FSxN이 포함된 AWS 기반 Red Hat OpenShift Service

NetApp Solutions

NetApp
January 09, 2025

목차

FSxN이 포함된 AWS 기반 Red Hat OpenShift Service

NetApp ONTAP가 포함된 AWS 기반 Red Hat OpenShift Service

개요

이 섹션에서는 Rosa에서 실행되는 애플리케이션의 영구 스토리지 계층으로 FSx for ONTAP을 활용하는 방법을 알아봅니다. Rosa 클러스터에 NetApp Trident CSI 드라이버 설치, FSx for ONTAP 파일 시스템 프로비저닝 및 샘플 상태 저장 애플리케이션 배포를 보여 줍니다. 또한 응용 프로그램 데이터를 백업하고 복원하기 위한 전략도 보여 줍니다. 이 통합 솔루션을 사용하면 AZ 간에 손쉽게 확장되는 공유 스토리지 프레임워크를 구축하여 Trident CSI 드라이버를 사용하여 데이터 크기 조정, 보호 및 복원 프로세스를 간소화할 수 있습니다.

필수 구성 요소

- "설치하 고 있습니다"
- "Red Hat 계정"
- IAM 사용자가 "적절한 권한이 있어야 합니다"ROSA 클러스터를 생성하고 액세스합니다
- "AWS CLI를 참조하십시오"
- "로사 CLI"
- "OpenShift 명령줄 인터페이스" (OC)
- 도움 3 "문서화"
- "HCP ROSA 클러스터"
- "Red Hat OpenShift 웹 콘솔에 액세스합니다"

이 다이어그램은 여러 AZ에 배포된 ROSA 클러스터를 보여 줍니다. Rosa 클러스터의 마스터 노드, 인프라 노드는 Red Hat의 VPC에 있고, 작업자 노드는 고객 어카운트의 VPC에 있습니다. 동일한 VPC 내에 FSx for ONTAP 파일 시스템을 생성하고 ROSA 클러스터에 Trident 드라이버를 설치하여 이 VPC의 모든 서브넷이 파일 시스템에 연결할 수 있게 합니다.

초기 설정

- 1. NetApp ONTAP용 FSx 프로비저닝**

ROSA 클러스터와 동일한 VPC에서 다중 AZ FSx for NetApp ONTAP를 생성합니다. 이 작업은 여러 가지 방법으로 수행할 수 있습니다. CloudFormation Stack을 사용하여 FSxN을 생성하는 방법에 대한 세부 정보를 제공합니다

- a. GitHub 리포지토리를 복제합니다**

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

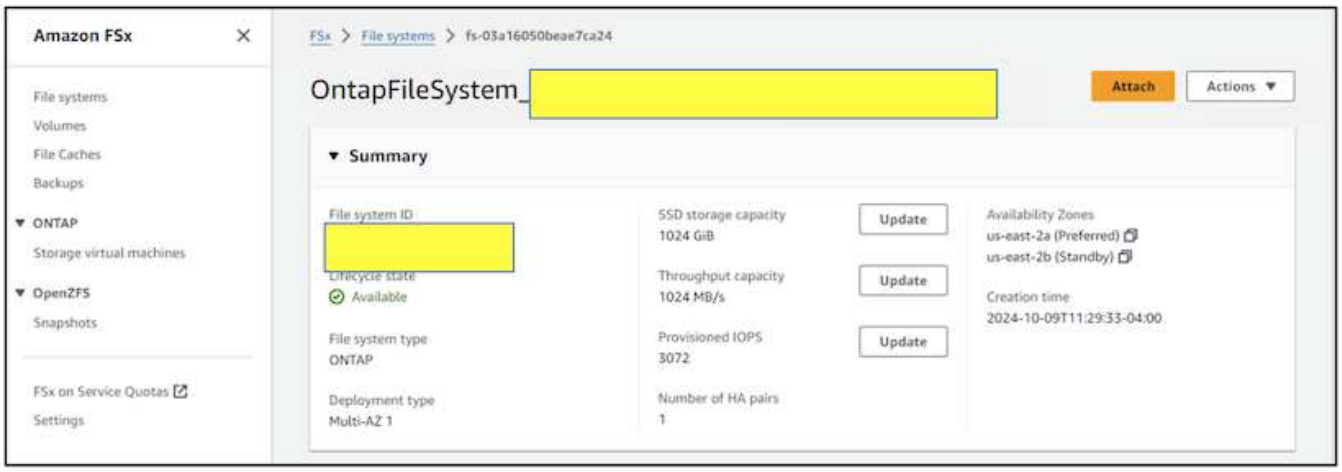
- b. CloudFormation Stack을 실행합니다. **매개 변수 값을 사용자 고유의 값으로 바꾸어 아래 명령을 실행합니다.

```
$ 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
```

장소: region-name: ROSA 클러스터가 배포된 지역과 동일 subnet1_ID: FSxN subnet2_ID: FSxN VPC_ID용 대기 서브넷의 id: ROSA 클러스터가 배포된 VPC의 id, routable1_ID: CIDR ONTAP에 대한 라우팅 테이블의 ID: CIDR에 대한 액세스 허용 범위 설정 0.0.0.0/0 또는 적절한 CIDR을 사용하여 모든 트래픽이 FSx for ONTAP의 특정 포트에 액세스하도록 허용할 수 있습니다. 관리자 암호 정의: FSxN에 로그인하기 위한 암호 SVM 암호 정의: 생성할 SVM에 로그인하기 위한 암호 정의

아래와 같이 Amazon FSx 콘솔을 사용하여 파일 시스템 및 SVM(스토리지 가상 머신)이 생성되었는지 확인합니다.



- 2. Rosa 클러스터용 Trident CSI 드라이버를 설치 및 구성합니다**
- a. Trident Helm 리포지토리를 추가합니다**

```
$ helm repo add netapp-trident https://netapp.github.io/trident-helm-chart
```

- b. Helm**을 사용하여 Trident를 설치합니다

```
$ helm install trident netapp-trident/trident-operator --version
100.2406.0 --create-namespace --namespace trident
```

설치하는 버전에 따라 표시된 명령에서 version 매개 변수를 변경해야 합니다. 올바른 버전 번호는 를 "문서화"참조하십시오. Trident를 설치하는 추가 방법은 Trident를 "문서화"참조하십시오.

C. 모든 **Trident Pod**가 실행 중인지 확인합니다

```
[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]#
```

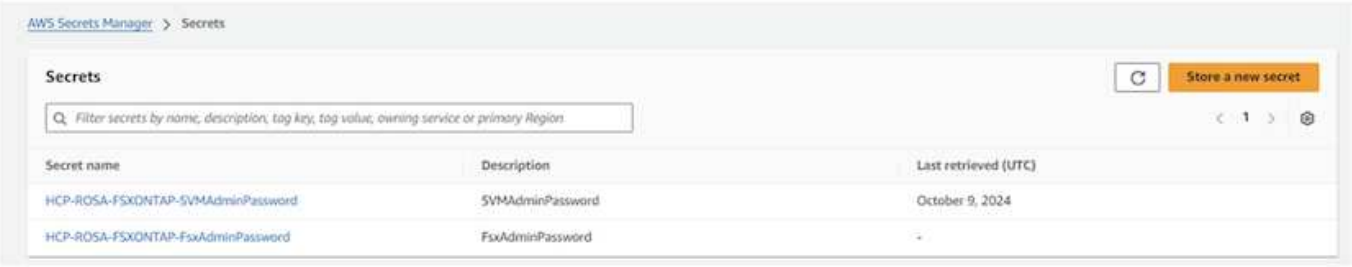
- 3. Trident CSI 백엔드에서 FSx for ONTAP(ONTAP NAS)** 를 사용하도록 구성합니다

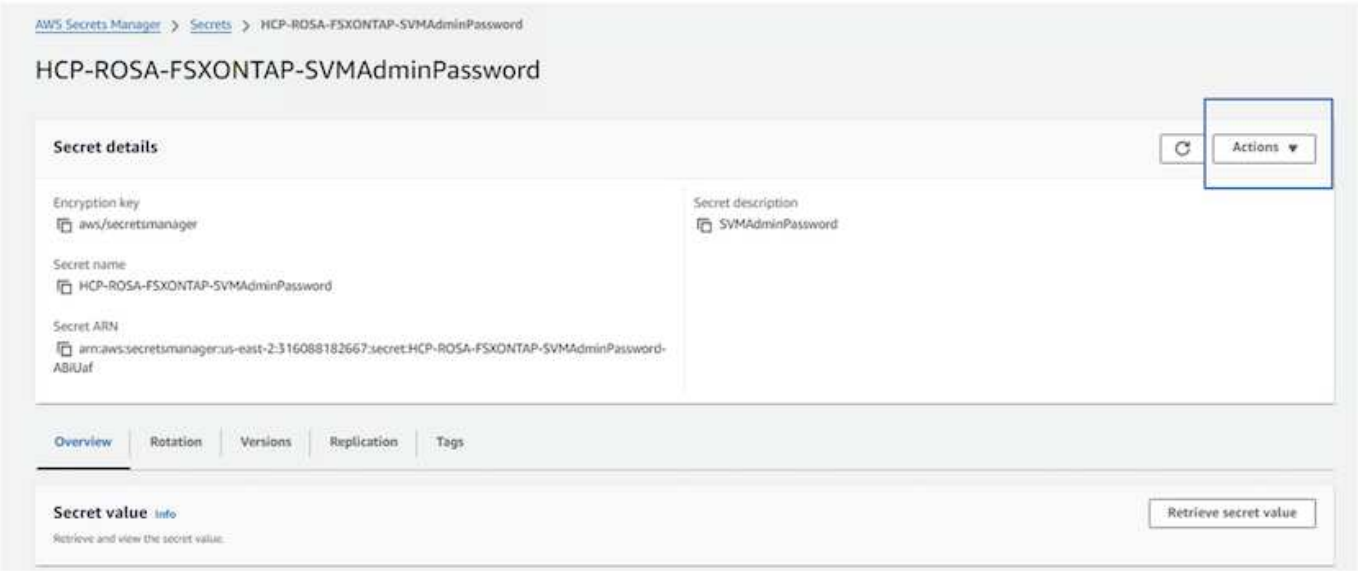
Trident 백 엔드 구성은 Trident에 스토리지 시스템과 통신하는 방법을 알려줍니다(이 경우에는 FSx for ONTAP). 백엔드를 생성하기 위해 클러스터 관리 및 NFS 데이터 인터페이스와 함께 연결할 스토리지 가상 시스템의 자격 증명을 제공합니다. 을 사용하여 "ONTAP - NAS 드라이버"FSx 파일 시스템에서 스토리지 볼륨을 프로비저닝합니다.

- a. 먼저 다음 YAML** 을 사용하여 SVM 자격 증명에 대한 암호를 만듭니다

```
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>
```

아래와 같이 AWS Secrets Manager에서 FSxN용으로 생성된 SVM 암호를 검색할 수도 있습니다.





- b. 다음 명령을 사용하여 ROSA 클러스터에 SVM 자격 증명에 대한 암호를 추가합니다**

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

다음 명령을 사용하여 Trident 네임스페이스에 암호가 추가되었는지 확인할 수 있습니다

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

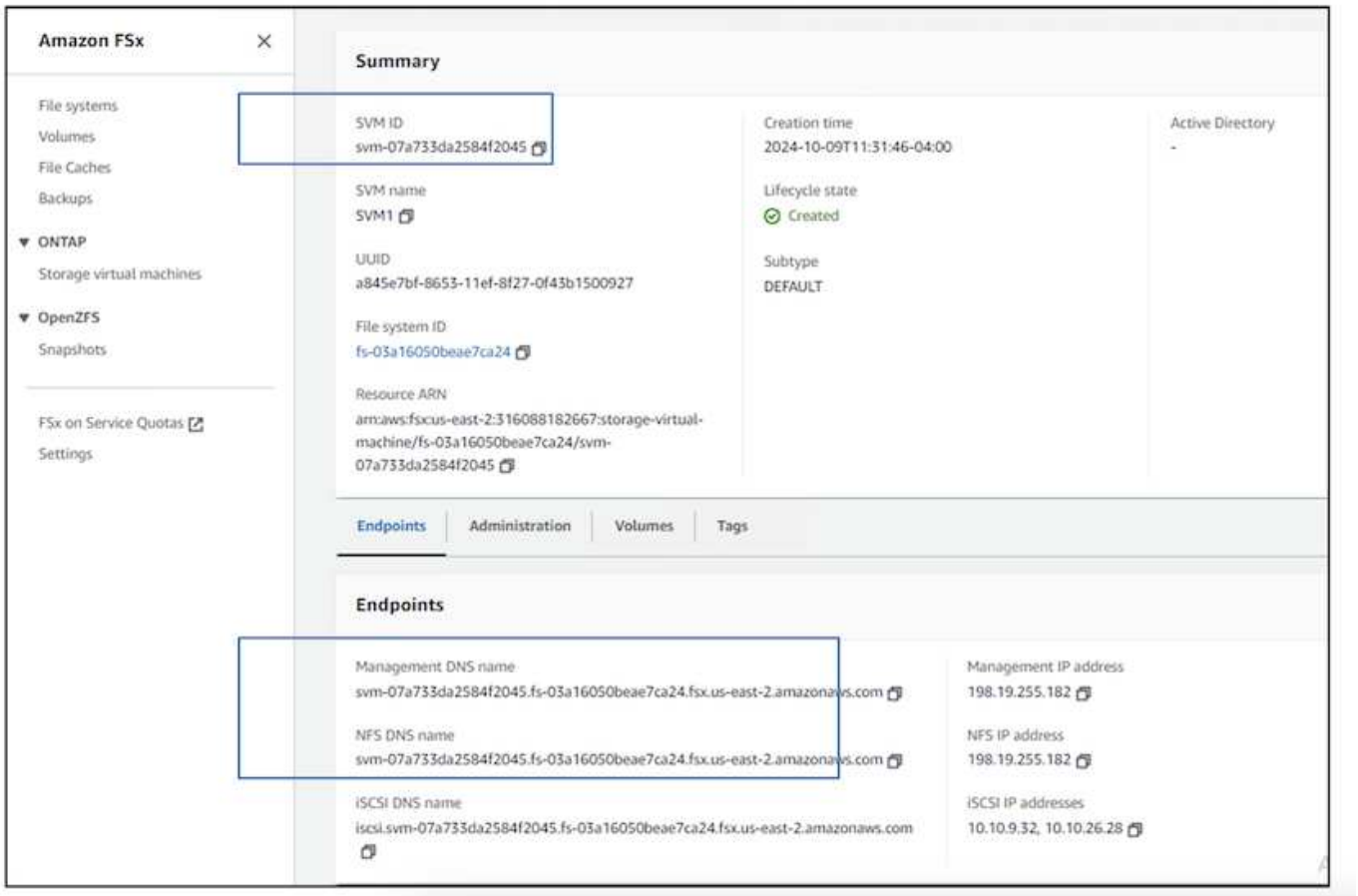
```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# oc get secrets -n trident | grep backend-fsx-ontap-nas-secret
backend-fsx-ontap-nas-secret         Opaque                  2      21h
[root@localhost hcp-testing]#
```

- C. 그런 다음 백엔드 객체를 생성하고, 복제된 **Git** 리포지토리의 FSX 디렉토리로 이동합니다**. Backend-ONTAP-nas.yaml** 파일을 엽니다**.** 다음 항목을 관리 **DNS** 이름 dataLIF 로**, Amazon FSx SVM**의 **NFS DNS** 이름으로**,** svm** 을 SVM 이름으로 바꿉니다. 다음 명령을 사용하여 백엔드 객체를 생성합니다.

다음 명령을 사용하여 백엔드 객체를 생성합니다.

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

아래 스크린샷에 표시된 것처럼 Amazon FSx 콘솔에서 관리 DNS 이름, NFS DNS 이름 및 SVM 이름을 확인할 수 있습니다



- d. 이제 다음 명령을 실행하여 백엔드 객체가 생성되었고 Phase가 Bound로 표시되고 Status가 Success** 인지 확인합니다

```
[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. 스토리지 클래스 생성** 이제 Trident 백엔드가 구성되었으므로 백엔드를 사용할 Kubernetes 스토리지 클래스를 생성할 수 있습니다. 스토리지 클래스는 클러스터에서 사용할 수 있는 리소스 개체입니다. 애플리케이션에 대해 요청할 수 있는 스토리지 유형을 설명하고 분류합니다.
- a. FSX 폴더에서 storage-class-csi-nas.yaml 파일을 검토합니다.**

```
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 클러스터에서 스토리지 클래스를 생성하고 Trident-CSI 스토리지 클래스가 생성되었는지 확인합니다.**

```
[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]#
```

Trident CSI 드라이버 설치 및 FSx for ONTAP 파일 시스템에 대한 연결이 완료되었습니다. 이제 FSx for ONTAP의 파일 볼륨을 사용하여 ROSA에 샘플 PostgreSQL 상태 저장 애플리케이션을 배포할 수 있습니다.

- C. Trident-CSI 스토리지 클래스를 사용하여 생성된 PVC 및 PVS가 없는지 확인합니다.**

- d. 응용 프로그램이 Trident CSI를 사용하여 PV를 생성할 수 있는지 확인합니다.**

FSX** 폴더에 제공된 PVC-Trident.yaml 파일을 사용하여 PVC를 생성합니다.

```
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_image11.png["Trident를 사용하여 테스트
PVC를 생성합니다"]
```

- 5. 샘플 PostgreSQL 상태 저장 응용 프로그램을 배포합니다**
- a. helm을 사용하여 PostgreSQL**를 설치합니다

```
$ 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. 응용 프로그램 포드가 실행 중이고 응용 프로그램에 대해 PVC 및 PV가 생성되었는지 확인합니다.**

```
[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 클라이언트 배포**

다음 명령을 사용하여 설치된 **PostgreSQL** 서버의 암호를 가져옵니다**.**

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

다음 명령을 사용하여 **PostgreSQL** 클라이언트를 실행하고 암호를 사용하여 서버에 연결합니다

```
$ 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. 데이터베이스 및 테이블 만들기 테이블에 대한 스키마를 만들고 테이블에 두 개의 데이터 행을 삽입합니다.**

```
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 ONTAP가 포함된 AWS 기반 Red Hat OpenShift Service

이 문서에서는 NetApp ONTAP를 AWS(ROSA)에서 Red Hat OpenShift Service와 함께 사용하는 방법에 대해 설명합니다.

볼륨 스냅샷을 생성합니다

- 1. 앱 볼륨 스냅샷 만들기** 이 섹션에서는 앱과 연결된 볼륨의 Trident 스냅샷을 만드는 방법을 보여 줍니다. 이것은 앱 데이터의 특정 시점 복제본입니다. 애플리케이션 데이터가 손실되면 이 시점 복사본에서 데이터를 복구할 수 있습니다. 참고: 이 스냅샷은 ONTAP(사내 또는 클라우드)의 원래 볼륨과 동일한 애그리게이트에 저장됩니다. 따라서 ONTAP 스토리지 Aggregate가 손실되면 해당 스냅샷에서 앱 데이터를 복구할 수 없습니다.
- a. VolumeSnapshotClass 생성 volume-snapshot-class.yaml이라는 파일에 다음 매니페스트를 저장합니다

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

위의 매니페스트를 사용하여 스냅샷을 만듭니다.

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

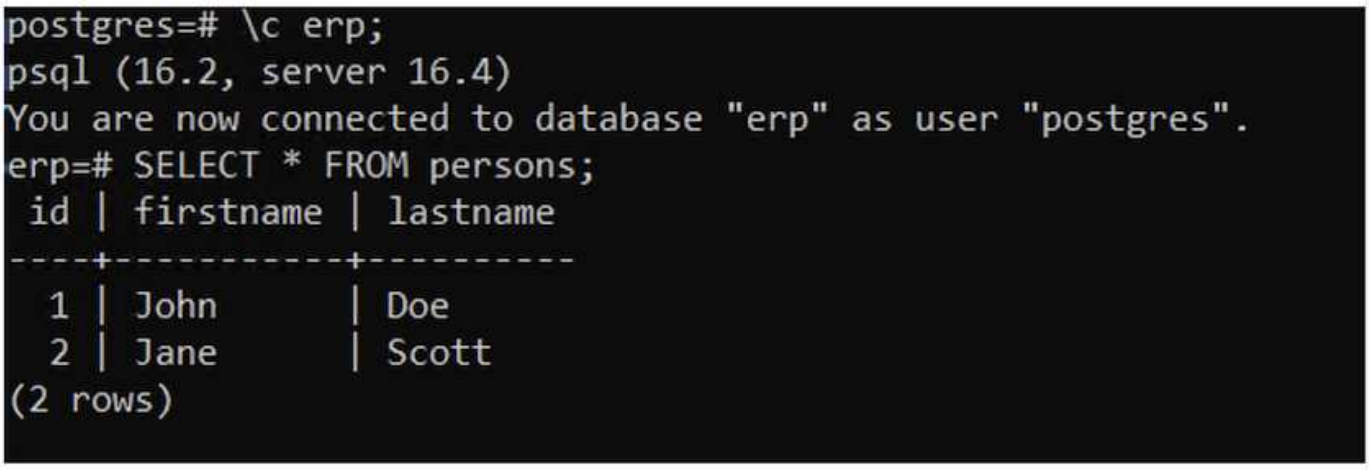
- b. 그런 다음 스냅샷을 생성합니다** VolumeSnapshot을 생성하여 PostgreSQL 데이터의 시점 복제본을 생성하여 기존 PVC의 스냅샷을 생성합니다. 이렇게 하면 파일 시스템 백엔드에서 공간을 거의 차지하지 않는 FSx 스냅샷이 생성됩니다. 다음 매니페스트를 volume-snapshot.yaml이라는 파일에 저장합니다.

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

- C. 볼륨 스냅샷을 생성하고 볼륨 스냅샷이 생성되었는지 확인합니다**

데이터 손실을 시뮬레이션하기 위해 데이터베이스를 삭제합니다(데이터 손실은 다양한 이유로 발생할 수 있습니다. 여기서는 데이터베이스를 삭제하여 시뮬레이션합니다.)

- d. 데이터 손실을 시뮬레이션하기 위해 데이터베이스를 삭제합니다(데이터 손실은 다양한 이유로 발생할 수 있습니다. 여기서는 데이터베이스를 삭제하여 시뮬레이션합니다.)**



볼륨 스냅샷에서 복원합니다

- 1. 스냅샷에서 복원** 이 섹션에서는 앱 볼륨의 Trident 스냅샷에서 응용 프로그램을 복원하는 방법을 보여 줍니다.
- a. 스냅샷에서 볼륨 클론을 생성합니다**

볼륨을 이전 상태로 복원하려면 스냅샷의 데이터를 기반으로 새 PVC를 생성해야 합니다. 이렇게 하려면 다음

매니페스트를 pvc-clone.yaml이라는 파일에 저장합니다

```
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
```

위의 매니페스트를 사용하여 스냅샷을 소스로 사용하여 PVC를 생성하여 볼륨의 클론을 생성합니다. 매니페스트를 적용하고 클론이 생성되었는지 확인합니다.

```
[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. 원래 PostgreSQL 설치를 삭제합니다**

```
[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. 새 클론 PVC**를 사용하여 새 PostgreSQL 애플리케이션을 생성합니다

```
$ 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. 응용 프로그램 포드가 실행 중인지 확인합니다**

```
[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]#
```

- 예 포드가 클론을 PVC**로 사용하는지 확인합니다

```
[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) 데이터베이스가 예상대로 복원되었는지 확인하려면 컨테이너 콘솔로 돌아가 기존 데이터베이스를 표시합니다

```
[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)
```

데모 비디오

Amazon FSx for NetApp ONTAP은 호스팅된 Control Plane을 사용하는 AWS 기반 Red Hat OpenShift Service를 제공합니다

Red Hat OpenShift 및 OpenShift 솔루션에 대한 더 많은 비디오를 찾을 수 "여기"있습니다.