Red hat OpenShift Service auf AWS mit FSxN

NetApp Solutions

NetApp
December 19, 2024

Inhalt

Red hat OpenShift Service auf AWS mit FSxN

Red hat OpenShift Service auf AWS mit NetApp ONTAP

Überblick

In diesem Abschnitt zeigen wir, wie FSX für ONTAP als persistente Storage-Ebene für auf ROSA ausgeführte Applikationen genutzt werden kann. Es zeigt die Installation des NetApp Trident-CSI-Treibers auf einem ROSA-Cluster, die Bereitstellung eines FSX für ONTAP-Dateisystems und die Bereitstellung einer Beispielanwendung. Es zeigt auch Strategien für die Sicherung und Wiederherstellung Ihrer Anwendungsdaten auf. Mit dieser integrierten Lösung können Sie ein Shared Storage Framework entwickeln, das sich mühelos über alle Verfügbarkeitszonen hinweg skalieren lässt. Dadurch werden die Skalierungs-, Sichern- und Wiederherstellungsprozesse mit dem Trident CSI-Treiber vereinfacht.

Voraussetzungen

- "AWS Konto"
- "Ein Red hat Konto"
- IAM-Benutzer "Mit entsprechenden Berechtigungen" zum Erstellen und Zugreifen auf ROSA-Cluster
- "AWS CLI"
- "ROSA CLI"
- "OpenShift -Befehlszeilenschnittstelle" (oc)
- Helm 3 "Dokumentation"
- "EIN HCP-ROSA-CLUSTER"
- "Zugriff auf die Red hat OpenShift -Webkonsole"

Dieses Diagramm zeigt den in mehreren AZS bereitgestellten ROSA-Cluster. Master-Nodes des ROSA Clusters, Infrastruktur-Nodes sind in der VPC von Red hat, während sich die Worker-Nodes in einer VPC im Konto des Kunden befinden. Wir werden ein FSX für ONTAP-Dateisystem innerhalb der gleichen VPC erstellen und den Trident-Treiber im ROSA-Cluster installieren, damit alle Subnetze dieser VPC mit dem Dateisystem verbinden können.

Ersteinrichtung

1. Bereitstellung FSX für NetApp ONTAP

Erstellen Sie ein Multi-AZ FSX für NetApp ONTAP in demselben VPC wie das ROSA-Cluster. Es gibt mehrere Möglichkeiten, dies zu tun. Die Details zur Erstellung von FSxN mit einem CloudFormation Stack werden bereitgestellt

A.Klonen Sie das GitHub Repository

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

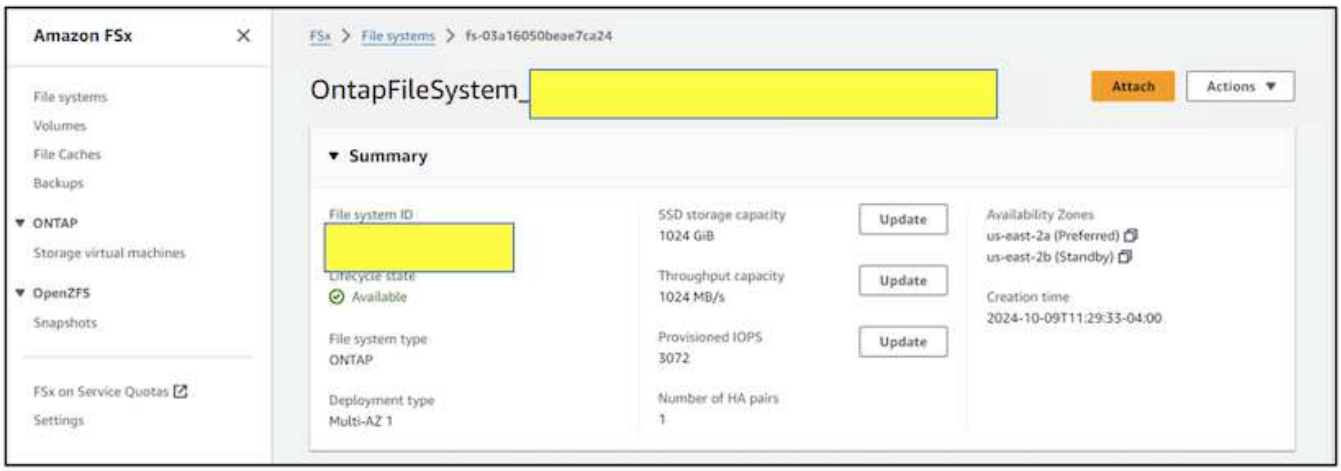
B.Starten Sie den CloudFormation Stack Führen Sie den folgenden Befehl aus, indem Sie die Parameterwerte durch Ihre eigenen Werte ersetzen:

```
$ 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
```

Wobei : Region-Name: Identisch mit der Region, in der der ROSA-Cluster bereitgestellt wird subnet1_ID : id des bevorzugten Subnetzes für FSxN subnet2_ID: id des Standby-Subnetzes für FSxN VPC_ID: id des VPC, in dem der ROSA-Cluster bereitgestellt wird routetable1_ID, routetable2_ID: ids der Routingtabellen, die mit den oben gewählten Subnetzen für die-Zugriffsregeln für den Zugriff auf die ONTAP-Gruppen zugeordnet sind. Sie können 0.0.0.0/0 oder jede geeignete CIDR verwenden, um allen Verkehr zu erlauben, auf die spezifischen Ports von FSX für ONTAP zuzugreifen. Administrator-Passwort definieren: Ein Passwort für die Anmeldung bei FSxN SVM-Passwort definieren: Ein Passwort für die Anmeldung bei SVM, die erstellt werden soll.

Überprüfen Sie mithilfe der Amazon FSX Konsole, ob Ihr Filesystem und Ihre Storage Virtual Machine (SVM) erstellt wurden:



2.Installieren und Konfigurieren des Trident CSI-Treibers für den ROSA-Cluster

A.Hinzufügen des Trident-Helm-Repository

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

B.Installieren Sie Trident mithilfe von Helm

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

Abhängig von der Version, die Sie installieren, muss der Versionsparameter im angezeigten Befehl geändert werden. Die korrekte Versionsnummer finden Sie im"Dokumentation". Weitere Informationen zur Installation von Trident finden Sie im Trident "Dokumentation".

C.Überprüfen Sie, ob sich alle Trident-Pods im laufenden Zustand befinden

```
[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]#
```

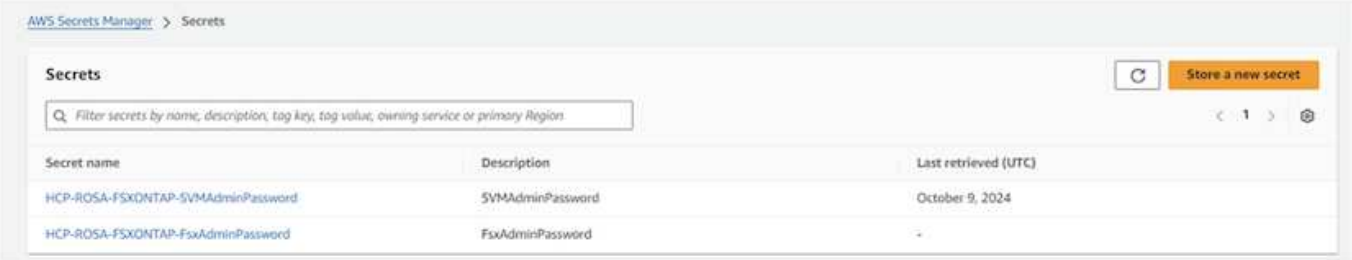
3. Konfigurieren Sie das Trident CSI-Backend für die Verwendung von FSX for ONTAP (ONTAP NAS)

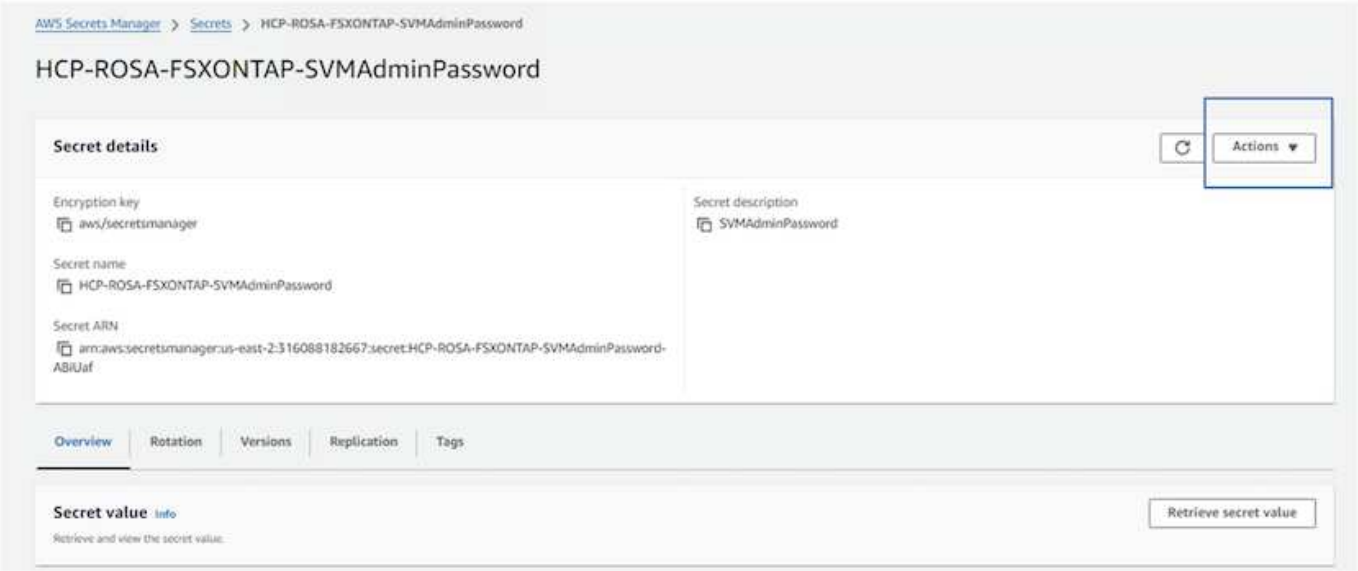
Die Trident Back-End-Konfiguration sagt Trident über die Kommunikation mit dem Storage-System (in diesem Fall FSX für ONTAP). Für die Erstellung des Backends stellen wir die Anmeldeinformationen der zu verbindenen Storage Virtual-Maschine sowie die Cluster-Management- und NFS-Datenschnittstellen bereit. Wir werden die verwenden"ontap-nas-Treiber", um Speicher Volumen im FSX Dateisystem bereitzustellen.

A. Erstellen Sie zunächst einen Schlüssel für die SVM-Anmeldeinformationen mit der folgenden yaml

```
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>
```

Das für FSxN erstellte SVM-Passwort können Sie wie unten gezeigt im AWS Secrets Manager abrufen.





B.als Nächstes fügen Sie den Schlüssel für die SVM-Anmeldeinformationen mit dem folgenden Befehl zum ROSA-Cluster hinzu

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

Mit dem folgenden Befehl können Sie überprüfen, ob der Geheimschlüssel im Trident-Namespace hinzugefügt wurde

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

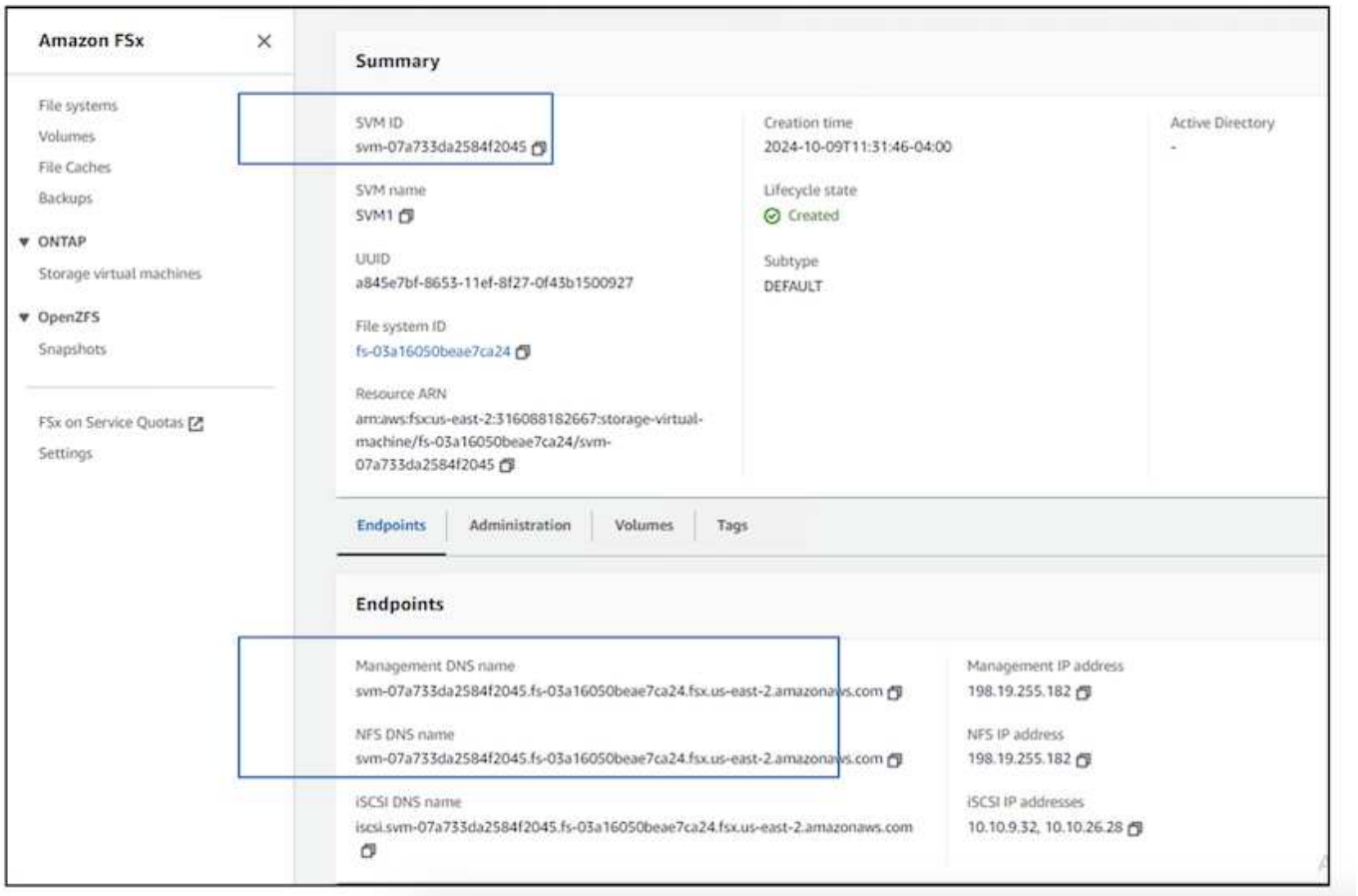
```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# oc get secrets -n trident | grep backend-fsx-ontap-nas-secret
backend-fsx-ontap-nas-secret         Opaque                  2      21h
[root@localhost hcp-testing]#
```

c. Erstellen Sie als nächstes das Backend-Objekt dafür, gehen Sie in das **fsx** Verzeichnis Ihres geklonten Git-Repository. Öffnen Sie die Datei Backend-ONTAP-nas.yaml. Ersetzen Sie folgendes: **ManagementLIF** mit dem Management DNS-Namen **dataLIF** mit dem NFS DNS-Namen der Amazon FSX svm und **svm** mit dem svm-Namen. Erstellen Sie das Backend-Objekt mit dem folgenden Befehl.

Erstellen Sie das Backend-Objekt mit dem folgenden Befehl.

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

Wie in der Abbildung unten gezeigt, erhalten Sie den Management-DNS-Namen, den NFS-DNS-Namen und den SVM-Namen von der Amazon FSX-Konsole



D. Führen Sie nun den folgenden Befehl aus, um zu überprüfen, ob das Backend-Objekt erstellt wurde und Phase „gebunden“ und Status „erfolgreich“ anzeigt.

```
[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. Storage Class erstellen Nachdem nun das Trident-Backend konfiguriert ist, können Sie eine Kubernetes-Storage-Klasse erstellen, um das Backend zu verwenden. Storage-Klasse ist ein Ressourcenobjekt, das dem Cluster zur Verfügung gestellt wird. Es beschreibt und klassifiziert den Speichertyp, den Sie für eine Anwendung anfordern können.

A. Überprüfen Sie die Datei Storage-class-csi-nas.yaml im fsx-Ordner.

```
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. Erstellen Sie eine Storage-Klasse im ROSA-Cluster, und überprüfen Sie, ob die Trident-csi-Storage-Klasse erstellt wurde.

```
[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]#
```

Damit ist die Installation des Trident-CSI-Treibers und dessen Anbindung an das Dateisystem FSX for ONTAP abgeschlossen. Jetzt können Sie eine Beispielanwendung für PostgreSQL Stateful auf ROSA mit Dateivolumes auf FSX für ONTAP implementieren.

c. Vergewissern Sie sich, dass keine VES und VES mit der Trident-csi-Storage-Klasse erstellt wurden.

```
[root@localhost hcp-testing]#
[root@localhost hcp-testing]#
[root@localhost hcp-testing]# oc get pvc -A
NAMESPACE                       NAME                                STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
openshift-monitoring            prometheus-data-prometheus-k8s-0    Bound    pvc-9a4553a5-07e9-440a-8a90-99e384c97624   100Gi      RWO            gp3-csi        <unset>                 2d16h
openshift-monitoring            prometheus-data-prometheus-k8s-1    Bound    pvc-7d949aef-e00d-4d9a-8b54-514ed85fbab2   100Gi      RWO            gp3-csi        <unset>                 2d16h
openshift-virtualization-os-images   centos-stream9-bae111cdd5a1    Bound    pvc-deb61444-cb3f-449b-8d7d-39d028496c16   30Gi       RWO            gp3-csi        <unset>                 24h
openshift-virtualization-os-images   centos-stream9-d82f4a141a44    Bound    pvc-82b0e84a-e5ef-452b-bf90-1eae4fe162c1   30Gi       RWO            gp3-csi        <unset>                 44h
openshift-virtualization-os-images   fedora-21a6f3e628cd            Bound    pvc-64f375ad-d377-456d-83a0-368e413ae79c   30Gi       RWO            gp3-csi        <unset>                 44h
openshift-virtualization-os-images   rhel8-0b52df0eb259             Bound    pvc-2dc6de48-5916-411e-9cb3-99598f50be4c   30Gi       RWO            gp3-csi        <unset>                 44h
openshift-virtualization-os-images   rhel9-2521bd116e64             Bound    pvc-f4374ce7-568d-4afc-b635-0228cf4544d4   30Gi       RWO            gp3-csi        <unset>                 44h
[root@localhost hcp-testing]# oc get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                                                                  STORAGECLASS   VOLUMEATTRIBUTESCLASS
pvc-2dc6de48-5916-411e-9cb3-99598f50be4c   30Gi       RWO            Delete           Bound    openshift-virtualization-os-images/rhel8-0b52df0eb259                  gp3-csi        <unset>
pvc-64f375ad-d377-456d-83a0-368e413ae79c   30Gi       RWO            Delete           Bound    openshift-virtualization-os-images/fedora-21a6f3e628cd                 gp3-csi        <unset>
pvc-7d949aef-e00d-4d9a-8b54-514ed85fbab2   100Gi      RWO            Delete           Bound    openshift-monitoring/prometheus-data-prometheus-k8s-1                  gp3-csi        <unset>

pvc-82b0e84a-e5ef-452b-bf90-1eae4fe162c1   30Gi       RWO            Delete           Bound    openshift-virtualization-os-images/centos-stream9-d82f4a141a44         gp3-csi        <unset>
pvc-9a4553a5-07e9-440a-8a90-99e384c97624   100Gi      RWO            Delete           Bound    openshift-monitoring/prometheus-data-prometheus-k8s-0                  gp3-csi        <unset>

pvc-deb61444-cb3f-449b-8d7d-39d028496c16   30Gi       RWO            Delete           Bound    openshift-virtualization-os-images/centos-stream9-bae111cdd5a1         gp3-csi        <unset>
pvc-f4374ce7-568d-4afc-b635-0228cf4544d4   30Gi       RWO            Delete           Bound    openshift-virtualization-os-images/rhel9-2521bd116e64                  gp3-csi        <unset>
[root@localhost hcp-testing]#
```

D. Überprüfen Sie, ob Anwendungen PV mit Trident CSI erstellen können.

Erstellen Sie eine PVC mit der Datei pvc-Trident.yaml, die im Ordner **fsx** enthalten ist.

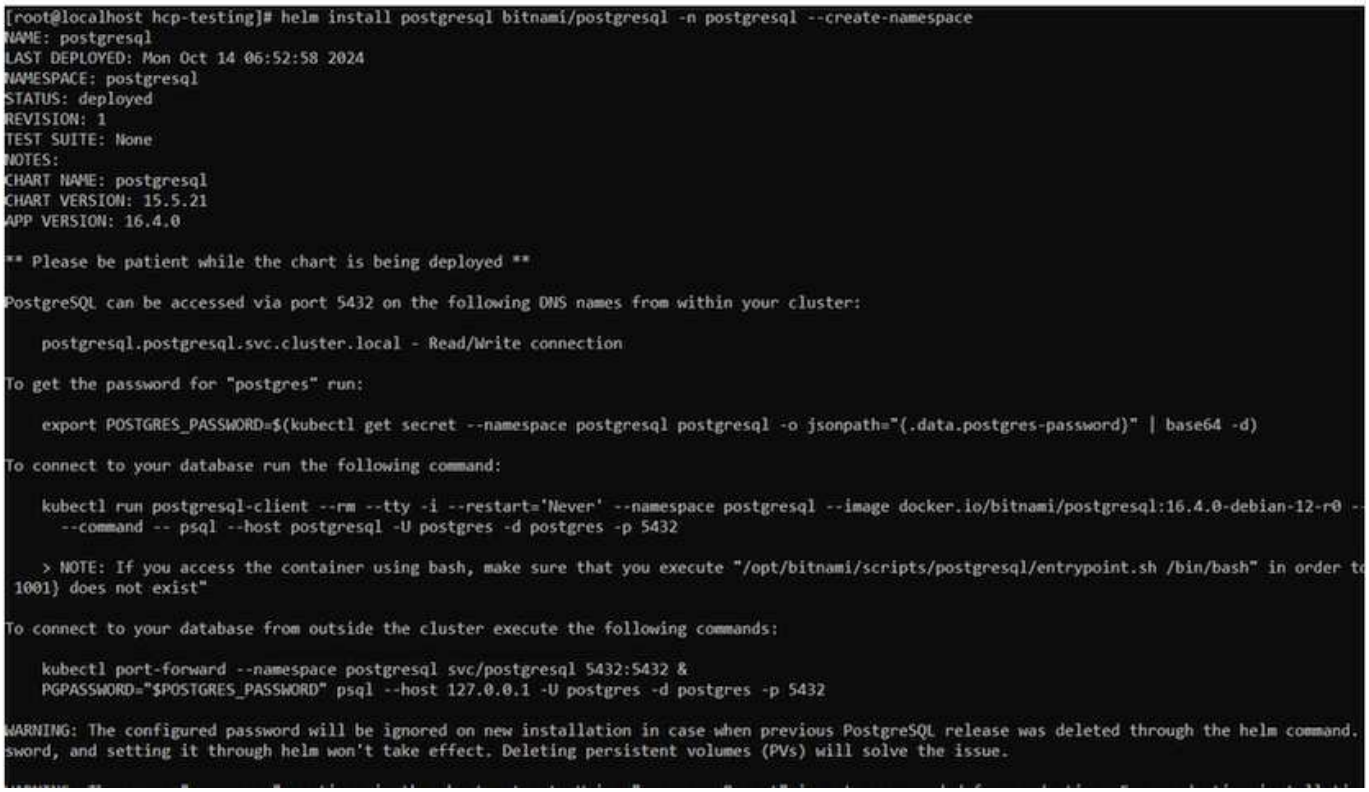
```
pvc-trident.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: basic
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  storageClassName: trident-csi
```

```
 You can issue the following commands to create a pvc and verify that it
has been created.
image:redhat_openshift_container_rosa_image11.png["Test-PVC mit Trident
erstellen"]
```

5. Stellen Sie eine Beispielanwendung für PostgreSQL Stateful bereit

A. Verwenden Sie Helm, um postgresql zu installieren

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



B. Überprüfen Sie, ob der Anwendungspod ausgeführt wird und eine PVC und ein PV für die Anwendung erstellt werden.

c. PostgreSQL-Client implementieren

Verwenden Sie den folgenden Befehl, um das Passwort für den postgresql Server zu erhalten, der installiert wurde.

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

Verwenden Sie den folgenden Befehl, um einen postgresql-Client auszuführen und mit dem Passwort eine Verbindung zum Server herzustellen

```
$ 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. Erstellen Sie eine Datenbank und eine Tabelle. Erstellen Sie ein Schema für die Tabelle und fügen Sie 2 Datenzeilen in die Tabelle ein.

```
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)
```

Red hat OpenShift Service auf AWS mit NetApp ONTAP

In diesem Dokument wird die Verwendung von NetApp ONTAP mit dem Red hat OpenShift Service on AWS (ROSA) beschrieben.

Erstellen Sie Einen Volume-Snapshot

1. Erstellen Sie einen Snapshot des App-Volumes in diesem Abschnitt wird gezeigt, wie Sie einen Trident-Snapshot des mit der App verknüpften Volumes erstellen.Dies ist eine Point-in-Time-Kopie der App-Daten. Falls die Applikationsdaten verloren gehen, können wir die Daten von dieser zeitpunktgenaue Kopie wiederherstellen. HINWEIS: Dieser Snapshot wird im selben Aggregat wie das ursprüngliche Volume in ONTAP gespeichert (On-Premises oder in der Cloud). Wenn also das ONTAP Storage-Aggregat verloren geht, können wir die Applikationsdaten nicht aus dem Snapshot wiederherstellen.

**A. Erstellen einer VolumeSnapshotClass Speichern Sie das folgende Manifest in einer Datei namens Volume-Snapshot-class.yaml

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

Erstellen Sie mithilfe des oben genannten Manifests einen Snapshot.

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

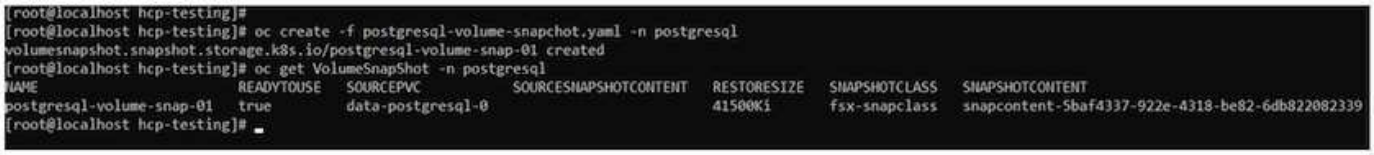
B. Erstellen Sie anschließend einen Snapshot Erstellen Sie einen Snapshot der vorhandenen PVC, indem Sie VolumeSnapshot erstellen, um eine Point-in-Time-Kopie Ihrer PostgreSQL-Daten zu erstellen. Dies erzeugt einen FSX Snapshot, der fast keinen Platz im Dateisystem-Backend beansprucht. Speichern Sie das

folgende Manifest in einer Datei namens Volume-Snapshot.yaml:

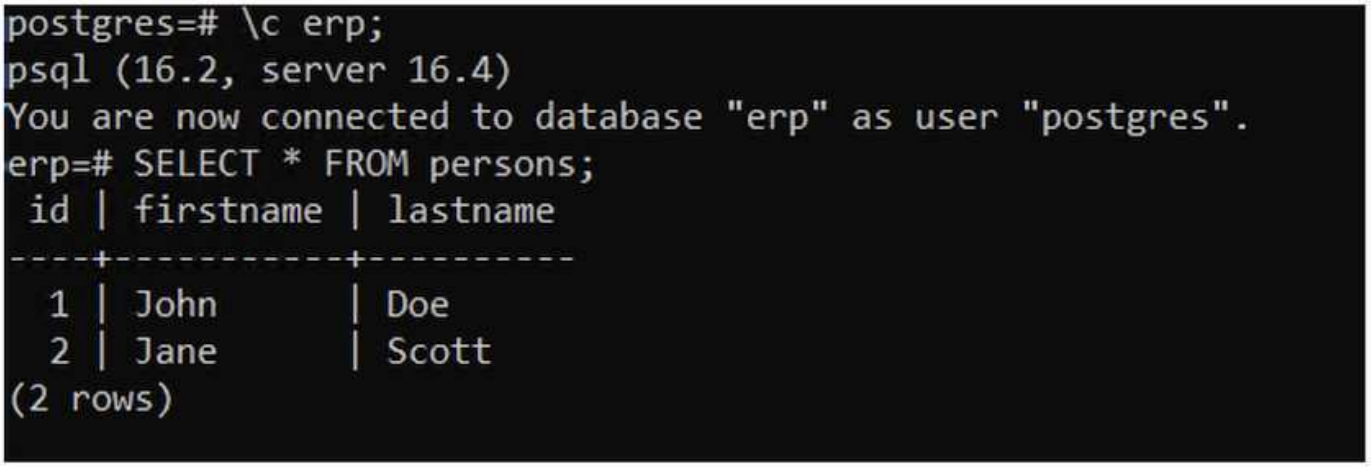
```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
 name: postgresql-volume-snap-01
spec:
 volumeSnapshotClassName: fsx-snapclass
 source:
   persistentVolumeClaimName: data-postgresql-0
```

c. Erstellen Sie den Volume-Snapshot und bestätigen Sie, dass er erstellt wurde

Löschen Sie die Datenbank, um den Verlust von Daten zu simulieren (Datenverlust kann aus einer Vielzahl von Gründen passieren, hier simulieren wir es einfach durch Löschen der Datenbank)



D. Löschen Sie die Datenbank, um den Verlust von Daten zu simulieren (Datenverlust kann aus verschiedenen Gründen passieren, hier simulieren wir sie einfach durch Löschen der Datenbank)



Wiederherstellen aus Volume Snapshot

1. Wiederherstellung aus Snapshot in diesem Abschnitt zeigen wir, wie eine Anwendung aus dem Trident-Snapshot des App-Volumes wiederhergestellt werden kann.

A. Erstellen Sie einen Volume-Klon aus dem Snapshot

Um den vorherigen Zustand des Volumes wiederherzustellen, müssen Sie eine neue PVC auf der Grundlage der Daten in dem Snapshot erstellen, den Sie erstellt haben. Speichern Sie dazu das folgende Manifest in einer Datei namens 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
```

Erstellen Sie einen Klon des Volumes, indem Sie mithilfe des oben genannten Manifests eine PVC mithilfe des Snapshots als Quelle erstellen. Wenden Sie das Manifest an, und stellen Sie sicher, dass der Klon erstellt wird.

```
[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. Löschen Sie die ursprüngliche postgresql-Installation

```
[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. Erstellen Sie eine neue postgresql-Anwendung mit dem neuen 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. Stellen Sie sicher, dass der Anwendungs-POD den Status läuft aufweist

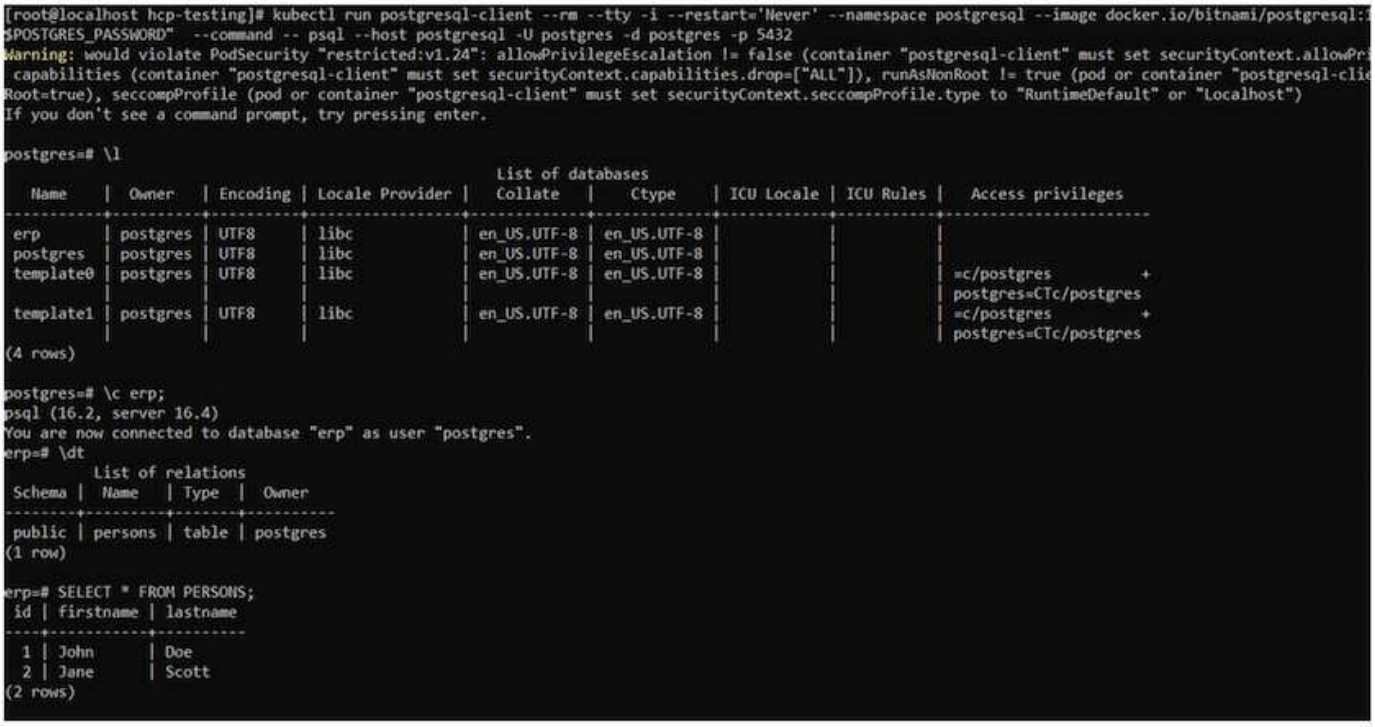
```
[root@localhost hcp-testing]# oc get pods -n postgresql
NAME           READY   STATUS    RESTARTS   AGE
postgresql-0   1/1     Running   0          2m1s
[root@localhost hcp-testing]#
```

E. Vergewissern Sie sich, dass der Pod den Klon als PVC verwendet

```
[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) um zu überprüfen, ob die Datenbank wie erwartet wiederhergestellt wurde, gehen Sie zurück zur Container-Konsole und zeigen Sie die vorhandenen Datenbanken an



Demovideo

Amazon FSX for NetApp ONTAP: Red hat OpenShift Service auf AWS mit gehosteter Kontrollebene

Weitere Videos zu Red hat OpenShift- und OpenShift-Lösungen finden Sie "Hier".