-réseaux de ce VPC de se connecter au système de fichiers.

Configuration initiale

1. Provisionner FSx pour NetApp ONTAP

Créez un FSx multi-AZ pour NetApp ONTAP dans le même VPC que le cluster ROSA. Il existe plusieurs façons de procéder. Les détails de la création de FSxN à l'aide d'une pile CloudFormation sont fournis

a.Cloner le dépôt GitHub

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

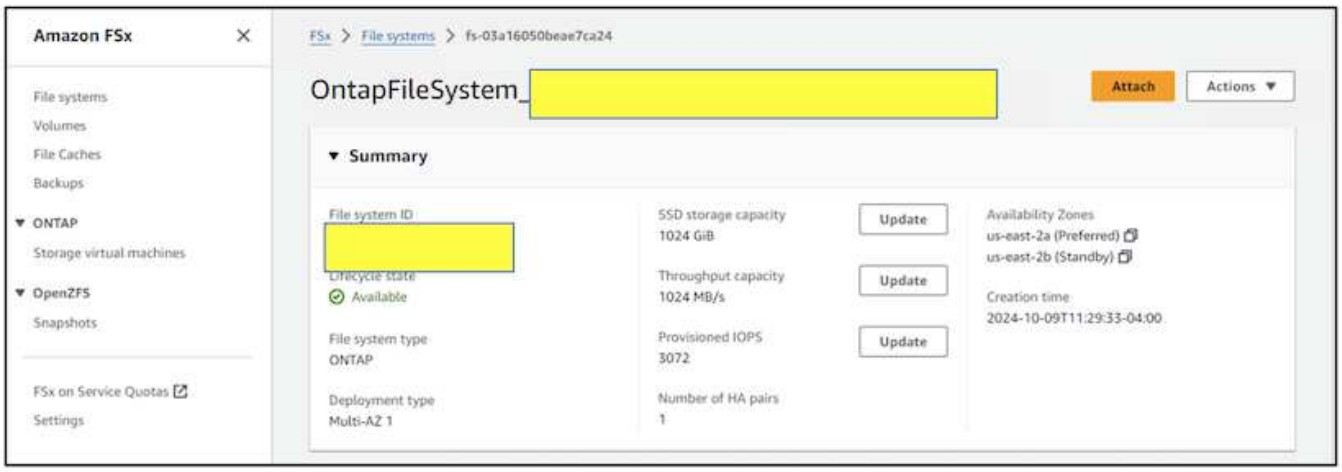
b.Exécutez la pile CloudFormation Exécutez la commande ci-dessous en remplaçant les valeurs des paramètres par vos propres valeurs :

```
$ 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
```

Où : region-name : identique à la région où le cluster ROSA est déployé subnet1_ID : identifiant du sous-réseau préféré pour FSxN subnet2_ID : identifiant du sous-réseau de secours pour FSxN VPC_ID : identifiant du VPC où le cluster ROSA est déployé routetable1_ID, routetable2_ID : identifiants des tables de routage associées aux sous-réseaux choisis ci-dessus your_allowed_CIDR : plage CIDR autorisée pour les règles d'entrée des groupes de sécurité FSx for ONTAP pour contrôler l'accès. Vous pouvez utiliser 0.0.0.0/0 ou tout CIDR approprié pour permettre à tout le trafic d'accéder aux ports spécifiques de FSx pour ONTAP. Définir le mot de passe administrateur : un mot de passe pour se connecter à FSxN Définir le mot de passe SVM : un mot de passe pour se connecter à la SVM qui sera créée.

Vérifiez que votre système de fichiers et votre machine virtuelle de stockage (SVM) ont été créés à l'aide de la console Amazon FSx , comme indiqué ci-dessous :



2.Installer et configurer le pilote Trident CSI pour le cluster ROSA

b.Installer Trident

Les nœuds de travail du cluster ROSA sont préconfigurés avec des outils NFS qui vous permettent d'utiliser les protocoles NAS pour le provisionnement et l'accès au stockage.

Si vous souhaitez utiliser iSCSI à la place, vous devez préparer les nœuds de travail pour iSCSI. À partir de la version Trident 25.02, vous pouvez facilement préparer les nœuds de travail du cluster ROSA (ou de tout cluster OpenShift) pour effectuer des opérations iSCSI sur le stockage FSxN. Il existe deux manières simples d'installer Trident 25.02 (ou version ultérieure) qui automatisent la préparation des nœuds de travail pour iSCSI. 1. en utilisant le node-prep-flag depuis la ligne de commande à l'aide de l'outil tridentctl. 2. Utilisation de l'opérateur Trident certifié Red Hat depuis le hub opérateur et personnalisation. 3. Utilisation de Helm.

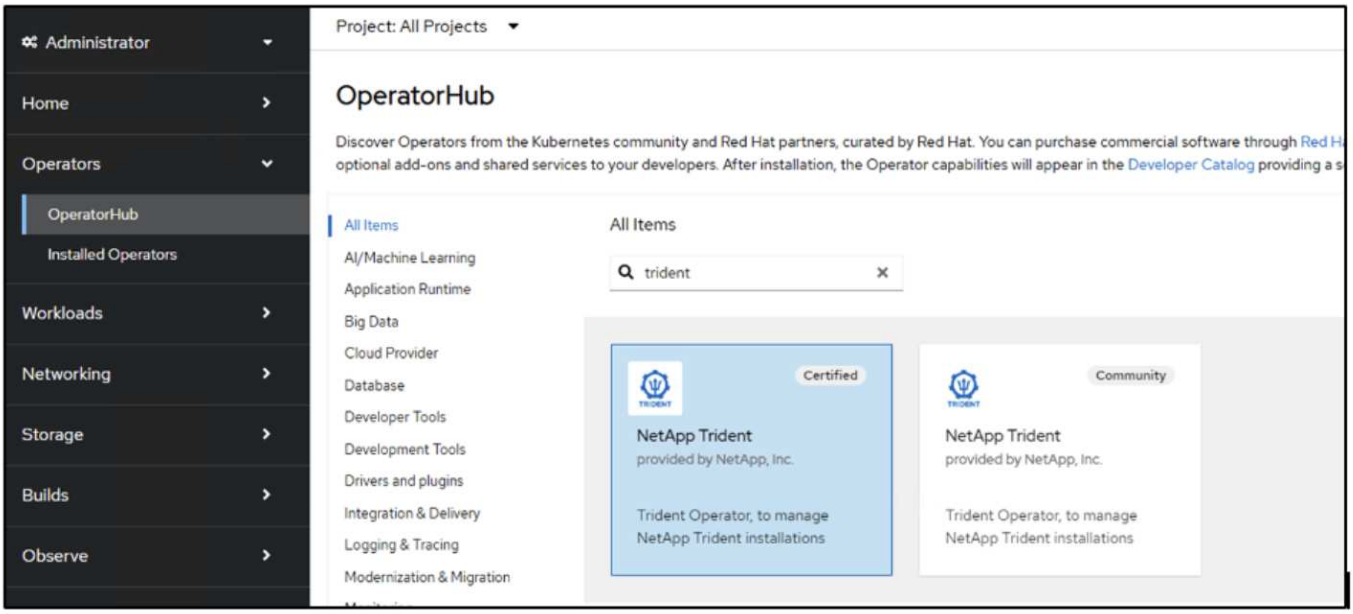
L'utilisation de l'une des méthodes ci-dessus sans activer la préparation des nœuds vous permettra d'utiliser uniquement les protocoles NAS pour le provisionnement du stockage sur FSxN.

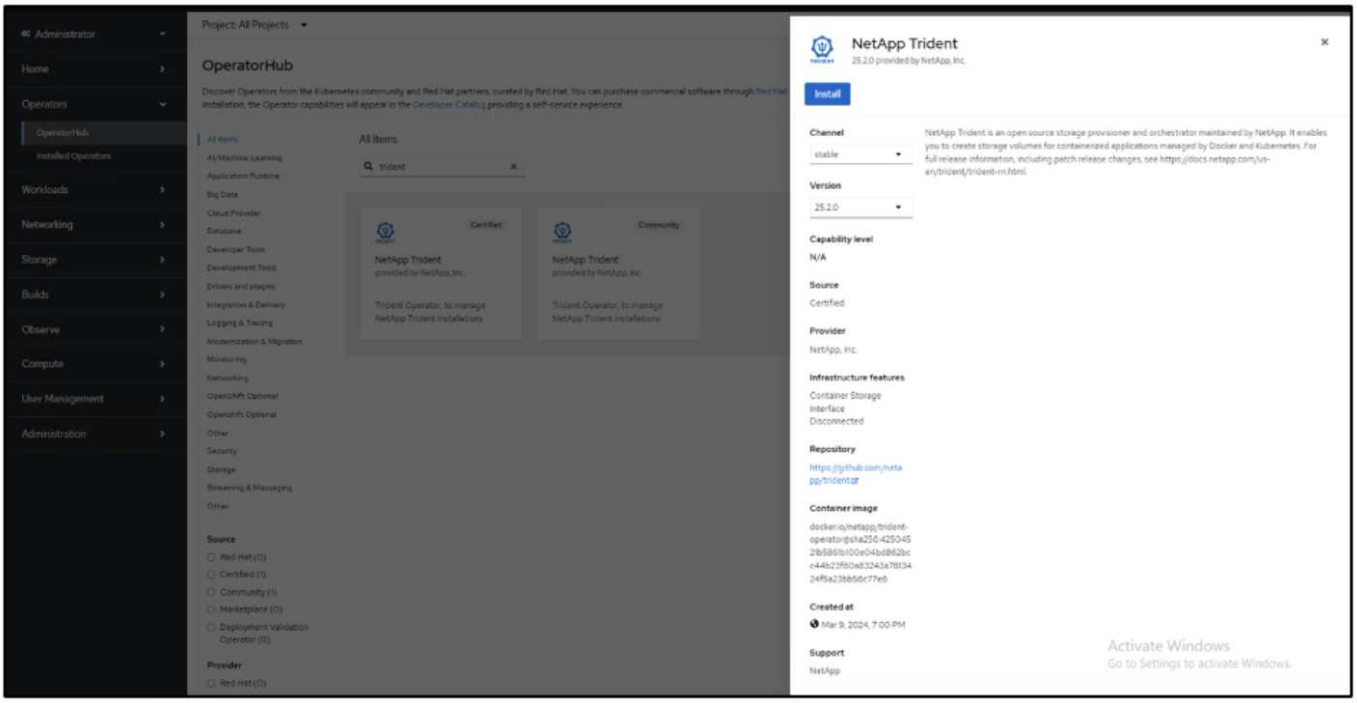
Méthode 1 : utiliser l'outil tridentctl

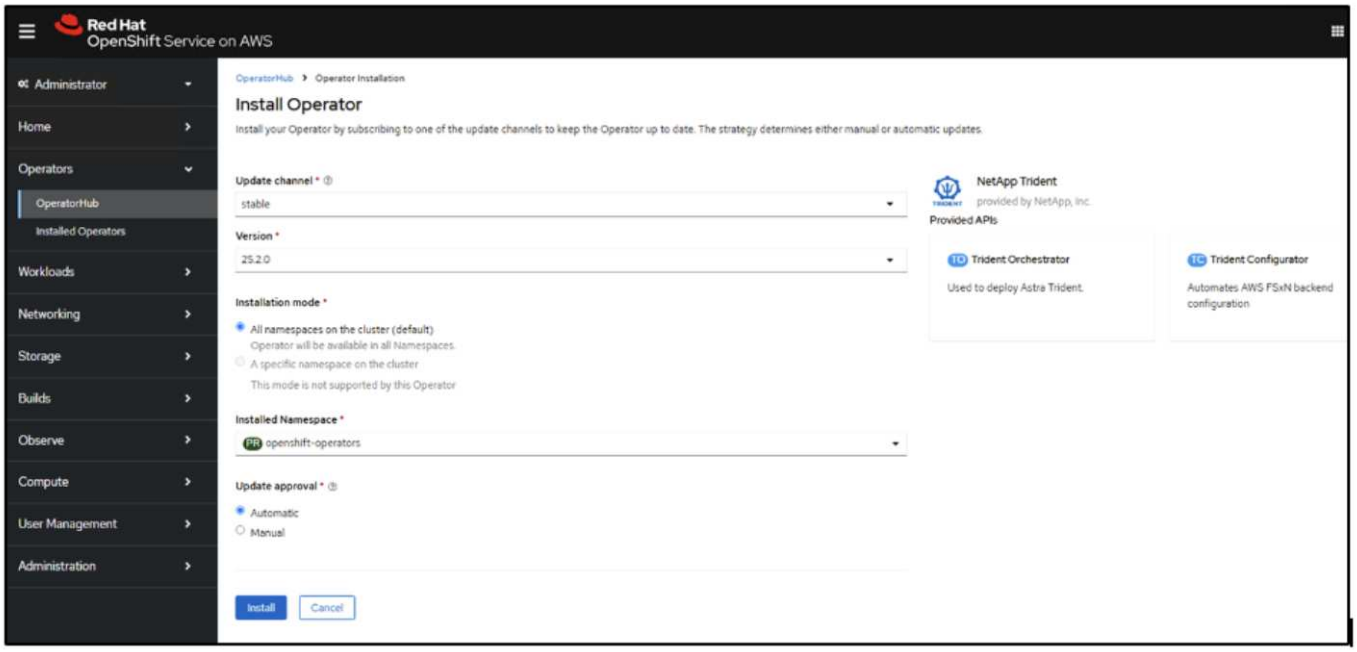
Utilisez l'indicateur node-prep et installez Trident comme indiqué. Avant d'émettre la commande d'installation, vous devez avoir téléchargé le package d'installation. "la documentation ici" .

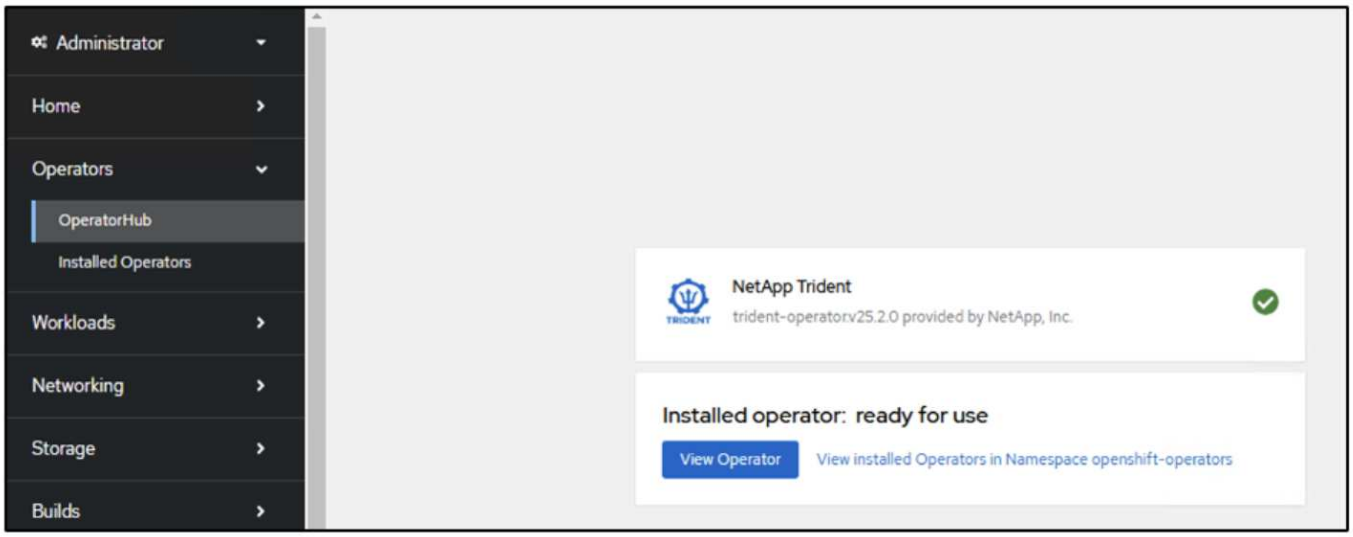
```
#./tridentctl install trident -n trident --node-prep=iscsi
```

Méthode 2 : utiliser l'opérateur Trident certifié Red Hat et personnaliser Depuis OperatorHub, recherchez l'opérateur Trident certifié Red Hat et installez-le.

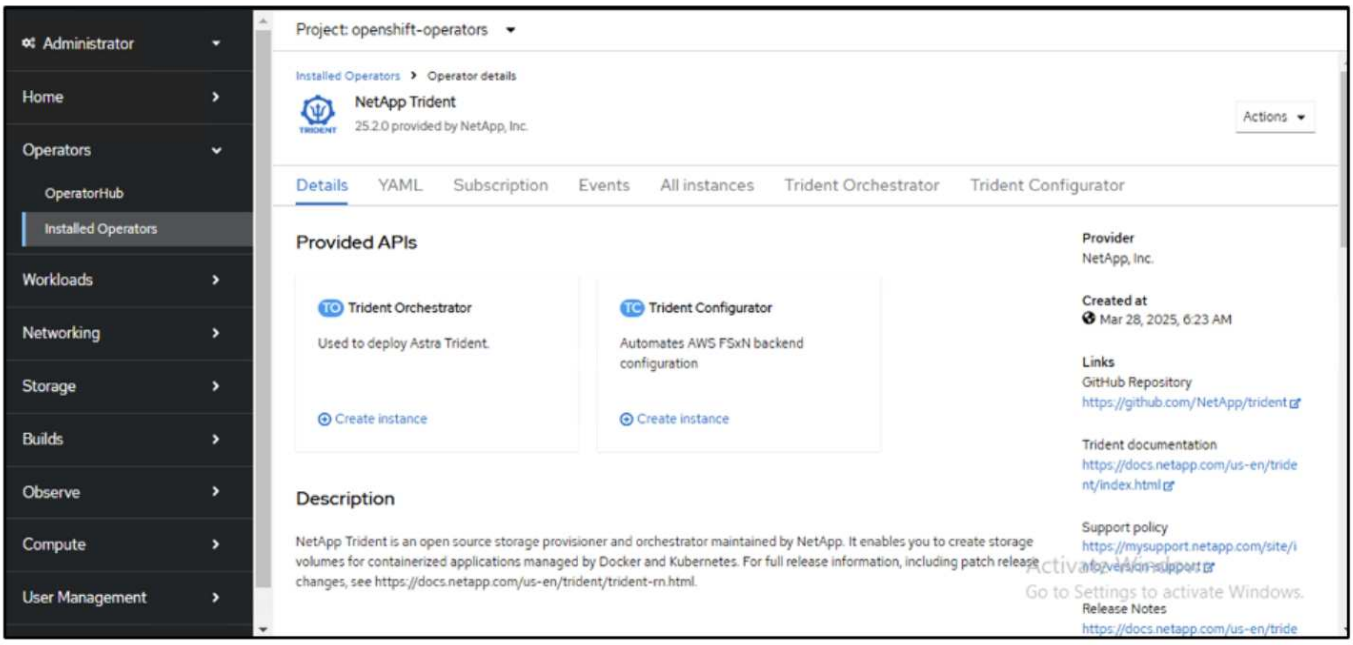


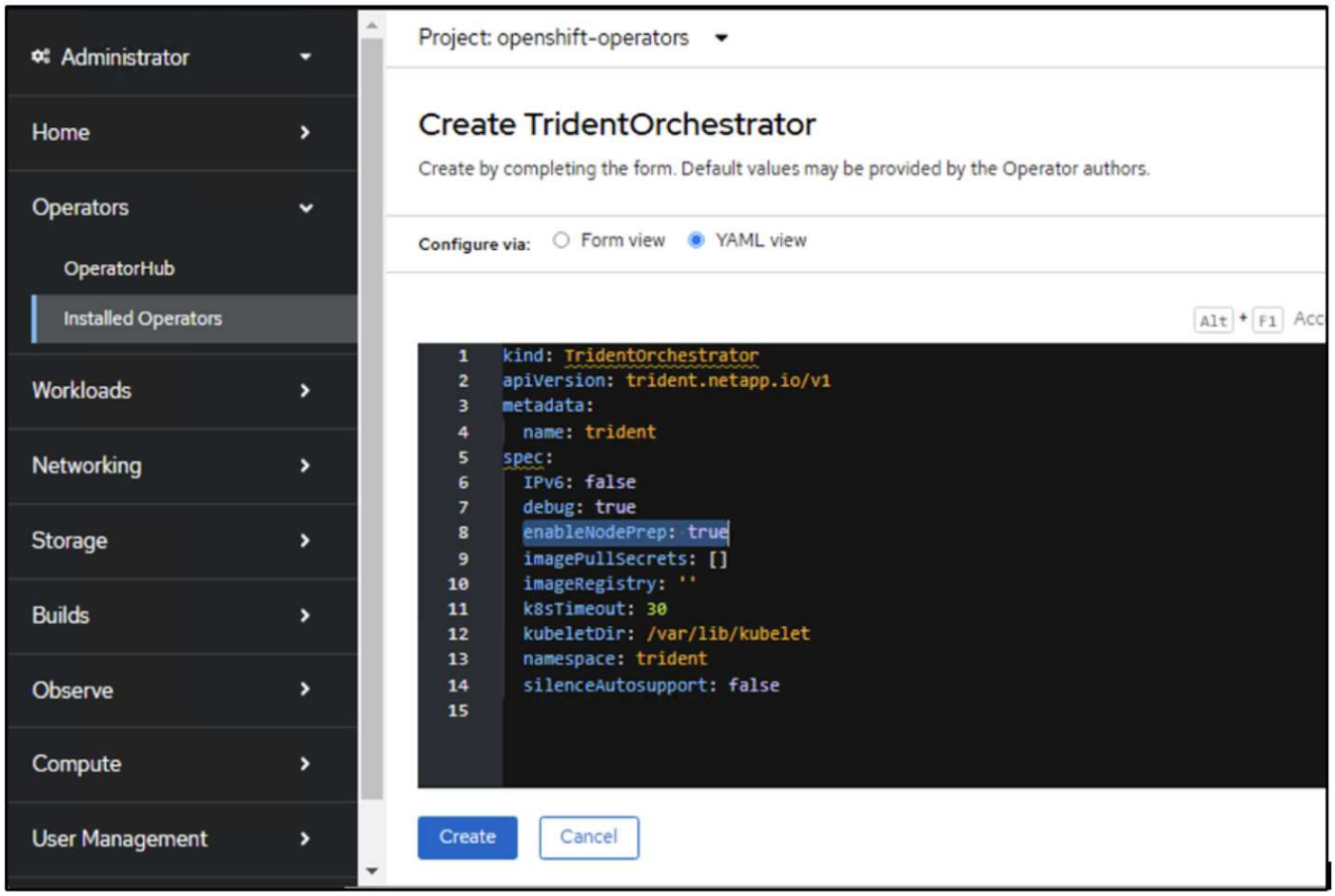

Administrator
Home
Operators
OperatorHub
Installed Operators
Workloads
Networking
Storage
Builds
Observe
Compute
User Management
Administration
Project: All Projects
OperatorHub
All Items
trident
NetApp Trident
provided by NetApp, Inc.
Trident Operator, to manage NetApp Trident installations
Certified
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.

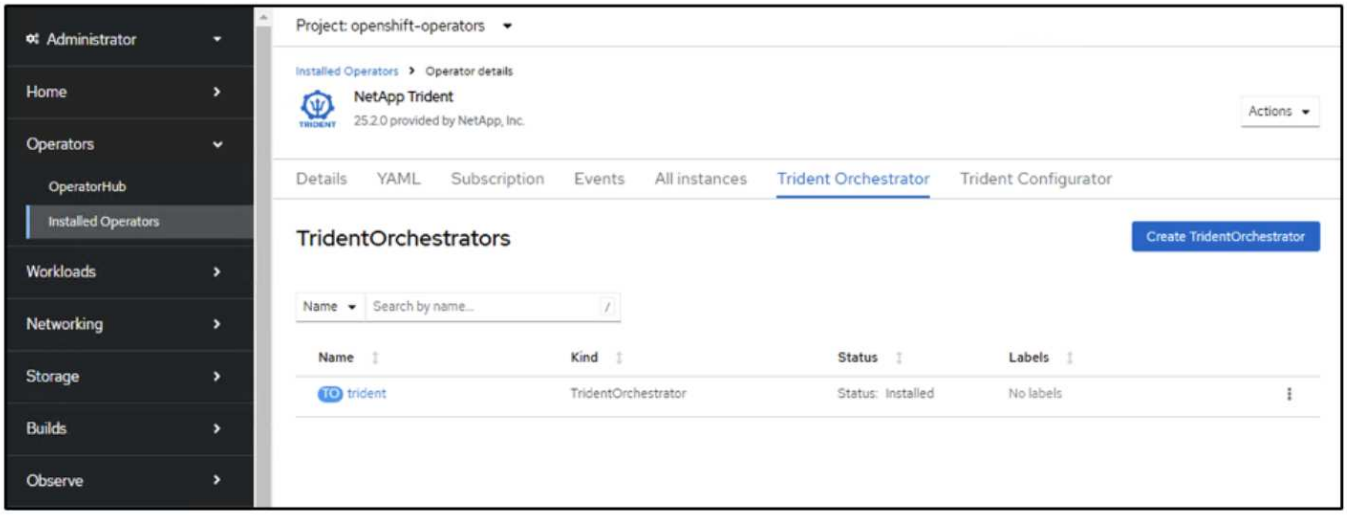

Red Hat
OpenShift Service on AWS
Administrator
Home
Operators
OperatorHub
Installed Operators
Workloads
Networking
Storage
Builds
Observe
Compute
User Management
Administration
OperatorHub > Operator Installation
Install Operator
Install your Operator by subscribing to one of the update channels to keep 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



Ensuite, créez l'instance Trident Orchestrator. Utilisez la vue YAML pour définir des valeurs personnalisées ou activer la préparation du nœud iscsi pendant l'installation.








```
[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]#
```

L’installation de Trident à l’aide de l’une des méthodes ci-dessus préparera les nœuds de travail du cluster ROSA pour iSCSI en démarrant les services iscsid et multipathd et en définissant les éléments suivants dans le fichier /etc/multipath.conf

```
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.Vérifiez que tous les modules Trident sont en cours d'exécution

```
[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]#
```

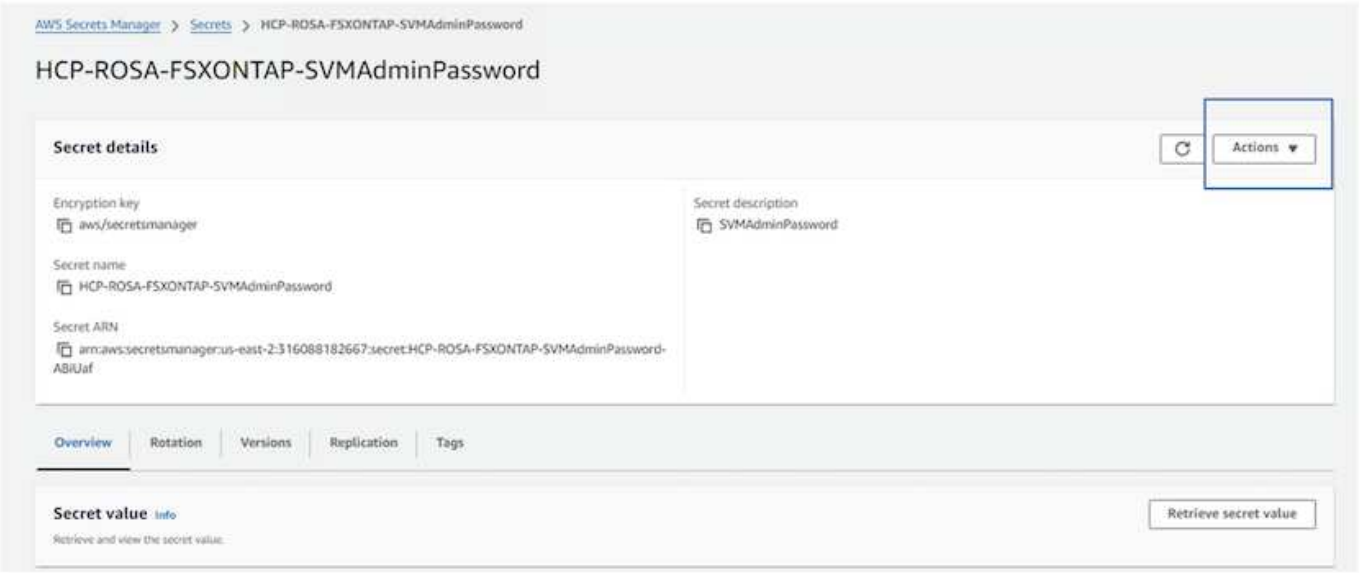
3. Configurer le backend Trident CSI pour utiliser FSx pour ONTAP (ONTAP NAS)

La configuration back-end de Trident indique à Trident comment communiquer avec le système de stockage (dans ce cas, FSx pour ONTAP). Pour créer le backend, nous fournirons les informations d'identification de la machine virtuelle de stockage à laquelle se connecter, ainsi que les interfaces de gestion de cluster et de données NFS. Nous utiliserons le"pilote ontap-nas" pour provisionner des volumes de stockage dans le système de fichiers FSx.

un. Tout d'abord, créez un secret pour les informations d'identification SVM en utilisant le fichier yaml suivant

```
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>
```

Vous pouvez également récupérer le mot de passe SVM créé pour FSxN à partir d'AWS Secrets Manager comme indiqué ci-dessous.



b.Ensuite, ajoutez le secret des informations d'identification SVM au cluster ROSA à l'aide de la commande suivante

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

Vous pouvez vérifier que le secret a été ajouté dans l'espace de noms trident à l'aide de la commande suivante

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

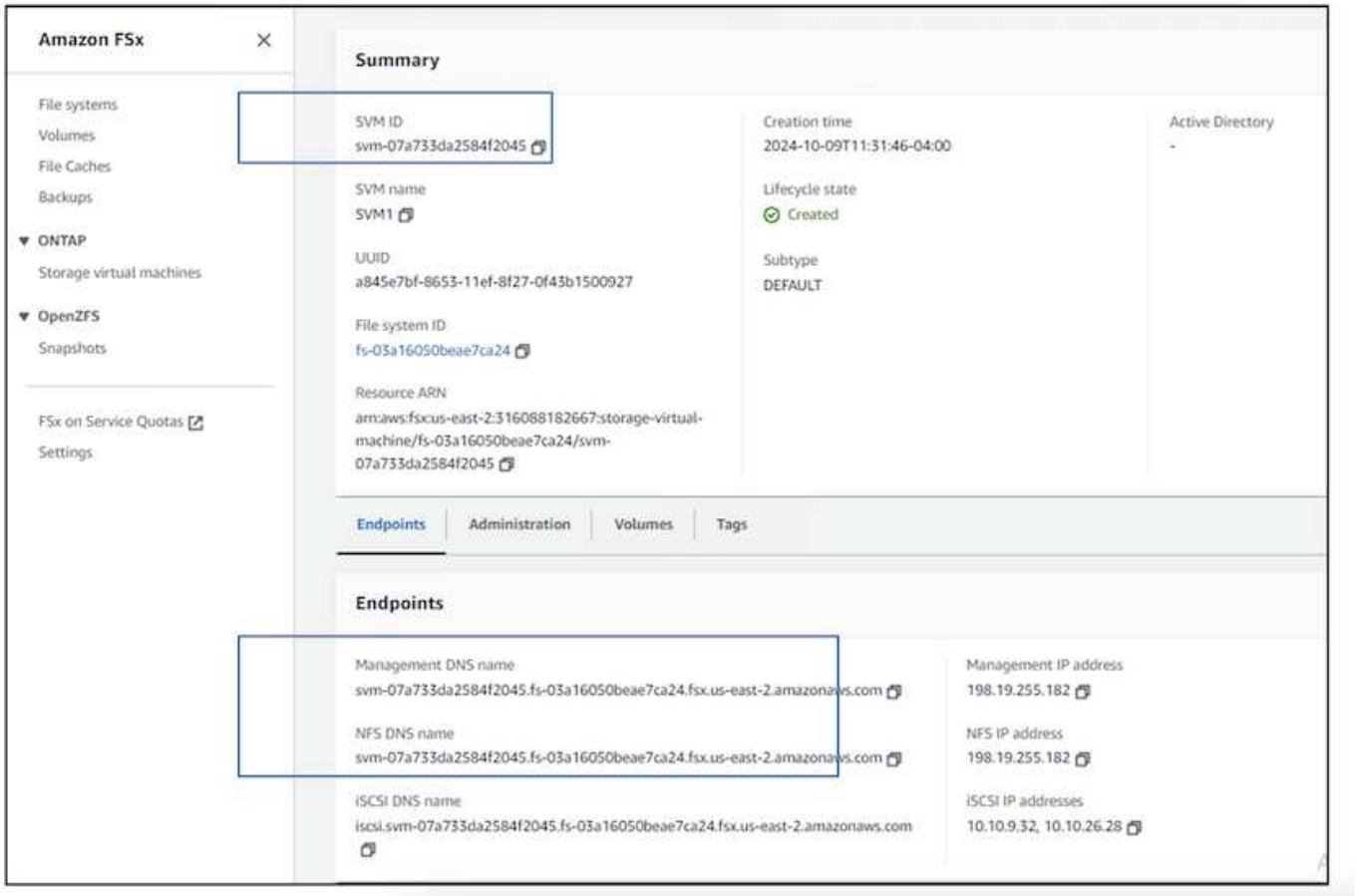
```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# oc get secrets -n trident | grep backend-fsx-ontap-nas-secret
backend-fsx-ontap-nas-secret         Opaque                  2      21h
[root@localhost hcp-testing]#
```

c. Ensuite, créez l'objet backend Pour cela, déplacez-vous dans le répertoire **fsx** de votre dépôt Git cloné. Ouvrez le fichier backend-ontap-nas.yaml. Remplacez les éléments suivants : **managementLIF** par le nom DNS de gestion, **dataLIF** par le nom DNS NFS de la SVM Amazon FSx et **svm** par le nom de la SVM. Créez l'objet backend à l'aide de la commande suivante.

Créez l'objet backend à l'aide de la commande suivante.

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

Vous pouvez obtenir le nom DNS de gestion, le nom DNS NFS et le nom SVM à partir de la console Amazon FSx, comme indiqué dans la capture d'écran ci-dessous.



d. Maintenant, exécutez la commande suivante pour vérifier que l'objet backend a été créé et que la phase affiche Bound et que le statut est Success

```
[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. Créer une classe de stockage Maintenant que le backend Trident est configuré, vous pouvez créer une classe de stockage Kubernetes pour utiliser le backend. La classe de stockage est un objet ressource mis à disposition du cluster. Il décrit et classe le type de stockage que vous pouvez demander pour une application.

un. Vérifiez le fichier storage-class-csi-nas.yaml dans le dossier fsx.

```
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. Créez une classe de stockage dans le cluster ROSA et vérifiez que la classe de stockage trident-csi a été créée.

```
[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]#
```

Ceci termine l'installation du pilote Trident CSI et sa connectivité au système de fichiers FSx pour ONTAP . Vous pouvez désormais déployer un exemple d'application Postgresql avec état sur ROSA à l'aide de volumes de fichiers sur FSx pour ONTAP.

c. Vérifiez qu'aucun PVC et PV n'est créé à l'aide de la classe de stockage trident-csi.

d. Vérifiez que les applications peuvent créer des PV à l'aide de Trident CSI.

Créez un PVC en utilisant le fichier pvc-trident.yaml fourni dans le dossier **fsx**.

```
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["créer un test PVC à l'aide
de Trident"]
```

Pour utiliser iSCSI, vous devez avoir activé iSCSI sur les nœuds de travail comme indiqué précédemment et vous devez créer un backend iSCSI et une classe de stockage. Voici quelques exemples de fichiers yaml.

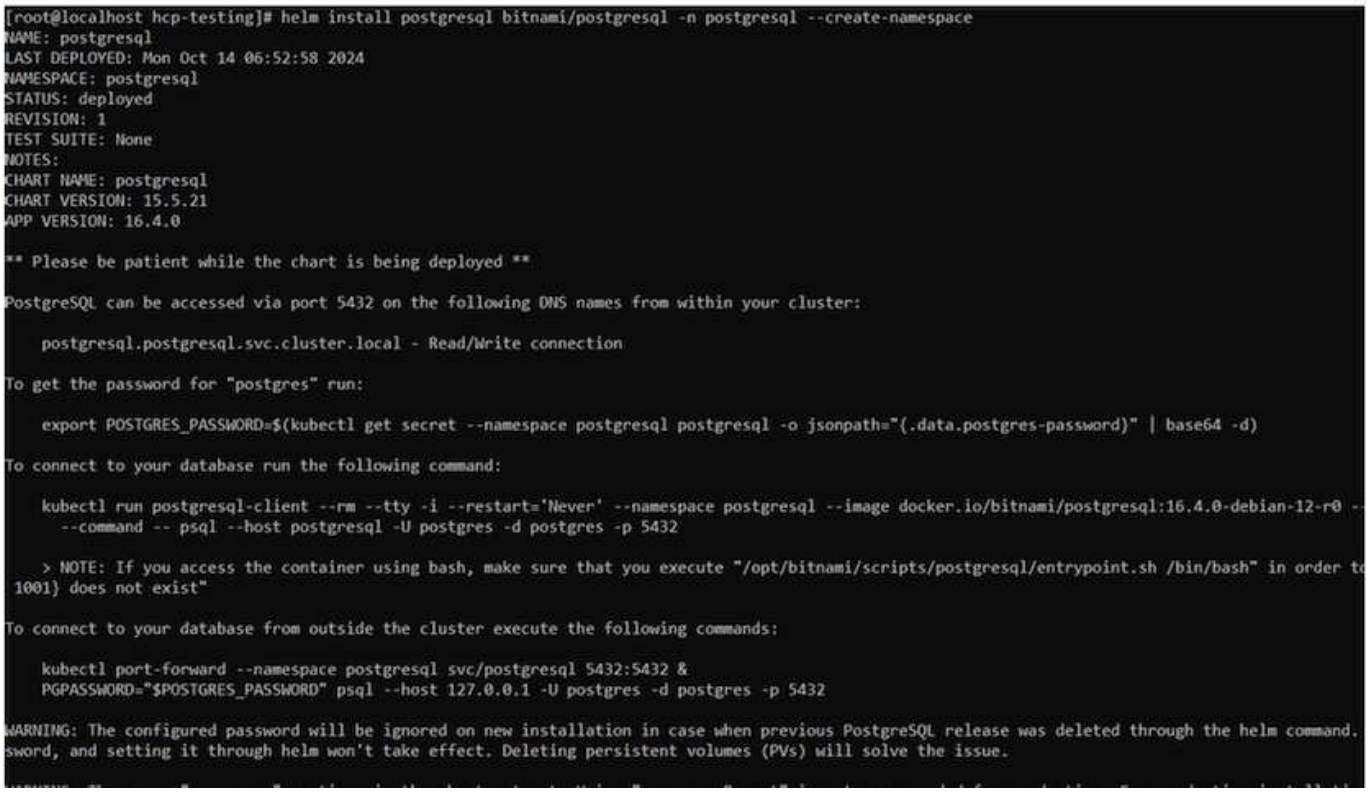
```
cat tbc.yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-san-secret
type: Opaque
stringData:
  username: fsxadmin
  password: <password for the fsxN filesystem>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 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. Déployer un exemple d'application Postgresql avec état

un. Utilisez Helm pour installer PostgreSQL

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



b. Vérifiez que le pod d'application est en cours d'exécution et qu'un PVC et un PV sont créés pour l'application.

c. Déployer un client Postgresql

Utilisez la commande suivante pour obtenir le mot de passe du serveur postgresql qui a été installé.

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

Utilisez la commande suivante pour exécuter un client postgresql et vous connecter au serveur en utilisant le mot de passe

```
$ 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. Créer une base de données et une table. Créez un schéma pour la table et insérez 2 lignes de données dans la table.

```
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)
```

Service Red Hat OpenShift sur AWS avec NetApp ONTAP

Ce document décrit comment utiliser NetApp ONTAP avec le service Red Hat OpenShift sur AWS (ROSA).

Créer un instantané de volume

1. Créer un instantané du volume de l'application Dans cette section, nous montrerons comment créer un instantané trident du volume associé à l'application. Il s'agira d'une copie ponctuelle des données de l'application. Si les données de l'application sont perdues, nous pouvons récupérer les données à partir de cette copie à ce moment-là. REMARQUE : cet instantané est stocké dans le même agrégat que le volume d'origine dans ONTAP(sur site ou dans le cloud). Ainsi, si l'agrégat de stockage ONTAP est perdu, nous ne pouvons pas récupérer les données de l'application à partir de son instantané.

**un. Créer un VolumeSnapshotClass Enregistrez le manifeste suivant dans un fichier appelé volume-snapshot-class.yaml

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

Créez un instantané en utilisant le manifeste ci-dessus.

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

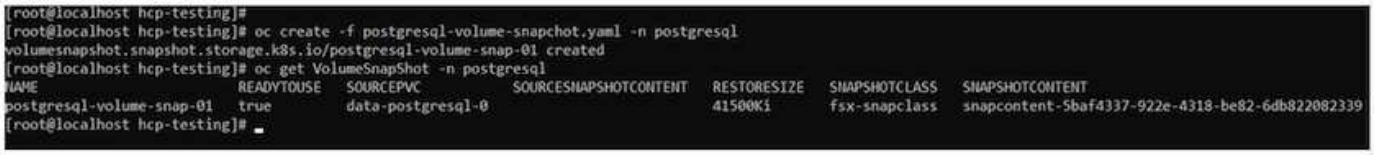
b. Ensuite, créez un instantané Créez un instantané du PVC existant en créant VolumeSnapshot pour prendre une copie ponctuelle de vos données Postgresql. Cela crée un instantané FSx qui ne prend presque pas de place dans le backend du système de fichiers. Enregistrez le manifeste suivant dans un fichier appelé

volume-snapshot.yaml :

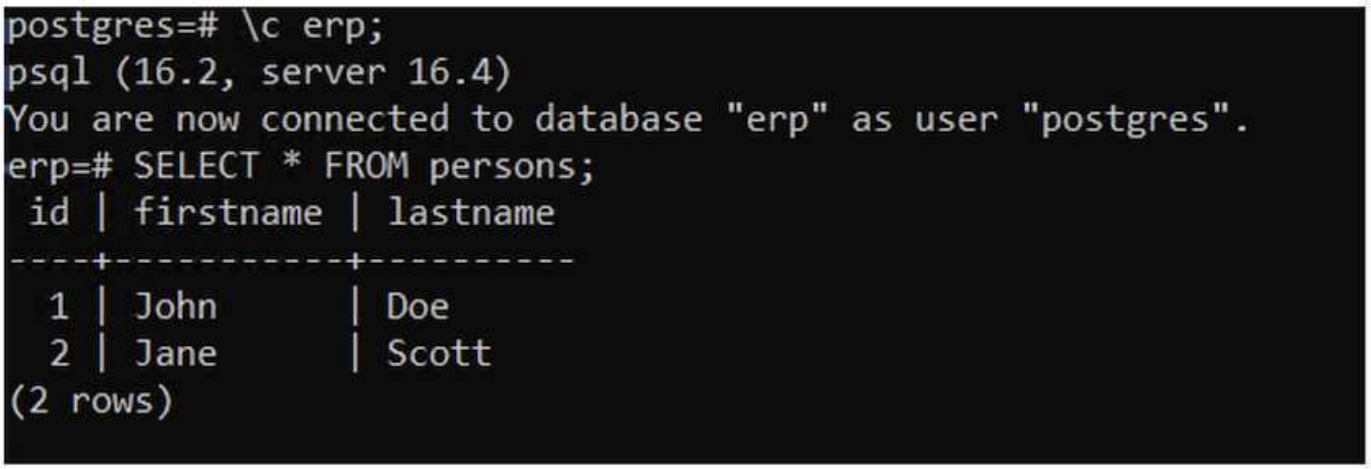
```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
 name: postgresql-volume-snap-01
spec:
 volumeSnapshotClassName: fsx-snapclass
 source:
   persistentVolumeClaimName: data-postgresql-0
```

c. Créez l'instantané du volume et confirmez qu'il est créé

Supprimez la base de données pour simuler la perte de données (la perte de données peut survenir pour diverses raisons, ici nous la simulons simplement en supprimant la base de données)



d. Supprimez la base de données pour simuler la perte de données (la perte de données peut survenir pour diverses raisons, ici nous la simulons simplement en supprimant la base de données)



Restaurer à partir d'un instantané de volume

1. Restaurer à partir d'un instantané Dans cette section, nous montrerons comment restaurer une application à partir de l'instantané trident du volume de l'application.

un. Créer un clone de volume à partir du snapshot

Pour restaurer le volume à son état précédent, vous devez créer un nouveau PVC basé sur les données de l'instantané que vous avez pris. Pour ce faire, enregistrez le manifeste suivant dans un fichier nommé pvc-clone.yaml

```
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
```

Créez un clone du volume en créant un PVC en utilisant l'instantané comme source à l'aide du manifeste ci-dessus. Appliquez le manifeste et assurez-vous que le clone est créé.

```
[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. Supprimer l'installation postgresql d'origine

```
[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. Créer une nouvelle application postgresql en utilisant le nouveau clone PVC

```
$ 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. Vérifiez que le pod d'application est en cours d'exécution

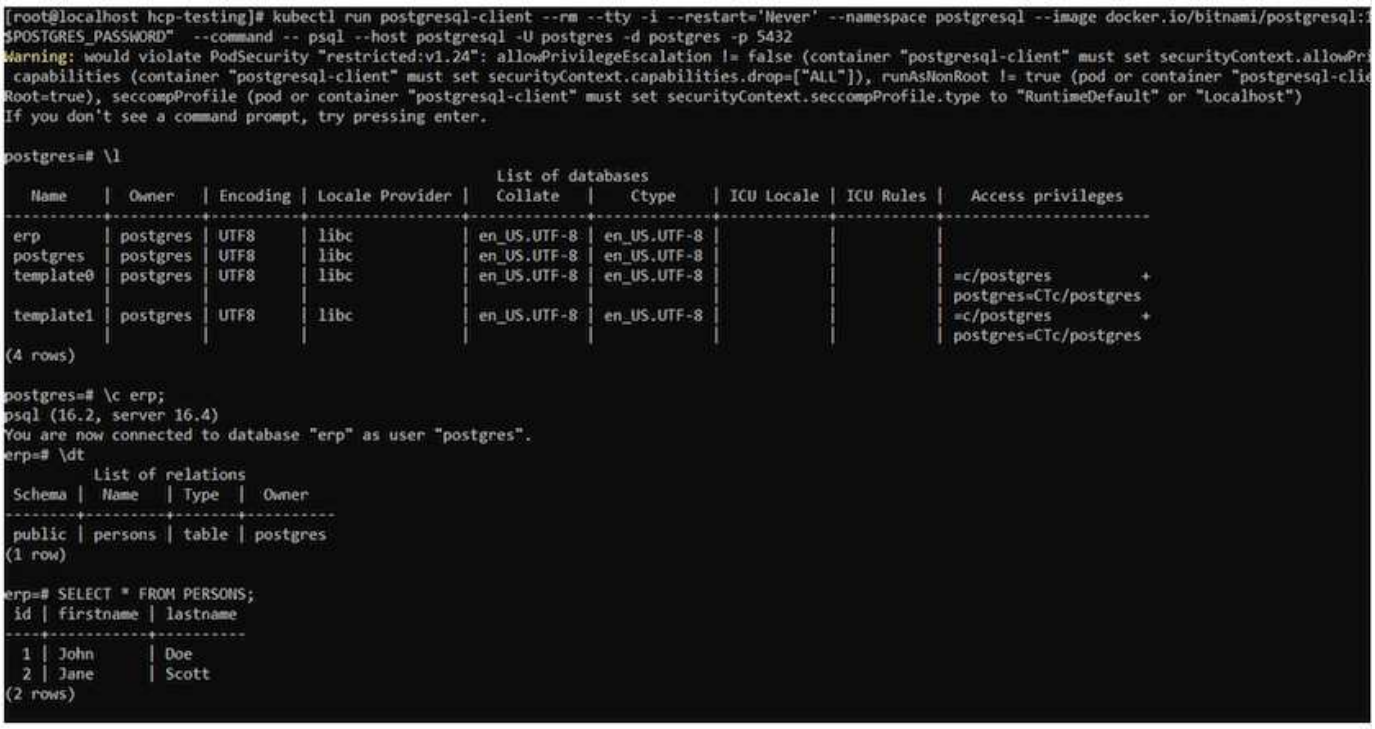
```
[root@localhost hcp-testing]# oc get pods -n postgresql
NAME           READY   STATUS    RESTARTS   AGE
postgresql-0   1/1     Running   0          2m1s
[root@localhost hcp-testing]#
```

e. Vérifiez que le pod utilise le clone comme 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) Pour valider que la base de données a été restaurée comme prévu, revenez à la console du conteneur et affichez les bases de données existantes



Vidéo de démonstration

Amazon FSx for NetApp ONTAP avec Red Hat OpenShift Service sur AWS à l'aide du plan de contrôle hébergé

D'autres vidéos sur Red Hat OpenShift et les solutions OpenShift sont disponibles"ici" .