vSphere Kubernetes Service mit NetApp Storage

NetApp container solutions

NetApp
September 01, 2026

Inhalt

vSphere Kubernetes Service mit NetApp Storage

Informationen zum VMware vSphere Kubernetes Service mit NetApp ONTAP

VMware vSphere Kubernetes Service (VKS) ist ein Managed-Kubernetes-Angebot von VMware, das es Unternehmen ermöglicht, containerisierte Anwendungen mit Kubernetes bereitzustellen, zu verwalten und zu skalieren. In Kombination mit NetApp ONTAP Speicher bietet VKS Persistenz, Leistung und Datenmanagement der Enterprise-Klasse für Cloud-native Workloads.

VKS ist in VMware Cloud Foundation (VCF) eingebettet und entspricht dem Upstream CNCF Kubernetes, wodurch die Kompatibilität mit dem breiteren Kubernetes-Ökosystem und den zugehörigen Tools gewährleistet wird.

Hauptmerkmale des vSphere Kubernetes Service

1. **Verwaltete Kubernetes-Cluster:** vSphere Kubernetes Service vereinfacht die Bereitstellung und Verwaltung von Kubernetes-Clustern. Er automatisiert Aufgaben wie Clustererstellung, Upgrades, Skalierung und Überwachung.
2. **Integration mit der VMware-Infrastruktur:** VKS integriert sich nahtlos in VMware vSphere, NSX-T und andere VMware Lösungen, sodass Unternehmen ihre bestehende VMware Infrastruktur für Kubernetes Workloads nutzen können.
3. **Multi-Cloud und Hybrid Cloud Support:** vSphere Kubernetes Service unterstützt die Bereitstellung in privaten Clouds, öffentlichen Clouds (z. B. AWS, Azure, Google Cloud) und Hybrid Cloud-Umgebungen, was Unternehmen Flexibilität bei der Verwaltung ihrer Anwendungen bietet.
4. **Integrierte Sicherheit:** VKS beinhaltet Sicherheitsfunktionen wie rollenbasierte Zugriffssteuerung (RBAC), Richtliniendurchsetzung und Integration mit VMware NSX für Netzwerksicherheit.
5. **Unterstützung für das Kubernetes-Ökosystem:** VKS unterstützt die vollständige Upstream-Kubernetes-API und gewährleistet so die Portabilität zwischen verschiedenen Umgebungen sowie die Kompatibilität mit dem gesamten Kubernetes-Ökosystem, einschließlich Tools wie Istio für Service Mesh, Prometheus für Monitoring und anderen CNCF-zertifizierten Tools. Organisationen können Standard-Kubernetes-Kenntnisse und Integrationen direkt auf VKS-Cluster anwenden.
6. **Entwicklerfreundliche Tools:** VKS unterstützt standardmäßige Kubernetes-Entwickler-Workflows, einschließlich CI/CD-Pipelines, GitOps-Praktiken und Tools wie kubectl und Helm, sodass Entwicklungsteams containerisierte Anwendungen erstellen und bereitstellen können, ohne proprietäre Schnittstellen erlernen zu müssen.
7. **Skalierbarkeit und hohe Verfügbarkeit:** VKS bietet Funktionen wie automatische Skalierung und Fehlertoleranz, um sicherzustellen, dass Anwendungen hochverfügbar und performant bleiben.
8. **Beobachtbarkeit und Überwachung:** VKS integriert sich mit standardmäßigen Kubernetes-kompatiblen Überwachungstools, um Echtzeit-Einblicke in die Cluster- und Anwendungsleistung bereitzustellen.

Anwendungsfälle für den vSphere Kubernetes Service

1. **Anwendungsmodernisierung:** Unternehmen können veraltete Anwendungen modernisieren, indem sie diese containerisieren und auf Kubernetes-Clustern bereitstellen, die von VKS verwaltet werden.
2. **DevOps Enablement:** VKS unterstützt DevOps-Workflows durch die Bereitstellung von Automatisierung,

CI/CD-Integration und entwicklerfreundlichen Tools.

3. **Hybrid Cloud Deployments:** Unternehmen können VKS nutzen, um Kubernetes-Cluster in On-Premises- und Cloud-Umgebungen bereitzustellen, was einen konsistenten Betrieb ermöglicht.
4. **Mikroservices-Architektur:** VKS unterstützt die Anwendungsentwicklung auf Mikroservices-Basis und ermöglicht Teams, verteilte Systeme zu erstellen und zu skalieren.

Erste Schritte mit Storage und vSphere Kubernetes Service

Einen vSphere Kubernetes Service Cluster installieren

Ein vSphere Kubernetes Service (VKS) Cluster wird in Ihrer VCF 9.1 Umgebung installiert, und die Worker-Knoten werden mit den passenden Speicherdienstprogrammen für Ihre Workloads konfiguriert.

Über diese Aufgabe

NFS-Dienstprogramme werden automatisch auf den Worker-Knoten des von Ihnen erstellten Clusters installiert. Wenn Worker-Knoten mit installierten iSCSI- oder NVMe-Dienstprogrammen benötigt werden, ist ein benutzerdefiniertes OVA-Image zu erstellen und dieses für die Worker-Knoten zu verwenden. Das benutzerdefinierte Image kann mit Docker erstellt werden, und das `vcf` Befehlszeilentool kann anschließend verwendet werden, um aus diesem Image eine OVA-Datei zu erstellen. Diese OVA-Datei kann in die Content Library in vSphere hochgeladen werden. Bei der Erstellung des VKS-Clusters kann dieses Image aus der Content Library für die Node Pools ausgewählt werden.

Schritte

1. Den VKS Cluster erstellen:

   Sie können einen VKS-Cluster entweder mit dem Standard-Worker-Node-Image oder mit einem benutzerdefinierten Image erstellen. Das Standard-Worker-Node-Image enthält vorinstallierte NFS-Dienstprogramme, während das benutzerdefinierte Image iSCSI- und NVMe-Dienstprogramme enthalten kann. Die folgenden Optionen sind verfügbar:

   - **Nur NAS (Standard):** Ein VKS-Cluster wird mit dem Standard-Worker-Node-Image erstellt, das vorinstallierte NFS-Dienstprogramme enthält.
   - **SAN-fähig (benutzerdefiniertes Image):** Ein benutzerdefiniertes Docker-Image, das iSCSI- und NVMe-Dienstprogramme enthält, erstellen, eine OVA-Datei mit dem `vcf` Befehlszeilentool generieren, die OVA in die vSphere Content Library hochladen und anschließend den VKS-Cluster erstellen, wobei dieses benutzerdefinierte Image für die Node Pools ausgewählt wird.

     ▼ Anweisungen zum Erstellen eines reinen NAS-vSphere Kubernetes Service-Clusters mit dem Standard-Worker-Node-Image anzeigen

     Der VKS-Cluster wird mit dem Standard-Worker-Node-Image erstellt. In den Eingabeaufforderungen werden der Clustername, die Node-Pool-Konfiguration und weitere Einstellungen angegeben.

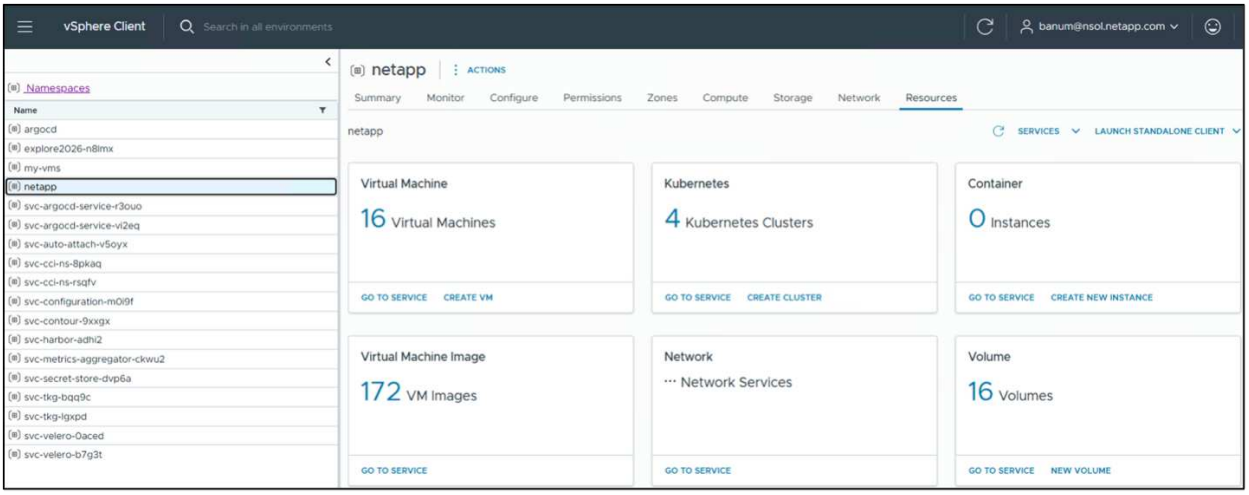
     Im vSphere Client zum Namespace wechseln, in dem der vSphere Kubernetes-Cluster erstellt werden soll. Auf der Registerkarte **Resources** dieses Namespace im Abschnitt Kubernetes auf **Create Cluster** klicken oder auf **Go to Service** und anschließend auf **Create Cluster** klicken.

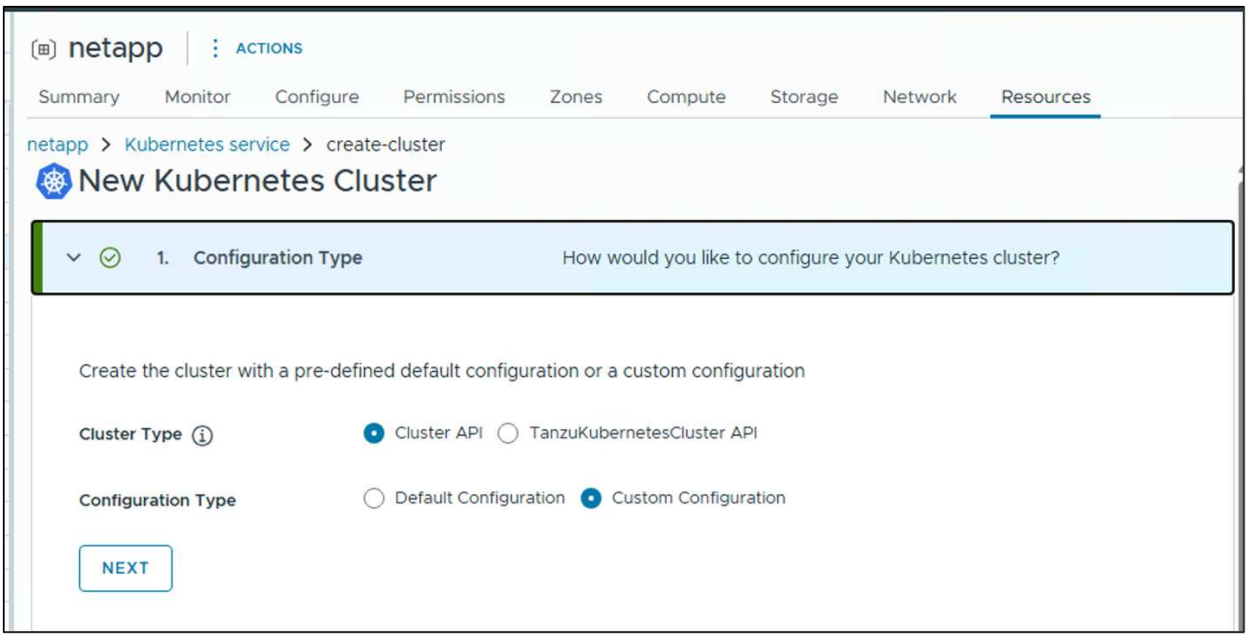
     Das Formular mit den Clusterdetails ausfüllen:

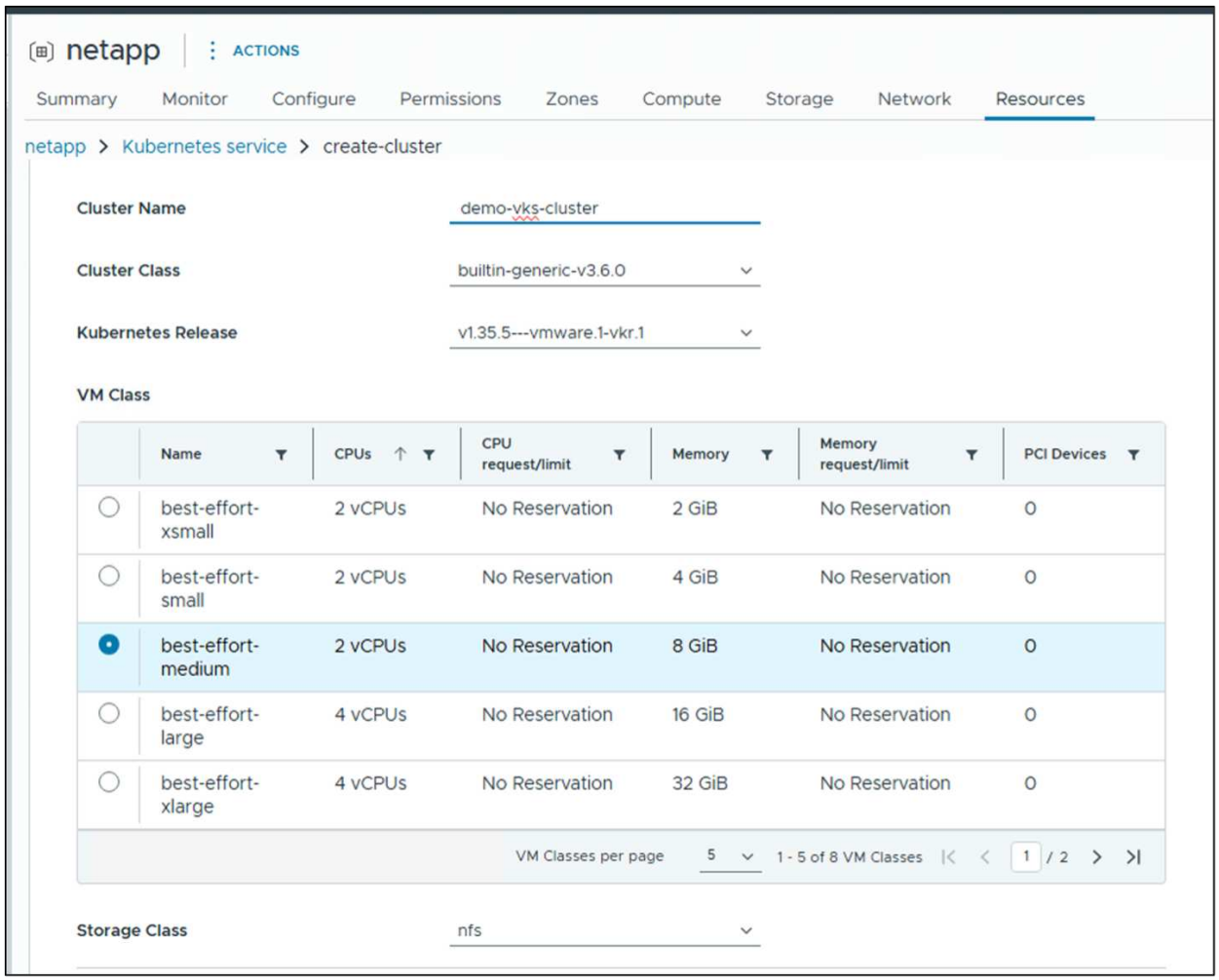
     - Im Abschnitt 1, **Konfigurationstyp**, **Benutzerdefiniert** auswählen.

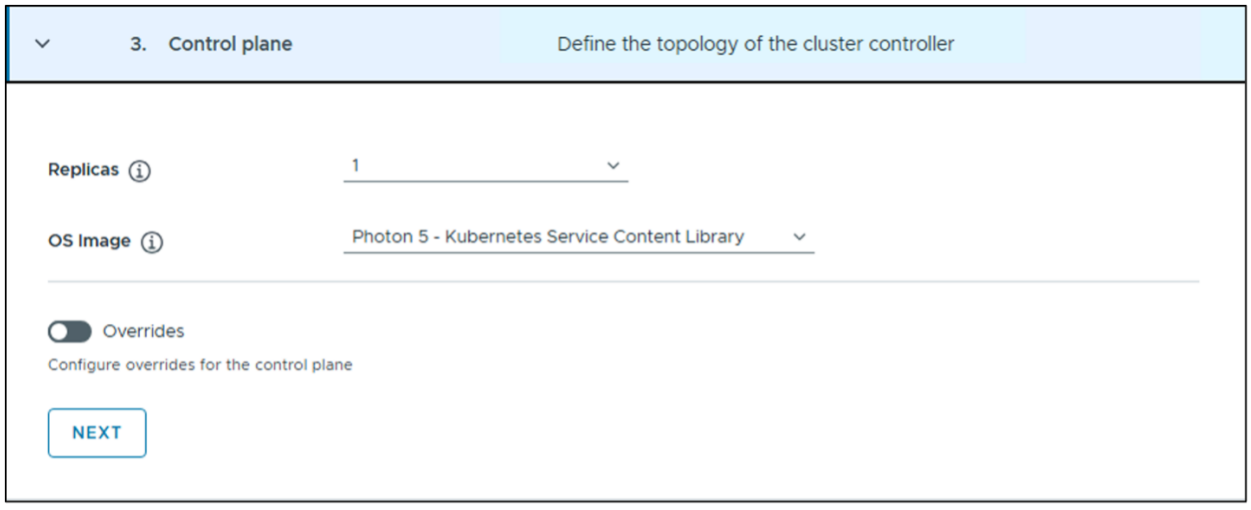
- Im Abschnitt 2, **Allgemeine Einstellungen**, einen Namen für den Cluster angeben und alle anderen Einstellungen auf ihren Standardwerten belassen.
- Alle Standardeinstellungen für Abschnitt 3 übernehmen.
- In Abschnitt 4, **Node Pools**, auf die Auslassungspunkte (…) neben dem Node Pool klicken und **Bearbeiten** auswählen. Die Anzahl der Replikate auf 3 erhöhen und die neueste Ubuntu-Version für das OS-Image auswählen. **Weiter** und anschließend **Überprüfen und bestätigen** auswählen.

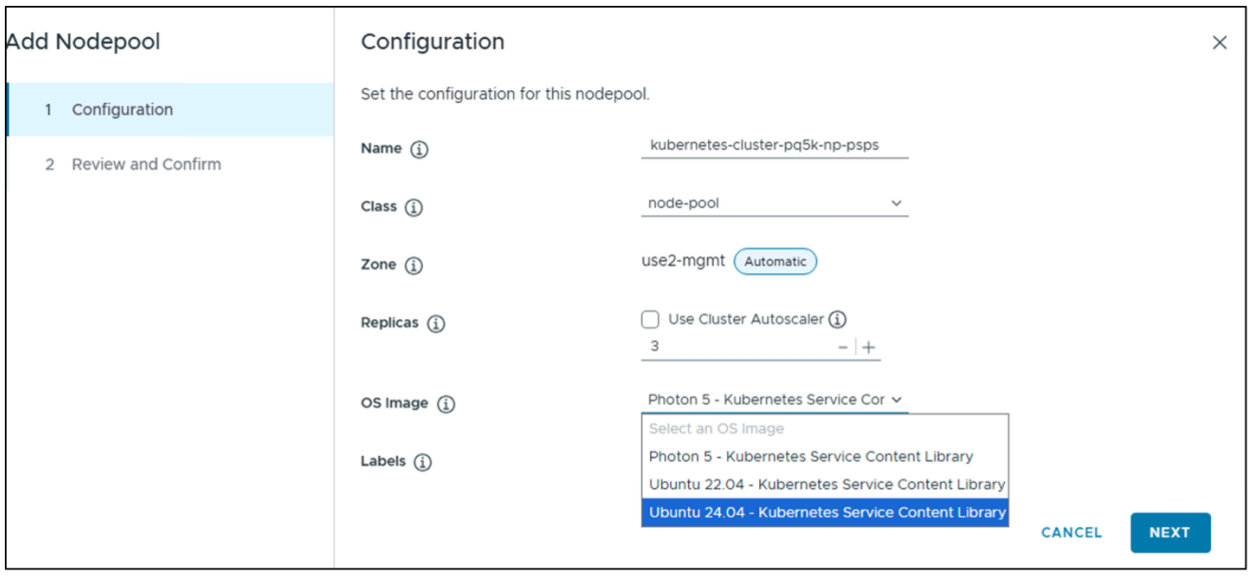
Nachdem die Clusterdetails überprüft wurden, auf **Fertigstellen** klicken. Der vSphere Kubernetes-Cluster wird in wenigen Minuten erstellt.

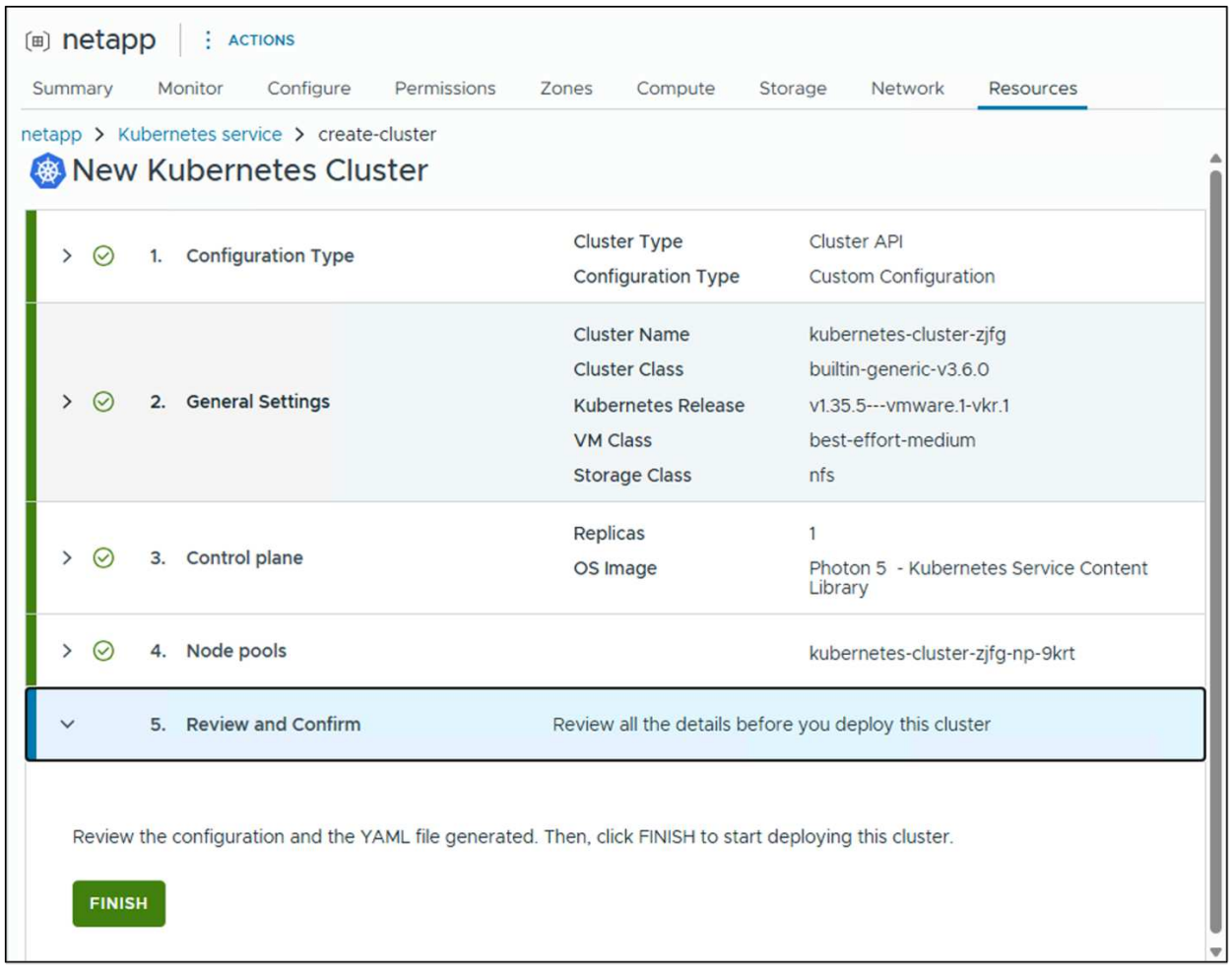


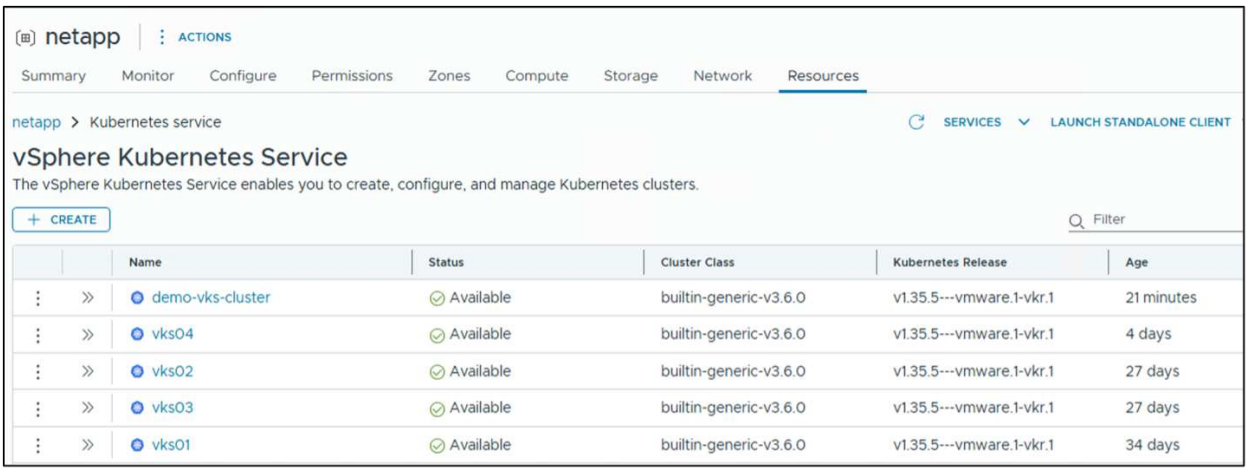



netapp
ACTIONS
Summary
Monitor
Configure
Permissions
Zones
Compute
Storage
Network
Resources
netapp > Kubernetes service > create-cluster
Cluster Name
demo-vks-cluster
Cluster Class
builtin-generic-v3.6.0
Kubernetes Release
v1.35.5---vmware.1-vkr.1
VM Class
Name
CPUs
CPU request/limit
Memory
Memory request/limit
PCI Devices
best-effort-xsmall
2 vCPUs
No Reservation
2 GiB
No Reservation
0
best-effort-small
2 vCPUs
No Reservation
4 GiB
No Reservation
0
best-effort-medium
2 vCPUs
No Reservation
8 GiB
No Reservation
0
best-effort-large
4 vCPUs
No Reservation
16 GiB
No Reservation
0
best-effort-xlarge
4 vCPUs
No Reservation
32 GiB
No Reservation
0
VM Classes per page
5
1 - 5 of 8 VM Classes
1 / 2
Storage Class
nfs


3. Control plane
Define the topology of the cluster controller
Replicas
1
OS Image
Photon 5 - Kubernetes Service Content Library
Overrides
Configure overrides for the control plane
NEXT


Add Nodepool
1 Configuration
2 Review and Confirm
Configuration
Set the configuration for this nodepool.
Name
kubernetes-cluster-pq5k-np-psps
Class
node-pool
Zone
use2-mgmt
Automatic
Replicas
Use Cluster Autoscaler
3
OS Image
Photon 5 - Kubernetes Service Cor
Select an OS Image
Photon 5 - Kubernetes Service Content Library
Ubuntu 22.04 - Kubernetes Service Content Library
Ubuntu 24.04 - Kubernetes Service Content Library
Labels
CANCEL
NEXT


netapp
ACTIONS
Summary
Monitor
Configure
Permissions
Zones
Compute
Storage
Network
Resources
netapp > Kubernetes service > create-cluster
New Kubernetes Cluster
1. Configuration Type
Cluster Type
Cluster API
Configuration Type
Custom Configuration
2. General Settings
Cluster Name
kubernetes-cluster-zjfg
Cluster Class
builtin-generic-v3.6.0
Kubernetes Release
v1.35.5---vmware.1-vkr.1
VM Class
best-effort-medium
Storage Class
nfs
3. Control plane
Replicas
1
OS Image
Photon 5 - Kubernetes Service Content Library
4. Node pools
kubernetes-cluster-zjfg-np-9krt
5. Review and Confirm
Review all the details before you deploy this cluster
Review the configuration and the YAML file generated. Then, click FINISH to start deploying this cluster.
FINISH



▼ Anleitung für die Erstellung eines benutzerdefinierten OVA-Images für SAN-fähige Worker-Knoten anzeigen

Ein Docker-Image mit den erforderlichen Hilfsprogrammen erstellen. Im folgenden Beispiel wird die Erstellung eines Docker-Images auf Basis von Ubuntu 24.04 mit installierten iSCSI- und Multipathing-Hilfsprogrammen gezeigt.

```
#san-dockerfile.txt
FROM ubuntu:24.04
# Prevent interactive prompts during package installation
ENV DEBIAN_FRONTEND=noninteractive
# Install required system packages for iSCSI and Multipathing
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
    nvme-cli \
    kmod \
    linux-tools-common \
    open-iscsi \
    lsscsi \
    sg3-utils \
    multipath-tools \
    scsitools  \
    chrony \
    networkd-dispatcher &&\
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
# Set iSCSI session scanning to manual as required by Trident
RUN sed -i 's/^\(node.session.scan\).*/\1 = manual/'
/etc/iscsi/iscsid.conf
# Configure device-mapper-multipath with Trident recommendations
RUN printf "defaults {\n    user_friendly_names yes\n
find_multipaths no\n}\n" > /etc/multipath.conf
RUN mkdir -p /etc/networkd-dispatcher/off.d /etc/networkd-
dispatcher/routable.d/
```

```
CMD ["bash"]
```

Das Docker-Image wird mit folgendem Befehl erstellt:

```
docker build -f san-dockerfile.txt -t ubuntu-2404:san .
```

Falls keine Docker registry verfügbar ist, kann folgender Befehl verwendet werden, um eine lokale Registry zu starten und das Image dorthin zu übertragen:

```
docker run -d -p 5000:5000 --restart=always --name registry
registry:2
```

Das Bild wird getaggt und gepusht.

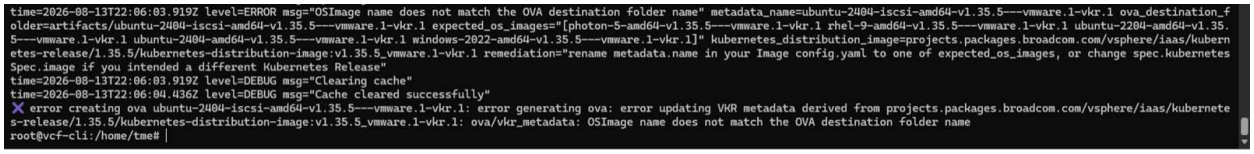
```
docker tag ubuntu-2404:san localhost:5000/ubuntu-2404:san
docker push localhost:5000/ubuntu-2404:san
```

Die Image Konfigurationsdatei erstellen (als `ubuntu2404vkr135-san.yml` speichern) und auf das Image verweisen:

```
#ubuntu2404vkr135-san.yml
apiVersion: imageconfiguration.vmware.com/v1alpha1
kind: Image
metadata:
    name: ubuntu-2404-amd64-v1.35.5---vmware.1-vkr.1
spec:
    osSpec:
        name: ubuntu
        version: "24.04"
        image: "localhost:5000/ubuntu-2404:san"
    kubernetesSpec:
        image:
projects.packages.broadcom.com/vsphere/iaas/kubernetes-
release/1.35.5/kubernetes-distribution-image:v1.35.5_vmware.1-vkr.1
    diskSize: 20Gi
    repositorySpec:
        debs:
            - name: noble
              url: http://archive.ubuntu.com/ubuntu
              suites:
                  - noble
              components:
                  - main
```

```
                - restricted
                - universe
                - multiverse
            - name: noble-security
              url: http://security.ubuntu.com/ubuntu
              suites:
                - noble-security
              components:
                - main
                - restricted
                - universe
                - multiverse
    packages:
      present:                  # Packages to install
        - name: open-iscsi
        - name: lsscsi
        - name: sg3-utils
        - name: multipath-tools
        - name: scsitools
        - name: nvme-cli
    services:
      - name: multipath-tools.service
        status: enabled
      - name: open-iscsi.service
        status: enabled
```

Der Metadatenname muss aus der Liste der vorab genehmigten Namen stammen. Ist dies nicht der Fall, tritt beim Versuch, die OVA-Datei zu erstellen, der unten gezeigte Fehler auf. Die Liste der vorab genehmigten Namen befindet sich in der VCF CLI-Hilfe für den `kr bake` Befehl.



Die OVA-Datei wird mithilfe der VCF-Befehlszeilenschnittstelle erstellt:

```
vcf kr bake -f <path/to/ubuntu2404vkr135-san.yml> --log-level DEBUG
--log-format plain
```

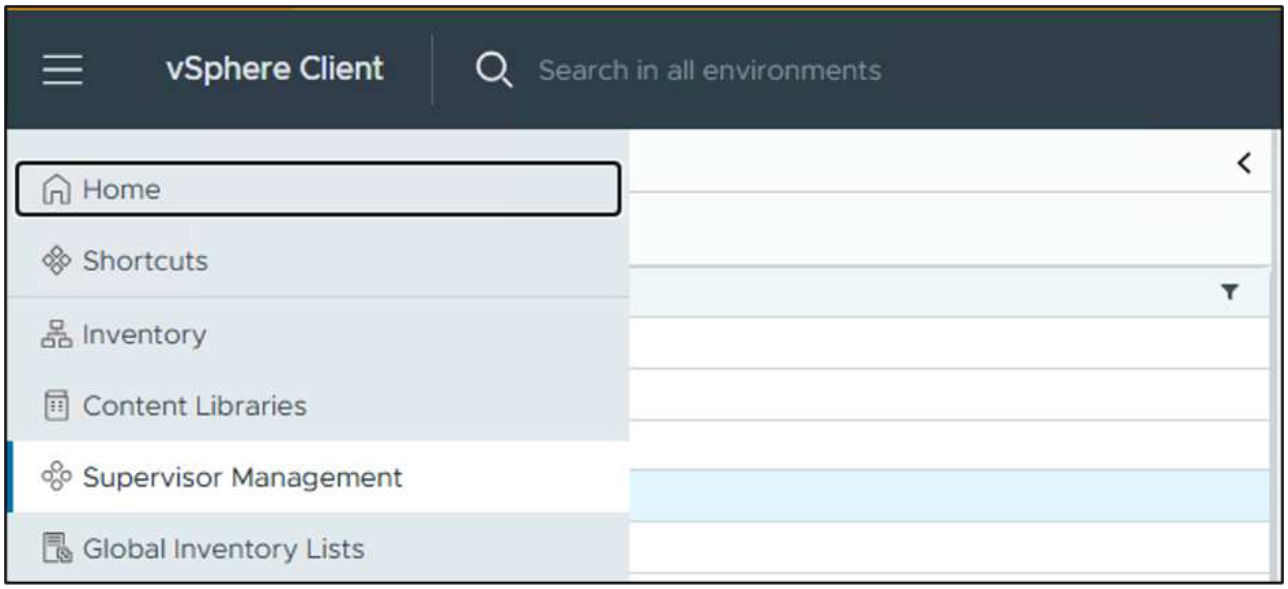
Übertragen Sie die OVA-Datei per SCP auf einen Rechner, von dem aus Sie sie in die vSphere Content Library hochladen können.

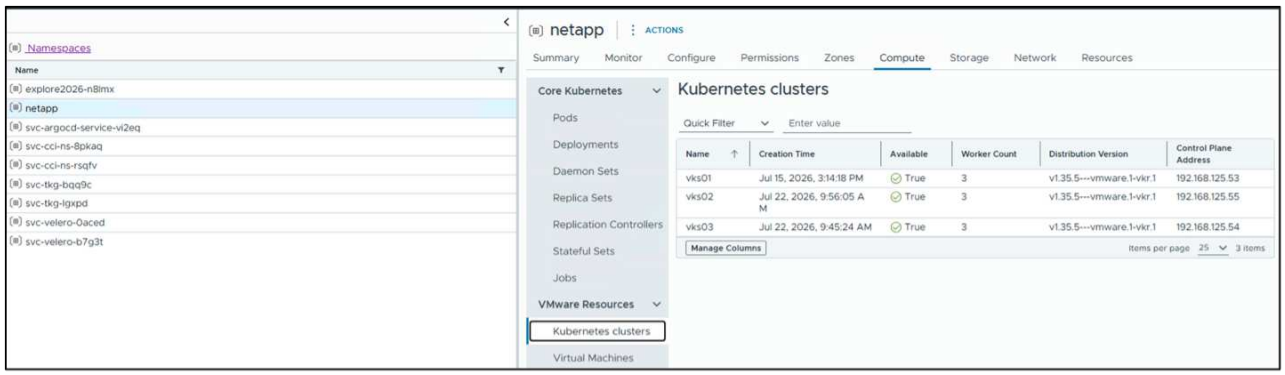
2. Nachdem der Cluster installiert ist, befindet er sich in vCenter, und die kubeconfig-Datei kann heruntergeladen werden.

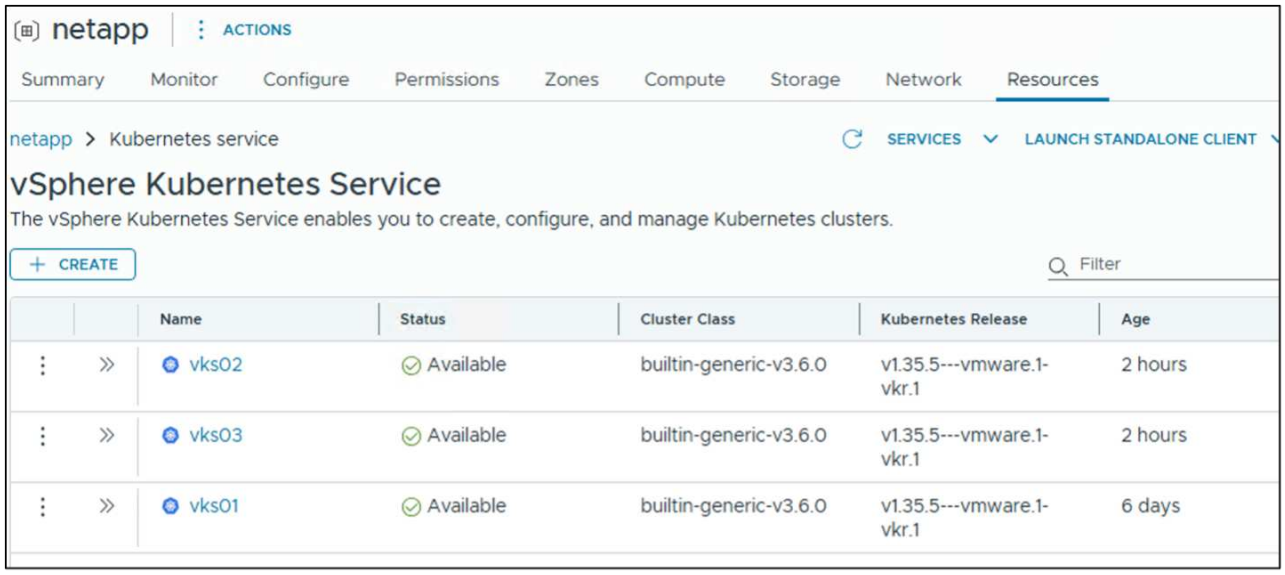
a. Im Hauptmenü **Supervisor Management** auswählen und den Namespace wählen, in dem das Cluster erstellt wurde.

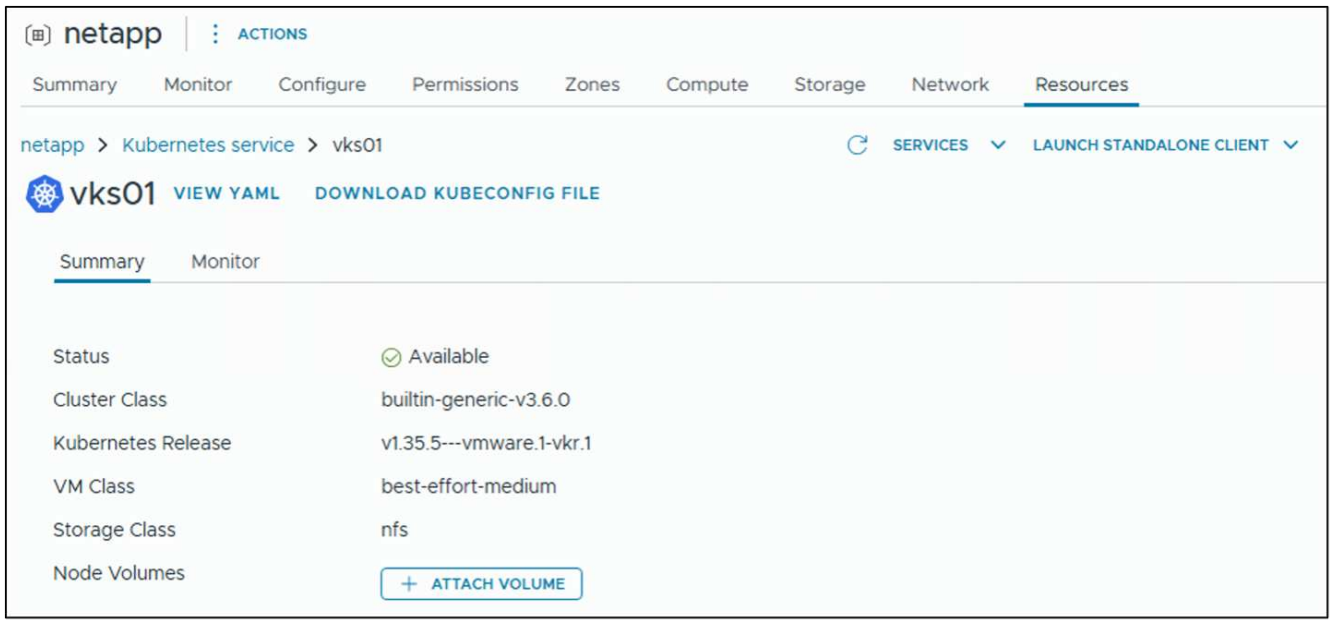
b. Den Tab **Compute** und anschließend **Kubernetes Clusters** auswählen. Alle Cluster im Namespace werden angezeigt.

c. Zum Tab **Ressourcen** wechseln, den benötigten Kubernetes-Cluster auswählen und dessen kubeconfig-Datei herunterladen.









3. Von einem Bastion-Host aus kann über die kubeconfig-Datei eine Verbindung zum Cluster hergestellt werden, um ihn über die CLI zu verwalten.

```
vksbastion@vks:~$ kubectl get nodes
NAME                                      STATUS   ROLES           AGE     VERSION
vks01-dhwbg-rpxst                         Ready    control-plane   6d20h   v1.35.5+vmware.1
vks01-vks01-np-pt27-764sx-cdd5q-g5qnw     Ready    <none>          3h45m   v1.35.5+vmware.1
vks01-vks01-np-pt27-764sx-cdd5q-ltj5s     Ready    <none>          6d20h   v1.35.5+vmware.1
vks01-vks01-np-pt27-764sx-cdd5q-npxmj     Ready    <none>          6d20h   v1.35.5+vmware.1
```

Videodemonstration

Die folgenden Videos demonstrieren zwei Möglichkeiten zur Erstellung eines vSphere Kubernetes-Clusters.

Erstellen eines vSphere Kubernetes-Clusters auf einem Supervisor Cluster

Erstellung eines vSphere Kubernetes Clusters mit VCF Automation

Informationen zu NetApp Storage für vSphere Kubernetes Service

Die Verwendung von NetApp als Storage für den vSphere Kubernetes Service (VKS) ist eine leistungsstarke Kombination, die die robusten Storage-Lösungen von NetApp nutzt, um die Performance, Skalierbarkeit und Zuverlässigkeit von Kubernetes-Workloads zu verbessern. NetApp Trident, ein Open-Source-Storage-Bereitsteller für dynamische Storage-Bereitstellung in Kubernetes, wird für die Integration von Storage mit Workloads auf dem vSphere Kubernetes Service verwendet.

Wichtigste Vorteile der Nutzung von NetApp Storage mit VKS

1. **Dynamische Speicherbereitstellung:** NetApp Trident ermöglicht die dynamische Bereitstellung von Speichervolumes, wodurch Kubernetes-Pods sofortige Storage-Bereitstellung entsprechend ihren Anforderungen anfordern können.
2. **Persistenter Speicher:** NetApp Lösungen bieten persistente Speicherfunktionen und gewährleisten so die Datenbeständigkeit und -verfügbarkeit auch nach Neustarts und Ausfällen von Pods.

3. **Hohe Leistungsfähigkeit:** NetApps Speicherlösungen, wie ONTAP, bieten hochleistungsfähigen Speicher mit geringer Latenz, was ideal für E/A-intensive Anwendungen ist, die auf Kubernetes laufen.
4. **Skalierbarkeit:** NetApp Storage-Systeme können skaliert werden, um den Anforderungen wachsender Kubernetes-Workloads gerecht zu werden und unterstützen groß angelegte Bereitstellungen.
5. **Datenmanagement:** NetApp bietet erweiterte Datenmanagementfunktionen, darunter Snapshots, Klonen und Datenreplizierung, die für Backup, Notfallwiederherstellung und Entwicklungs-Workflows von Vorteil sein können.
6. **Multi-Cloud und Hybrid Cloud Support:** NetApps Storage-Lösungen können in On-Premises-, Public-Cloud- und Hybrid-Cloud-Umgebungen bereitgestellt werden und bieten Flexibilität und Konsistenz im Storage-Management.

NetApp ONTAP Integrationsübersicht

NetApp ONTAP bietet Speicher der Enterprise-Klasse mit Funktionen wie Datendeduplizierung, Komprimierung und Verschlüsselung. Mit NetApp Trident lässt sich NetApp Storage in Kubernetes integrieren. NetApp Trident unterstützt verschiedene NetApp Storage-Plattformen, darunter ONTAP On-Premises, FSx für NetApp ONTAP in AWS, Azure NetApp Files in Azure und Google Cloud NetApp Volumes in Google Cloud.

Trident Protect wird verwendet, um den Datenschutz und die Notfallwiederherstellung für Ihre Kubernetes-Workloads zu gewährleisten. Mit Trident Protect lassen sich Snapshots und Klone erstellen sowie Daten über verschiedene Storage-Backends hinweg replizieren, sodass Ihre Daten im Falle eines Ausfalls sicher und wiederherstellbar sind.

Hauptmerkmale von Trident

CSI-Konformität Gewährleistung der Kompatibilität mit den neuesten Kubernetes-Speicherfunktionen und -Standards. **Deklarative Konfiguration** Ermöglicht es Benutzern, Speicheranforderungen in YAML-Dateien zu definieren, die dann automatisch von Trident verwaltet werden. **Treiber** Trident unterstützt Treiber für NFS-, CIFS/SMB-, iSCSI-, FC- und NVMe/TCP-Protokolle, die von ONTAP unterstützt werden. **Unterstützung für Hybrid- und Multi-Cloud-Umgebungen** Trident unterstützt Treiber für ONTAP Storage on-premises und verwaltete Speicherdienste bei den Hyperscalern, namentlich FSx für NetApp ONTAP in AWS, Azure NetApp Files in Azure und Google Cloud NetApp Volumes in Google Cloud. **Erweitertes Datenmanagement** Nutzung der Snapshot- und Klonfunktionen von ONTAP für effizienten Datenschutz und die schnelle Bereitstellung von Test- und Entwicklungsumgebungen.

Vollständige Details zu Trident finden sich in der "Trident Dokumentation".

Trident Protect

Trident Protect wird separat von Trident installiert. Vor der Bereitstellung sollte überprüft werden, ob Ihre Kubernetes-Plattform, Ihr Storage-Backend, Ihre Snapshot-Komponenten und Ihr Objektspeicher die "Trident Protect Anforderungen" erfüllen.

Hauptmerkmale von Trident Protect

Automatisierte Backups Trident Protect ermöglicht automatisierte, richtlinienbasierte Backups von Kubernetes-Anwendungen, sodass diese regelmäßig ohne manuelle Eingriffe gesichert werden.

Snapshot-Verwaltung Es nutzt die Snapshot-Funktionen von ONTAP, um häufige, zeitpunktgenaue Kopien von Containeranwendungen und VMs zu erstellen und bietet so einen zuverlässigen Mechanismus für die Wiederherstellung.

Verschlüsselung von Backup-Repositories Trident Protect unterstützt die passwortbasierte Verschlüsselung

für Restic- und Kopia-Backup-Repositories. Die Verschlüsselung von Persistent Volumes und während der Übertragung wird separat in den Storage- und Netzwerkebenen konfiguriert.

Compliance und Auditing Bietet Tools und Funktionen, die Organisationen dabei unterstützen, Compliance-Anforderungen zu erfüllen und regelmäßige Audits ihrer Datenschutzpraktiken durchzuführen.

Disaster Recovery Integriert sich in die Replikationsfunktionen von ONTAP, um robuste Disaster-Recovery-Lösungen bereitzustellen und die Geschäftskontinuität auch im Falle katastrophaler Ereignisse zu gewährleisten.

Vollständige Details zu Trident Protect finden sich im "Trident Protect Dokumentation".

NetApp ONTAP unterstützte Hardwaremodelle und Protokolle

NetApp ONTAP Software läuft auf einer Vielzahl von Hardwaremodellen, die jeweils auf unterschiedliche Leistungs- und Kapazitätsanforderungen ausgelegt sind. Trident erweitert seine umfassenden Funktionen, um verschiedene NetApp ONTAP Systeme zu unterstützen, darunter All Flash FAS (AFF), All SAN Array (ASA) und ASA R2 Systeme. Diese Unterstützung stellt sicher, dass Unternehmen das volle Potenzial ihrer Hochleistungs-Storage-Lösungen in containerisierten Umgebungen ausschöpfen können.

All Flash FAS (AFF) Systeme AFF Systeme bieten außergewöhnliche Leistung und geringe Latenz, was sie ideal für anspruchsvolle Anwendungen macht.

All SAN Array (ASA) Systeme ASA Systeme sind für dedizierte SAN-Umgebungen konzipiert und bieten robuste Leistung und Zuverlässigkeit.

ASA R2 Systems ASA R2-Systeme bieten erweiterte Funktionen und Möglichkeiten für SAN-Umgebungen.

AFX NetApp AFX ist eine speziell entwickelte, disaggregierte All-Flash-Storage-Architektur, die für KI-Workloads der Enterprise-Klasse konzipiert ist. Sie bietet leistungsstarkes NAS und ermöglicht die unabhängige Skalierung von Rechenleistung und Kapazität. NetApp AFX-Speichersysteme unterscheiden sich von anderen ONTAP-basierten Systemen (ASA, AFF und FAS) in der Implementierung ihrer Speicherschicht. AFX verwendet ein verteiltes Dateisystem, das hohe Leistung und Skalierbarkeit ermöglicht, während andere ONTAP-basierte Systeme eine traditionelle Speicherarchitektur nutzen.

1. Unterstützte Protokolle für AFF: NFS, CIFS/SMB, iSCSI, FC, NVMe/TCP
2. Unterstützte Protokolle für ASA: iSCSI, FC, NVMe/TCP
3. Für ASA R2 unterstützte Protokolle: iSCSI, FC, NVMe/TCP
4. Unterstützte Protokolle für AFX: Ab Trident 25.10 wird nur das NFS-Protokoll unterstützt; das SMB-Protokoll wird nicht unterstützt.

Trident Treiber für ONTAP Storage

Trident beinhaltet die folgenden CSI-Treiber, die jeweils in der Lage sind, ein Volume im Backend-Speicher bereitzustellen.

Für NAS-Protokolle auf unterstützten Hardwaremodellen

1. ontap-nas: Ein PVC wird auf ein PV abgebildet, das ein dediziertes FlexVol volume ist.
2. ontap-nas-economy: Jedes bereitgestellte PV ist ein Qtree, wobei die Anzahl der Qtrees pro FlexVol volume konfigurierbar ist (Standardwert ist 200, konfigurierbar zwischen 50 und 300).

   Eine PVC ist auf eine PV abgebildet, die ein Qtree auf einem gemeinsam genutzten FlexVol volume ist.

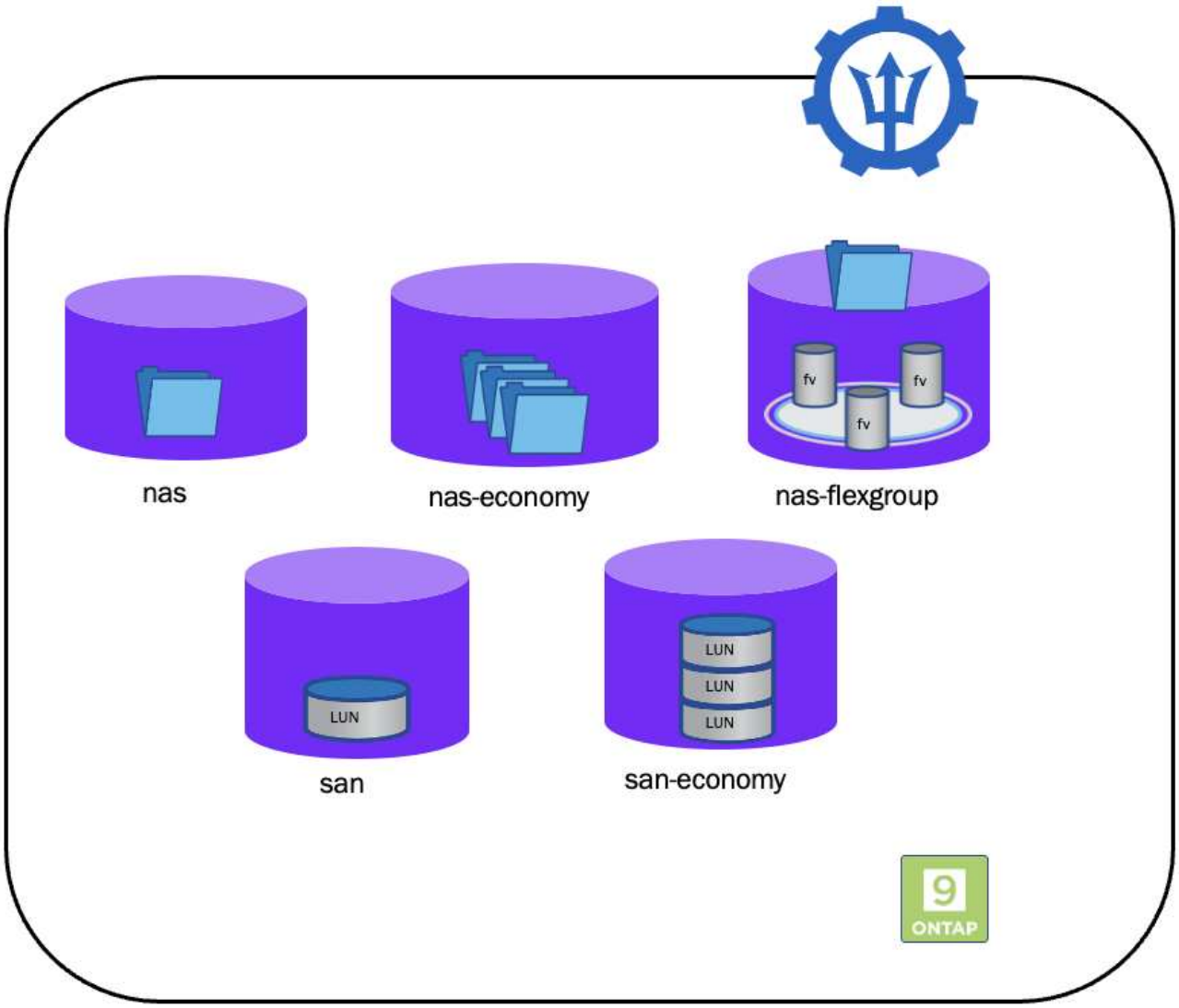
Sie können alle Festplatten der VMs als qtrees in einem einzigen Volume platzieren. Es ist jedoch zu beachten, dass Snapshots, Klone und Replikationsfunktionen von Trident nicht unterstützt werden. Wenn diese Funktionen vom Storage-Administrator verwaltet werden, erfolgt die Verwaltung nicht auf PV-Ebene.

1. ontap-nas-flexgroup: Jeder bereitgestellte PV ist ein vollständiger ONTAP FlexGroup, und alle einem SVM zugewiesenen Aggregate werden verwendet.

Für SAN-Protokolle auf unterstützten Hardwaremodellen

1. ontap-san: Eine PVC ist einer PV zugeordnet, die eine LUN auf einem dedizierten FlexVol volume ist.
2. ontap-san-economy: Ein PVC ist einem PV zugeordnet, das eine LUN auf einem gemeinsam genutzten FlexVol volume darstellt. Es können mehrere LUNs in dem gemeinsam genutzten FlexVol volume vorhanden sein (Standardwert ist 100, konfigurierbar zwischen 50 und 200 LUNs/PVs pro FlexVol volume).

Sie können alle Festplatten der VMs als LUNs in einem einzigen Volume ablegen. Dieser Treiber unterstützt jedoch Snapshots und Klone, aber keine Replikation. Wenn die Replikation vom Storage-Administrator verwaltet wird, ist sie nicht PV-granular.



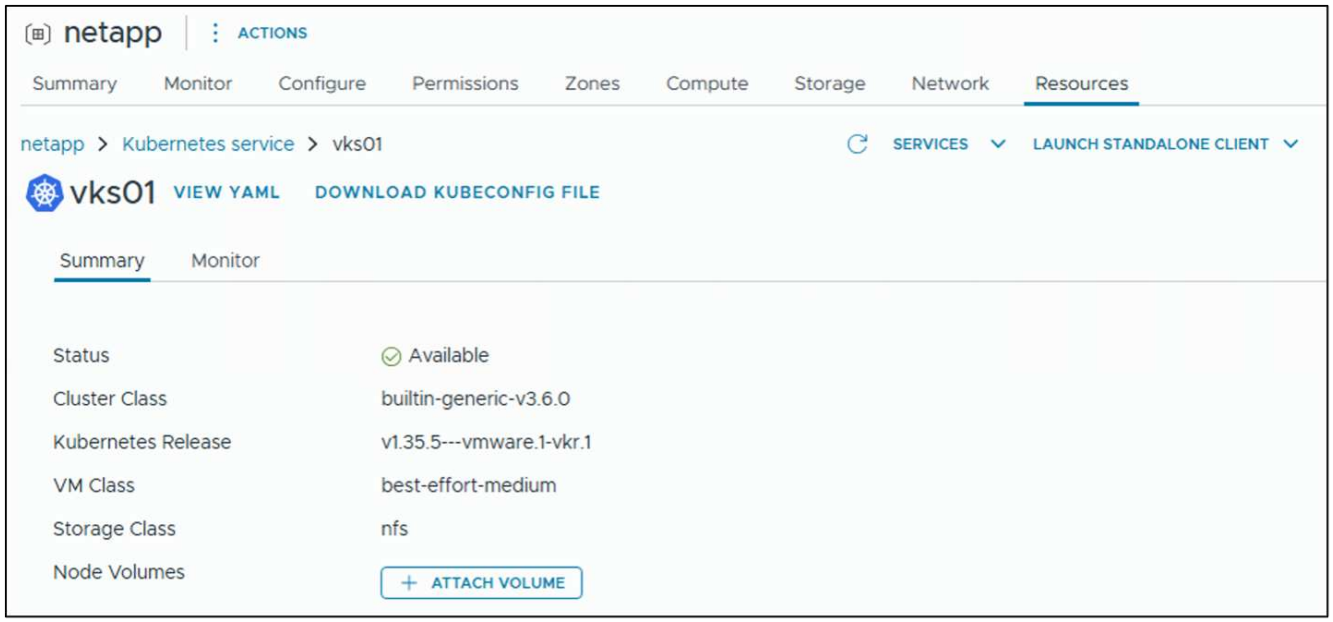
Eine vollständige Liste der über Trident Treiber bereitgestellten Funktionen sowie derjenigen, die vom Speicheradministrator angewendet werden müssen, ist im "ONTAP-Backend-Treiber" Abschnitt der Trident-Dokumentation zu finden.

Installation von NetApp Trident in einem VKS Cluster

NetApp Trident als dynamischer Storage Provisioner im vSphere Kubernetes Service-Cluster installieren, um NetApp ONTAP Storage mit Kubernetes-Workloads zu integrieren.

Bevor Sie beginnen

- Stellen Sie sicher, dass Ihr ONTAP Cluster konfiguriert und mit Ihrer VMware Kubernetes Umgebung verbunden ist.
- Stellen Sie sicher, dass auf Ihrem VKS-Cluster die iSCSI- oder NVMe-Tools installiert sind, wenn Ihre Workloads diese Protokolle für die Speicherung verwenden.
- Stellen Sie sicher, dass `kubectl` auf einem Rechner installiert ist, von dem aus eine Verbindung zum VKS-Cluster mithilfe der kubeconfig-Datei hergestellt werden kann.



Schritte

1. Der Trident Operator kann mithilfe eines Helm-Charts entsprechend den Anweisungen in der Trident-Dokumentation installiert werden:

   "https://docs.netapp.com/us-en/trident/trident-install/kubernetes-deploy-helm.html"

   Vor der Ausführung von Helm install muss der `trident`Namespace erstellt und gelabelt werden. Das `enforce=privileged`Label ist erforderlich, da Trident privilegierte Workloads ausführt. Ohne dieses Label verhindert der Kubernetes Pod Security Admission Controller das Starten der Trident-Pods.

   ```
   kubectl create namespace trident
   kubectl label namespace trident pod-
   security.kubernetes.io/enforce=privileged
   ```

▼ **Beispiel anzeigen**

```
helm install trident netapp-trident/trident-operator --version
100.2606.0 --namespace trident
```

Um die Debug-Protokollierung zu aktivieren, `--set tridentDebug=true` hinzufügen:

```
helm install trident netapp-trident/trident-operator --version
100.2606.0 --namespace trident --set tridentDebug=true
```

Trident kann auch manuell mithilfe des Trident Operator installiert werden. Die Vorgehensweisen in der Trident Dokumentation sind zu prüfen: "Den Trident Operator manuell bereitstellen"

Stellen Sie sicher, dass der `trident`Namespace mit den erforderlichen Pod-Sicherheitsbezeichnungen versehen ist, bevor Trident manuell installiert wird.

2. Vergewissern Sie sich, dass alle Trident Pods im Cluster ausgeführt werden.

▼ **Beispiel anzeigen**

```
kubectl get pods -n trident
```

```
vksbastion@vks:~/demo-vks$ kubectl get pods -n trident
NAME                                  READY   STATUS    RESTARTS        AGE
trident-controller-67f7cd9bf8-4b289   6/6     Running   0               3d21h
trident-node-linux-46c86              2/2     Running   0               3d21h
trident-node-linux-bh29q              2/2     Running   0               11d
trident-node-linux-fcvjw              2/2     Running   1 (3d20h ago)   3d20h
trident-node-linux-gkhls              2/2     Running   0               3d20h
trident-operator-5cbf7f857-h5hx8      1/1     Running   0               3d20h
vksbastion@vks:~/demo-vks$
```

Falls Trident Pods aufgrund eines `PodSecurity` Zulassungsfehlers nicht starten, wurden die Pod-Sicherheitslabels möglicherweise nicht auf den `trident` Namespace vor der Installation angewendet. Mit `--overwrite` können sie erneut angewendet und anschließend überprüft werden, ob die Pods starten.

```
kubectl label namespace trident pod-
security.kubernetes.io/enforce=privileged --overwrite
```

Vorbereitung der Speicher-Backend- und StorageClass-Konfigurationsdateien für Ihre Umgebung

1. Ein Trident Backend für ONTAP wird konfiguriert, indem die erforderlichen Verbindungsdetails und Speicherparameter definiert werden.
2. In Kubernetes werden Speicherklassen definiert, die NetApp Storage-Backends zugeordnet sind. Diese Klassen geben Parameter wie Leistungsmerkmale und Replikationsrichtlinien an.

Definieren Sie Trident Backends und Speicherklassen für die folgenden Protokolle gemäß den Anforderungen Ihrer Workloads:

- NAS (ontap-nas Treiber)
- NAS Economy (ontap-nas-economy-Treiber)
- SAN (ontap-san Treiber) für iSCSI, FC oder NVMe Protokolle
- SAN Economy (ontap-san-economy driver) für iSCSI

NAS

Erstellen Sie eine TridentBackendConfig und StorageClass für ONTAP NAS, um die Bereitstellung von persistentem Speicher auf NFS-Basis zu ermöglichen. Die Backend-Konfiguration enthält Anmeldeinformationen, die in einem Kubernetes Secret gespeichert sind, und verweist auf Ihre ONTAP SVM und Management-LIF.

Beispiel für ein Backend-Geheimnis und eine Backend Konfigurationsdatei mit Authentifizierung über Benutzername und Passwort (speichern als `tbc-nas.yaml`):

```
# tbc-nas.yaml
apiVersion: v1
kind: Secret
metadata:
  name: tbc-nas-secret
  namespace: trident
type: Opaque
data:
  username: "<base64-encoded cluster admin username>"
  password: "<base64-encoded cluster admin password>"
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: tbc-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: <ONTAP management LIF>
  backendName: tbc-nas
  svm: <your SVM name>
  storagePrefix: <your prefix for all volume names created using
this backend configuration>
  defaults:
    nameTemplate: "{{ .config.StoragePrefix }}_{{
.volume.Namespace }}_{{ .volume.RequestName }}"
  credentials:
```

```
    name: tbc-nas-secret
```

Beispiel für ein Backend-Geheimnis und eine Backend Konfigurationsdatei mit Clientzertifikat-Authentifizierung (speichern unter `tbc-nas.yaml`):

```
# tbc-nas.yaml
apiVersion: v1
kind: Secret
metadata:
  name: ontap-nas-secret
  namespace: trident
type: Opaque
stringData:
  clientPrivateKey: <client private key value>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: tbc-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: <ONTAP management LIF>
  backendName: tbc-nas
  svm: <your SVM name>
  storagePrefix: <your prefix for all volume names created using
this backend configuration>
  credentials:
    name: ontap-nas-secret
  clientCertificate: <client certificate>
```

StorageClass-Definition (speichern als `sc-nas.yaml`):

`mountOptions` ist ein optionaler Parameter, der zusätzliche Mount-Optionen für NFS-Volumes festlegt. Die `nconnect` Option ermöglicht mehrere parallele TCP-Verbindungen für einen einzelnen NFS-Mount, was die Leistung bei Workloads mit hohem Durchsatz verbessern kann. Der Standardwert ist 1, kann aber bis zum maximal zulässigen Wert des ONTAP Systems erhöht werden.

ONTAP unterstützt bis zu 16 Verbindungen für einen einzelnen NFS-Mount auf einem nconnect-fähigen Client. Trident übergibt die Mount-Optionen direkt an die Kubernetes-Worker-Knoten, daher sollte ein Wert gewählt werden, der sowohl vom Worker-Knoten-NFS-Client als auch von ONTAP unterstützt wird; im folgenden Beispiel werden vier Verbindungen verwendet.

```
# sc-nas.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: sc-nas
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
allowVolumeExpansion: true
mountOptions:
- nconnect=4
```

Erstellen Sie eine TridentBackendConfig und StorageClass für den ONTAP NAS Economy Treiber, um die NFS-basierte Bereitstellung von persistentem Speicher mithilfe von Qtrees zu ermöglichen. Die Backend-Konfiguration enthält Anmeldeinformationen, die in einem Kubernetes Secret gespeichert sind, und verweist auf Ihre ONTAP SVM und Management-LIF.

Beispiel für ein Backend-Geheimnis und eine Backend Konfigurationsdatei mit Authentifizierung über Benutzername und Passwort (speichern als `tbc-nas-economy.yaml`):

```
# tbc-nas-economy.yaml
apiVersion: v1
kind: Secret
metadata:
  name: tbc-nas-economy-secret
  namespace: trident
type: Opaque
data:
  username: "<base64-encoded cluster admin username>"
  password: "<base64-encoded cluster admin password>"
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: tbc-nas-economy
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas-economy
  managementLIF: <ONTAP management LIF>
  backendName: tbc-nas-eco
  svm: <your SVM name>
```

```
  storagePrefix: <your prefix for all volume names created using this
backend configuration>
  defaults:
    nameTemplate: "{{ .config.StoragePrefix }}_{{ .volume.Namespace
}}_{{ .volume.RequestName }}"
  credentials:
    name: tbc-nas-economy-secret
```

StorageClass-Definition (speichern als `sc-nas-economy.yaml`):

```
# sc-nas-economy.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: sc-nas-economy
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas-economy"
  media: "ssd"
  provisioningType: "thin"
allowVolumeExpansion: true
mountOptions:
- nconnect=4
```

Eine TridentBackendConfig und StorageClass für ONTAP SAN mit iSCSI ermöglichen die Bereitstellung von iSCSI-basiertem Blockspeicher. Die Backend-Konfiguration verwendet den ontap-san Treiber und enthält Anmeldeinformationen, die in einem Kubernetes Secret gespeichert sind. Der `sanType` Parameter verwendet standardmäßig das iSCSI-Protokoll, wenn er weggelassen wird.

Backend-Geheimnis und Backend Konfigurationsdatei mit Authentifizierung durch Benutzername und Passwort (speichern als `tbc-iscsi.yaml`):

```
# tbc-iscsi.yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-iscsi-secret
  namespace: trident
type: Opaque
data:
  username: "<base64-encoded cluster admin username>"
  password: "<base64-encoded cluster admin password>"
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
```

```
metadata:
  name: ontap-iscsi
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-san
  managementLIF: <ONTAP management LIF>
  backendName: ontap-iscsi
  svm: <your SVM name>
  credentials:
    name: backend-tbc-ontap-iscsi-secret
```

Backend-Geheimnis und Backend Konfigurationsdatei mit Clientzertifikats-Authentifizierung (speichern als `tbc-iscsi.yaml`):

```
# tbc-iscsi.yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-iscsi-secret
  namespace: trident
type: Opaque
stringData:
  clientPrivateKey: <client private key>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: ontap-iscsi
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-san
  managementLIF: <ONTAP management LIF>
  backendName: ontap-iscsi
  svm: <your SVM name>
  credentials:
    name: backend-tbc-ontap-iscsi-secret
  clientCertificate: <client certificate>
```

StorageClass-Definition (speichern als `sc-iscsi.yaml`):

```
# sc-iscsi.yaml
apiVersion: storage.k8s.io/v1
```

```
kind: StorageClass
metadata:
  name: sc-iscsi
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  media: "ssd"
  provisioningType: "thin"
  fsType: ext4
  snapshots: "true"
allowVolumeExpansion: true
```

Eine TridentBackendConfig und StorageClass für ONTAP SAN mit NVMe ermöglichen die Bereitstellung von NVMe-basiertem Blockspeicher. Die Backend-Konfiguration verwendet den ontap-san Treiber und enthält Anmeldeinformationen, die in einem Kubernetes Secret gespeichert sind. Der `sanType` Parameter muss auf `nvme` gesetzt werden.

Backend-Geheimnis und Backend Konfigurationsdatei mit Authentifizierung durch Benutzername und Passwort (speichern als `tbc-nvme.yaml`):

```
# tbc-nvme.yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-nvme-secret
  namespace: trident
type: Opaque
data:
  username: "<base64-encoded cluster admin username>"
  password: "<base64-encoded cluster admin password>"
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: ontap-nvme
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-san
  sanType: nvme
  managementLIF: <ONTAP management LIF>
  backendName: ontap-nvme
  svm: <your SVM name>
  credentials:
    name: backend-tbc-ontap-nvme-secret
```

Backend-Geheimnis und Backend Konfigurationsdatei mit Clientzertifikats-Authentifizierung (speichern als `tbc-nvme.yaml`):

```
# tbc-nvme.yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-nvme-secret
  namespace: trident
type: Opaque
stringData:
  clientPrivateKey: <client private key>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: ontap-nvme
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-san
  sanType: nvme
  managementLIF: <ONTAP management LIF>
  backendName: ontap-nvme
  svm: <your SVM name>
  credentials:
    name: backend-tbc-ontap-nvme-secret
  clientCertificate: <client certificate>
```

StorageClass-Definition (speichern als `sc-nvme.yaml`):

```
# sc-nvme.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: sc-nvme
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  media: "ssd"
  provisioningType: "thin"
  fsType: ext4
  snapshots: "true"
allowVolumeExpansion: true
```

In der Trident-Dokumentation stehen weitere Beispiele für YAML-Dateien sowie Informationen zu den von SAN- und NAS-Treibern unterstützten Zugriffsmodi und Volumenmodi zur Verfügung.

- "Beispiele mit NAS Treibern"
- "Beispiele mit SAN Treibern"

Eine VolumeSnapshotClass Konfigurationsdatei erstellen

Eine VolumeSnapshotClass-Definition erstellen. Diese Konfiguration ermöglicht Snapshot-basierte Operationen für persistente Volumes.

VolumeSnapshotClass-Definition (speichern als `snapshot-class.yaml`):

```
# snapshot-class.yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: trident-snapshotclass
driver: csi.trident.netapp.io
deletionPolicy: Retain
```

Die Konfigurationsdateien auf Ihrem Cluster anwenden.

Wenden Sie die in den vorherigen Schritten erstellten Konfigurationsdateien auf den vSphere Kubernetes Service-Cluster mithilfe von `kubectl` an. Dadurch werden die erforderlichen Secrets, Backend-Konfigurationen, Speicherklassen und die Snapshot-Klasse in Ihrem Cluster erstellt.

Schritte

1. Wenden Sie die TridentBackendConfig und StorageClass Dateien für jedes von Ihnen konfigurierte Protokoll an.

```
kubectl apply -f tbc-nas.yaml -n trident
kubectl apply -f sc-nas.yaml

kubectl apply -f tbc-nas-economy.yaml -n trident
kubectl apply -f sc-nas-economy.yaml

kubectl apply -f tbc-iscsi.yaml -n trident
kubectl apply -f sc-iscsi.yaml

kubectl apply -f tbc-nvme.yaml -n trident
kubectl apply -f sc-nvme.yaml

kubectl apply -f snapshot-class.yaml
```

2. Überprüfen Sie, ob die Ressourcen erfolgreich erstellt wurden.

Prüfen Sie TridentBackendConfig-Objekte:

```
kubectl get tbc -n trident
```

Prüfen Sie StorageClass-Objekte:

```
kubectl get storageclass
```

Überprüfen Sie VolumeSnapshotClass:

```
kubectl get volumesnapshotclass
```

Die Standard Trident Storage- und Snapshot-Klassen festlegen

Trident StorageClass und VolumeSnapshotClass als Standardwerte im vSphere Kubernetes Service Cluster festlegen.

Schritte

1. Legen Sie die Standard-Trident-StorageClass fest.

   Legen Sie eine von Trident unterstützte StorageClass als Standard für den Cluster fest, damit PersistentVolumeClaims diese automatisch verwenden, wenn keine Speicherklasse angegeben ist.

   Stellen Sie sicher, dass nur eine StorageClass als Standard festgelegt ist. Wenn bereits eine andere StorageClass als Standard festgelegt ist, sollte deren Annotation auf `false` gesetzt werden.

   Über die CLI:

   ```
   kubectl patch storageclass <storage class name> -p '{"metadata":
   {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
   ```

2. Legen Sie die Standard-Trident-VolumeSnapshotClass fest.

   Eine von Trident unterstützte VolumeSnapshotClass als Standard für den Cluster festlegen, um Snapshot-basierte Operationen für persistente Volumes zu ermöglichen. Dadurch wird sichergestellt, dass VolumeSnapshots automatisch den Trident CSI-Treiber verwenden, wenn keine Snapshot-Klasse angegeben ist.

   Stellen Sie sicher, dass nur eine VolumeSnapshotClass als Standard festgelegt ist. Wenn bereits eine andere VolumeSnapshotClass als Standard festgelegt ist, sollte deren Annotation auf `false` gesetzt werden.

   Über die CLI:

   ```
   kubectl patch volumesnapshotclass <snapshot class name> --type=merge -p
   '{"metadata":{"annotations":{"snapshot.storage.kubernetes.io/is-default-
   ```

```
class":"true"}}}'
```

Kubernetes Persistent Volume Claims ermöglichen die Anforderung von Speicherplatz für Workloads.

Trident stellt die benötigten Volumes automatisch aus dem Backend NetApp Storage bereit. Siehe "Workloads mit persistentem Speicher bereitstellen" für Beispiele zur Bereitstellung einer Workload mit PVCs in einem VKS-Cluster.

Eine Containeranwendung mit NetApp persistentem Speicher bereitstellen

Eine containerisierte Anwendung kann im vSphere Kubernetes Service-Cluster mit NetApp ONTAP persistentem Speicher bereitgestellt werden. Die folgenden Beispiele zeigen die Bereitstellung einer PostgreSQL Datenbank mit entweder NAS (ontap-nas) oder SAN iSCSI (ontap-san) Speicher.

Die folgende YAML-Konfiguration setzt POSTGRES_USER / POSTGRES_PASSWORD direkt im Manifest. Im Produktivbetrieb wird empfohlen, Kubernetes Secrets zur Verwaltung sensibler Informationen wie Datenbank-Anmeldedaten zu verwenden.

PostgreSQL mit NAS-Speicher (ontap-nas Treiber) bereitstellen

Sie können eine PostgreSQL-Containeranwendung bereitstellen, die von einem durch den ontap-nas-Treiber bereitgestellten NFS-Persistent Volume unterstützt wird. Im folgenden Beispiel wird gezeigt, wie ein PersistentVolumeClaim (PVC) und eine Bereitstellung für PostgreSQL erstellt werden.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pvc
spec:
  storageClassName: sc-nas # Specify the StorageClass for dynamic
provisioning
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
```

```
  template:
    metadata:
      labels:
        app: postgres
    spec:
      securityContext:
        fsGroup: 999  # Ensures proper permissions for the mounted
volume (PostgreSQL default user is 999)
      containers:
        - name: postgres
          image: postgres:15
          securityContext:
            allowPrivilegeEscalation: false
            runAsNonRoot: true
            runAsUser: 999  # PostgreSQL default user ID
            capabilities:
              drop:
                - ALL
            seccompProfile:
              type: RuntimeDefault
          env:
            - name: POSTGRES_DB
              value: "postgres"
            - name: POSTGRES_USER
              value: "<username>"
            - name: POSTGRES_PASSWORD
              value: "<password>"
            - name: PGDATA
              value: "/var/lib/postgresql/data/pgdata"
          ports:
            - containerPort: 5432
          volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: postgres-storage
              subPath: postgres-data  # Ensures data is stored in a
subdirectory to avoid permission issues
          resources:
            requests:
              memory: "512Mi"
              cpu: "250m"
            limits:
              memory: "1Gi"
              cpu: "500m"
      volumes:
        - name: postgres-storage
          persistentVolumeClaim:
```

```
            claimName: postgres-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-service
spec:
  type: ClusterIP
  ports:
    - port: 5432
      targetPort: 5432
  selector:
    app: postgres
```

PostgreSQL mit SAN Speicher bereitstellen (ontap-san iSCSI Treiber)

Die folgende YAML kann verwendet werden, um eine PostgreSQL-Containeranwendung bereitzustellen, die von einem iSCSI-Persistent-Volume unterstützt wird, das durch den ontap-san Treiber bereitgestellt wird. Die `Recreate` Bereitstellungsstrategie stellt sicher, dass der Pod vollständig beendet ist, bevor ein neuer startet, was für ReadWriteOnce iSCSI-Volumes erforderlich ist und Datenbeschädigung bei Pod-Neustarts verhindert. Diese Konfiguration unterstützt die Datenpersistenz beim Löschen und Neuerstellen von Pods und ist mit Trident Volume Snapshots sowie Backup- und Restore-Operationen kompatibel.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-block-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
  storageClassName: sc-iscsi # Must map to an ontap-san driver
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-block-app
spec:
  replicas: 1
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: postgres
  template:
```

```
    metadata:
      labels:
        app: postgres
    spec:
      # 1. POD LEVEL SECURITY CONTEXT
      securityContext:
        runAsNonRoot: true
        runAsUser: 999        # Official Postgres image default user ID
        runAsGroup: 999       # Official Postgres image default group ID
        fsGroup: 999
        seccompProfile:
          type: RuntimeDefault
      containers:
        - name: postgres
          image: postgres:15
          # 2. CONTAINER LEVEL SECURITY CONTEXT
          securityContext:
            allowPrivilegeEscalation: false
            capabilities:
              drop:
                - ALL
          env:
            - name: POSTGRES_DB
              value: "postgres"
            - name: POSTGRES_USER
              value: "<username>"
            - name: POSTGRES_PASSWORD
              value: "<password>"
            - name: PGDATA
              value: /var/lib/postgresql/data/pgdata
          ports:
            - containerPort: 5432
              name: postgres
          volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: postgres-block-storage
              subPath: postgres-data
          resources:
            requests:
              memory: "512Mi"
              cpu: "250m"
            limits:
              memory: "1Gi"
              cpu: "500m"
      volumes:
        - name: postgres-block-storage
```

```
          persistentVolumeClaim:
            claimName: postgres-block-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-service
spec:
  type: ClusterIP
  ports:
    - port: 5432
      targetPort: 5432
  selector:
    app: postgres
```

Die Datenbankbereitstellung validieren

Nach der Bereitstellung der Anwendung sollte überprüft werden, ob die Pods im Status "Running" sind und die Datenbank betriebsbereit ist. Mit den folgenden Befehlen lässt sich der Status der Pods, PVCs und Services prüfen. Es sollte sichergestellt werden, dass die Pods laufen und die PVCs an die entsprechenden Volumes gebunden sind.

```
kubectl get all -n <namespace>
kubectl get pvc -n <namespace>
```

```
vksbastion@vks:~$ kubectl get all -n postgres
NAME                                    READY   STATUS    RESTARTS   AGE
pod/postgres-block-app-7c9859c558-bxhbg   1/1     Running   0          20h

NAME                       TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/postgres-service   ClusterIP   10.100.154.170   <none>        5432/TCP   25h

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/postgres-block-app   1/1     1            1           25h

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/postgres-block-app-7c9859c558   1         1         1       25h
vksbastion@vks:~$ kubectl get pvc -n postgres
NAME                STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
postgres-block-pvc   Bound    pvc-b992a5cf-1127-4b6c-a775-80ab64e4b536   20Gi       RWO            sc-iscsi       <unset>                 25h
vksbastion@vks:~$
```

Je nach verwendetem Protokoll handelt es sich beim Backend-Speicher um ein Volume, einen Qtree oder eine LUN. Der im ONTAP-System bereitgestellte Speicher kann überprüft werden, indem das PersistentVolume (PV) beschrieben wird, um den internen Volume-Namen zu ermitteln, der anschließend in ONTAP CLI-Befehlen verwendet werden kann, um die Speicherdetails abzurufen. Mit dem folgenden Befehl lässt sich das PV beschreiben und der interne Volume-Name abrufen:

```
kubectl describe pv/<pv-name>
```

```
vksbastion@vks:~/demo-vks$ kubectl describe pv/pvc-4782b054-5aff-49ae-bc40-08cdfd30db77
Name:            pvc-4782b054-5aff-49ae-bc40-08cdfd30db77
Labels:          <none>
Annotations:     pv.kubernetes.io/provisioned-by: csi.trident.netapp.io
                 volume.kubernetes.io/provisioner-deletion-secret-name:
                 volume.kubernetes.io/provisioner-deletion-secret-namespace:
Finalizers:      [external-provisioner.volume.kubernetes.io/finalizer kubernetes.io/pv-protection external-attacher/csi-trident-ne
tapp-io]
StorageClass:    sc-nas
Status:          Bound
Claim:           default/postgres-pvc
Reclaim Policy:  Delete
Access Modes:    RWO
VolumeMode:      Filesystem
Capacity:        10Gi
Node Affinity:   <none>
Message:
Source:
    Type:              CSI (a Container Storage Interface (CSI) volume source)
    Driver:            csi.trident.netapp.io
    FSType:
    VolumeHandle:      pvc-4782b054-5aff-49ae-bc40-08cdfd30db77
    ReadOnly:          false
    VolumeAttributes:      backendUUID=8d6b9527-91dc-4847-b5cc-6aa7307cf1ba
                           internalName=trident_pvc_4782b054_5aff_49ae_bc40_08cdfd30db77
                           name=pvc-4782b054-5aff-49ae-bc40-08cdfd30db77
                           protocol=file
                           storage.kubernetes.io/csiProvisionerIdentity=1787189141467-7183-csi.trident.netapp.io
Events:                <none>
```

Abbildung 1. Beispiel anzeigen

```
vksbastion@vks:~/vks02$ kubectl describe pv/pvc-b992a5cf-1127-4b6c-a775-80ab64e4b536
Name:            pvc-b992a5cf-1127-4b6c-a775-80ab64e4b536
Labels:          <none>
Annotations:     pv.kubernetes.io/provisioned-by: csi.trident.netapp.io
                 volume.kubernetes.io/provisioner-deletion-secret-name:
                 volume.kubernetes.io/provisioner-deletion-secret-namespace:
Finalizers:      [external-provisioner.volume.kubernetes.io/finalizer kubernetes.io/pv-protection external-attacher/csi-trident-ne
tapp-io]
StorageClass:    sc-iscsi
Status:          Bound
Claim:           postgres/postgres-block-pvc
Reclaim Policy:  Delete
Access Modes:    RWO
VolumeMode:      Filesystem
Capacity:        20Gi
Node Affinity:   <none>
Message:
Source:
    Type:              CSI (a Container Storage Interface (CSI) volume source)
    Driver:            csi.trident.netapp.io
    FSType:            ext4
    VolumeHandle:      pvc-b992a5cf-1127-4b6c-a775-80ab64e4b536
    ReadOnly:          false
    VolumeAttributes:      backendUUID=01dd718d-04a4-4c43-95fb-dc52f56e8624
                           internalName=trident_pvc_b992a5cf_1127_4b6c_a775_80ab64e4b536
                           name=pvc-b992a5cf-1127-4b6c-a775-80ab64e4b536
                           protocol=block
                           storage.kubernetes.io/csiProvisionerIdentity=1786638282639-3882-csi.trident.netapp.io
Events:                <none>
```

Den internen Volume-Namen aus der Ausgabe kopieren und den folgenden Befehl ausführen, um die Speicherdetails von der ONTAP CLI zu erhalten.

Für NAS Volumes:

```
volume show -vserver <vserver> -volume <internal-volume-name>
```

```
NSOL-NetApp-A70-T19U05::> volume show -vserver vks -volume trident_pvc_4782b054_5aff_49ae_bc40_08cdfd30db77

                                    Vserver Name: vks
                                     Volume Name: trident_pvc_4782b054_5aff_49ae_bc40_08cdfd30db77
                                  Aggregate Name: NSOL_NetApp_A70_T19U05a_NVME_SSD_1
    List of Aggregates for FlexGroup Constituents: NSOL_NetApp_A70_T19U05a_
                                                   NVME_SSD_1
                                Encryption Type: volume
               List of Nodes Hosting the Volume: NSOL-NetApp-A70-T19U05a
                                    Volume Size: 10GB
                             Volume Data Set ID: 2179
                      Volume Master Data Set ID: 2155359667
                                    Volume State: online
                                    Volume Style: flex
                           Extended Volume Style: flexvol
                         FlexCache Endpoint Type: none
                          Is Cluster-Mode Volume: true
                           Is Constituent Volume: false
                  Number of Constituent Volumes: -
                                 Export Policy: default
                                       User ID: 0
                                      Group ID: 0
                                Security Style: unix
                               UNIX Permissions: ---rwxrwxrwx
                                  Junction Path: /trident_pvc_4782b054_5aff_49ae_bc40_08cdfd30db77
                           Junction Path Source: RW_volume
```

Für SAN LUNs kann der folgende Befehl verwendet werden, um die LUN-Details aus der ONTAP CLI abzurufen:

```
lun show -vserver <vserver> -volume <internal-volume-name>
```

```
NSOL-NetApp-A70-T19U05::> lun show -vserver vks -volume trident_pvc_b992a5cf_1127_4b
6c_a775_80ab64e4b536
Vserver   Path                            State   Mapped   Type        Size
--------- ------------------------------ ------- -------- -------- --------
vks       /vol/trident_pvc_b992a5cf_1127_4b6c_a775_80ab64e4b536/lun0
                                          online  mapped   linux        20GB

NSOL-NetApp-A70-T19U05::>
```

Wenn Sie `nconnect`Mount-Optionen in der Speicherklasse NAS oder NAS Economy verwendet haben, lässt sich die Anzahl der aktiven TCP-Verbindungen vom Worker-Knoten zum Speicherserver überprüfen.

Zuerst die Trident Backend-Konfigurationen auflisten, um den Backend-Namen zu finden, dann diesen beschreiben, um den vserver-Namen und die Daten-LIF abzurufen:

```
kubectl get tbc -n trident
kubectl describe tbc <backend-name> -n trident
```

Melden Sie sich anschließend beim ONTAP Cluster an und führen Sie den folgenden Befehl aus, wobei die Daten-LIF aus der obigen Ausgabe ersetzt wird:

```
network connections active show -local-address <data-lif> -local-port 2049
```

```
NSOL-NetApp-A70-T19U05::> network connections active show -local-address 192.168.164.214 -local-port 2049
Vserver     Interface              Remote
Name        Name:Local Port        Host:Port                    Protocol/Service
----------- ---------------------- ---------------------------- ----------------
Node: NSOL-NetApp-A70-T19U05b
vks         lif_VKS_4608:2049      192.168.178.41:821           TCP/nfs
vks         lif_VKS_4608:2049      192.168.178.54:843           TCP/nfs
vks         lif_VKS_4608:2049      192.168.178.54:745           TCP/nfs
vks         lif_VKS_4608:2049      192.168.178.54:866           TCP/nfs
vks         lif_VKS_4608:2049      192.168.178.54:759           TCP/nfs
5 entries were displayed.
```

Abbildung 2. Beispiel anzeigen

Nachdem sich die Pods im Status "Running" befinden, eine Verbindung zur Datenbank herstellen und die Datenoperationen überprüfen.

Schritte

1. Den PostgreSQL-Dienstport an Ihren lokalen Rechner weiterleiten:

```
kubectl port-forward svc/postgres-service 5432:5432 -n <namespace>
```

2. Von einem anderen Terminal aus mit psql eine Verbindung zur Datenbank herstellen:

```
psql "postgresql://<username>:<password>@127.0.0.1:5432/postgres"
```

3. Eine Datenbank und eine Tabelle erstellen und die Tabelle anschließend mit Daten füllen:

```
CREATE DATABASE employee;
```

Zur neuen Datenbank wechseln (psql-Metabefehl):

```
\c employee
```

Erstellen Sie die Tabelle, fügen Sie eine Zeile ein und rufen Sie die Ergebnisse ab:

```
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    department VARCHAR(50),
    hire_date DATE DEFAULT CURRENT_DATE,
    salary NUMERIC(10, 2)
);

INSERT INTO employees (first_name, last_name, email, department, salary)
VALUES ('John', 'Doe', 'john.doe@example.com', 'Engineering', 85000.00);
```

```
SELECT * FROM employees;
```

Videodemonstration

Das folgende Video demonstriert die Installation von Trident auf einem vSphere Kubernetes-Cluster und die Bereitstellung einer PostgreSQL Datenbank mit persistentem Speicher mithilfe von Trident.

Bereitstellung von Workloads auf einem vSphere Kubernetes Service Cluster mit ONTAP persistentem Speicher unter Verwendung von Trident

Datensicherung

Containeranwendungen in einem vSphere Kubernetes Service-Cluster mit der NetApp Console sichern und wiederherstellen

Containeranwendungen in einem vSphere Kubernetes Service (VKS) Cluster mithilfe der NetApp Console sichern und wiederherstellen. Dieses Verfahren umfasst das Erstellen und Verwalten von Backup- und Restore-Vorgängen über den Backup and Recovery Service über die NetApp Console, wobei ONTAP S3 Objektspeicher als Backup-Ziel verwendet wird.

NetApp Console ermöglicht die zentrale Verwaltung von Speicher- und Datendiensten in lokalen und Cloud-Umgebungen. Es stehen einheitliche Kontrolle, einfache Bedienung und sicheres Management zur Verfügung. Mehrere Datendienste und Verwaltungsfunktionen sind über die Benutzeroberfläche unter "console.netapp.com" zugänglich. Vollständige Informationen zu NetApp Console finden sich unter "NetApp Console-Dokumentation".

Dieser Abschnitt konzentriert sich auf den NetApp Backup and Recovery Datendienst für Kubernetes-Workloads, insbesondere Containeranwendungen auf vSphere Kubernetes Service-Clustern. Der NetApp Backup and Recovery Service ermöglicht das Erkennen, Verwalten, Erstellen und Zuordnen von Datensicherungsstrategien für Ihre Kubernetes-Anwendungen über eine einzige Schnittstelle. Eine vollständige Anleitung befindet sich unter "NetApp Backup and Recovery Dokumentation".

Backup und Recovery Workflow

1. **Stellen Sie sicher, dass Sie einen Console agent haben.**

   Stellen Sie sicher, dass in der Umgebung, in der Ihr VKS-Cluster ausgeführt wird, ein Console Agent bereitgestellt ist. Einzelheiten zur Installation eines Console Agent finden sich in der "NetApp Console Setup-Dokumentation".

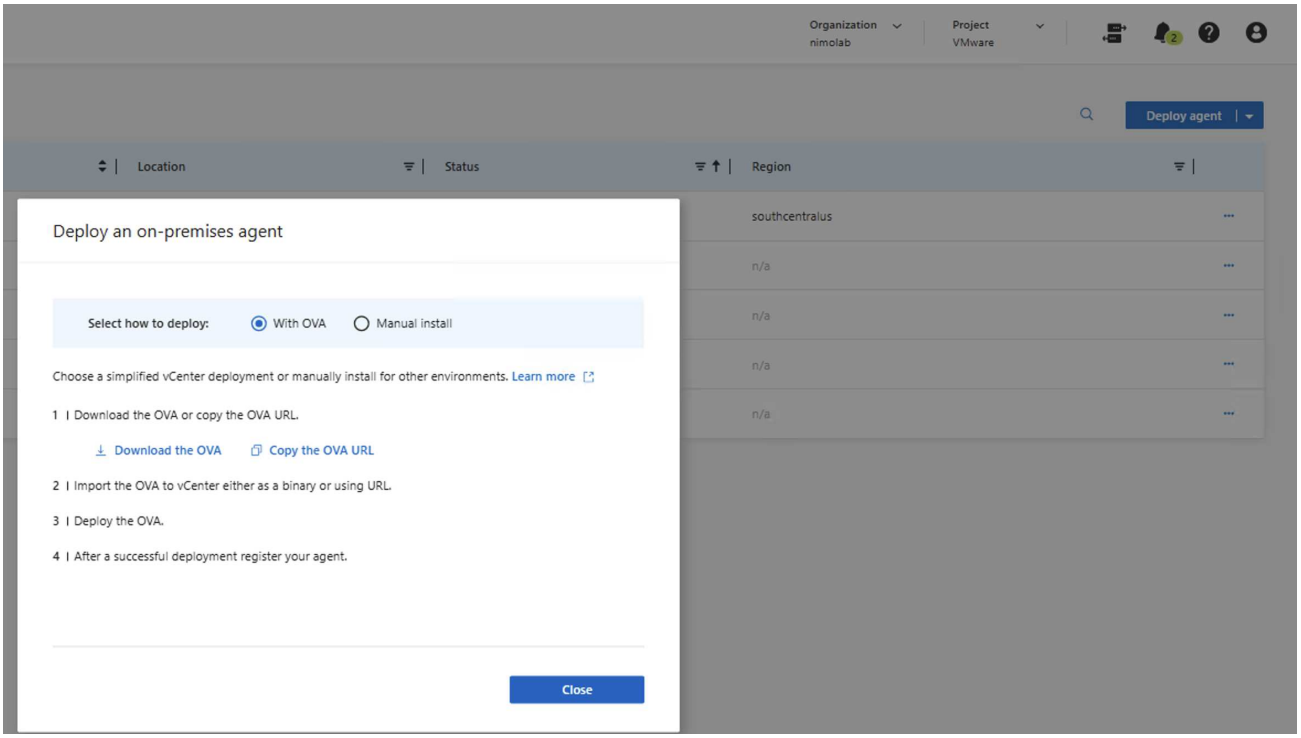
   ▼ **Beispiel anzeigen**

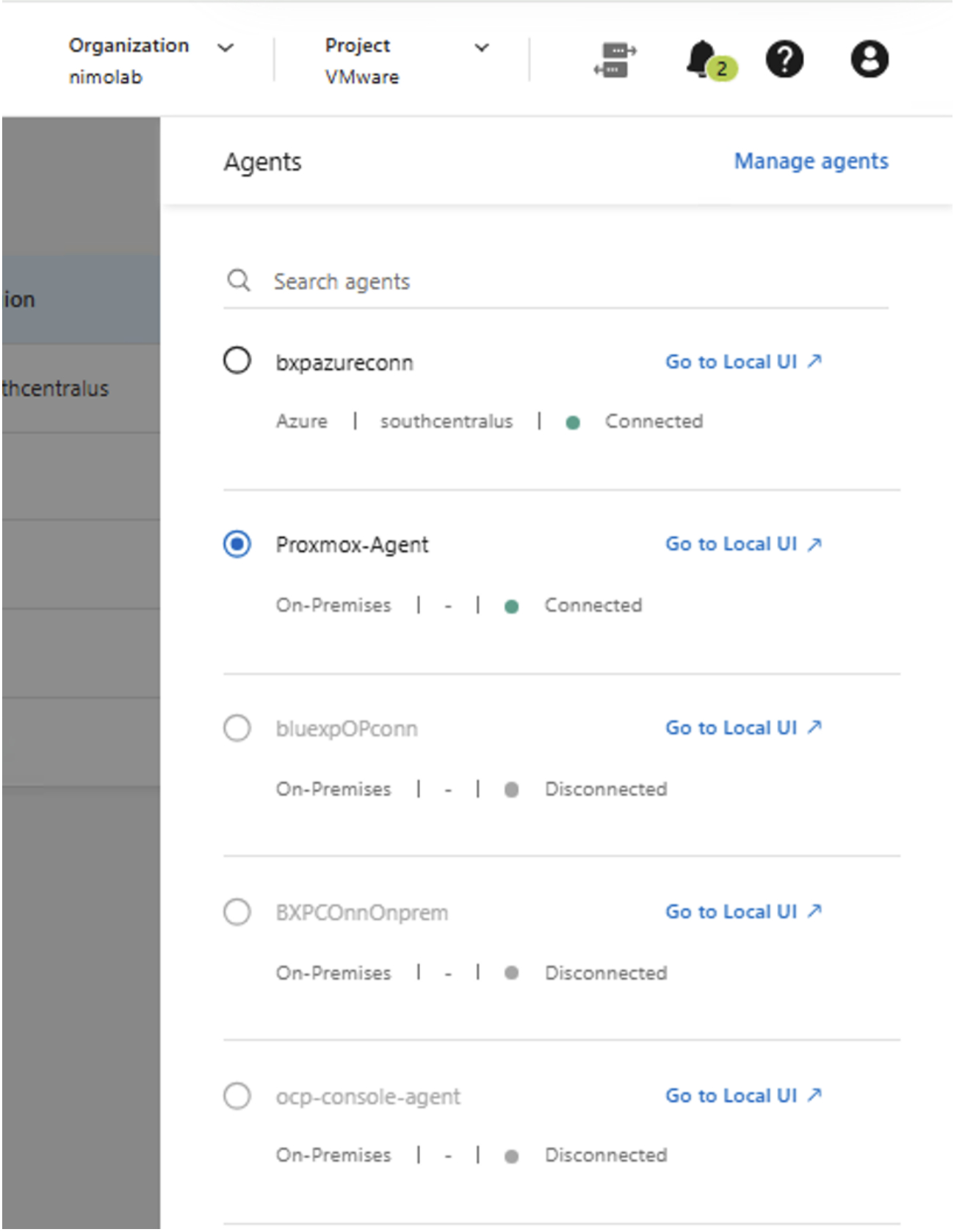
   In der NetApp Console auf „Agent bereitstellen“ klicken, wie im Screenshot gezeigt, und den Agenten in der Umgebung bereitstellen, in der der VKS-Cluster ausgeführt wird.

   In diesem Beispiel wird **Lokal > Mit OVA** ausgewählt, die OVA-URL kopiert und diese URL in vCenter verwendet, um den Console Agent bereitzustellen.

   Registrieren Sie ihn mithilfe der IP-Adresse des Agenten in der URL. Siehe "die Dokumentation" für detaillierte Anweisungen. Nach der Registrierung des Agenten ist dieser in der NetApp Console wie im

untenstehenden Screenshot sichtbar.





2. **Den ONTAP Cluster zur NetApp Console hinzufügen und NetApp Backup and Recovery aktivieren**

   Der primäre ONTAP-Cluster muss zur NetApp Console hinzugefügt und der Backup and Recovery Service darauf aktiviert werden, bevor Workloads geschützt werden können. Anweisungen finden sich unter "NetApp Backup and Recovery einrichten".

3. **In der NetApp Console den vSphere Kubernetes-Cluster entdecken**

▼ **Beispiel anzeigen**

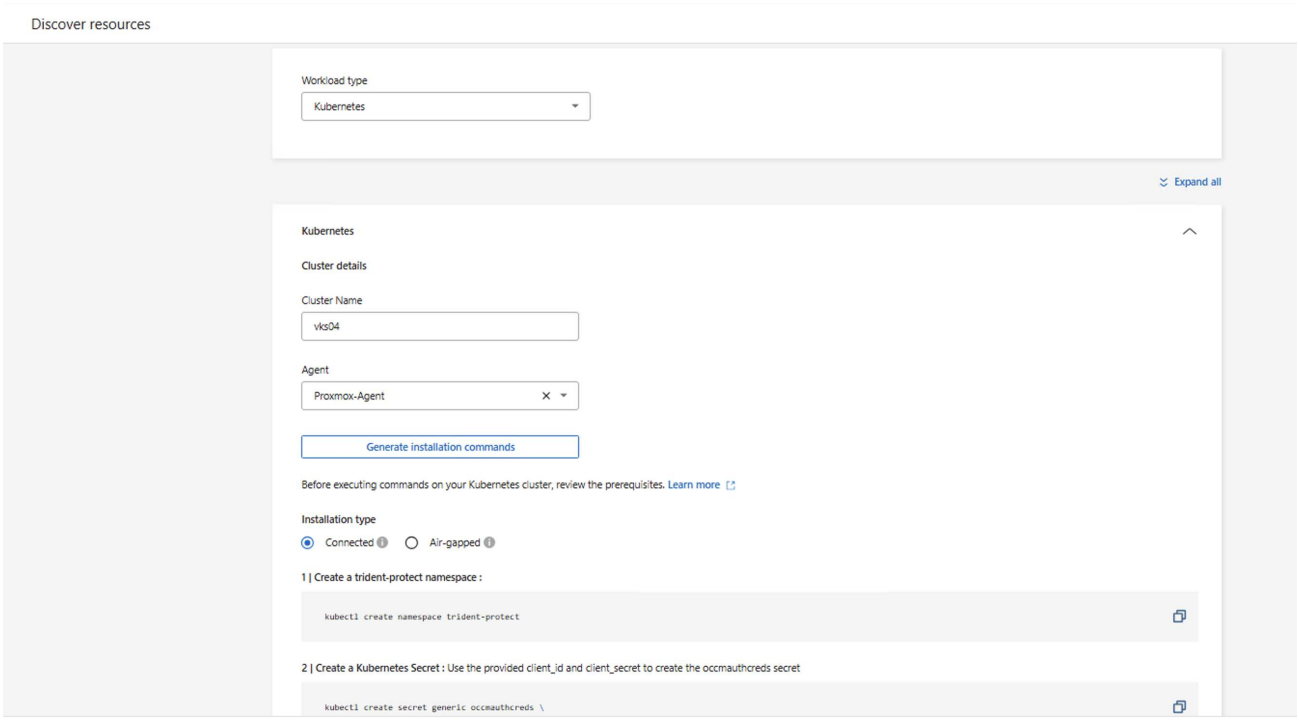
Für diesen Schritt muss der vSphere Kubernetes-Cluster ausgeführt werden und der Console-Agent in derselben Umgebung bereitgestellt sein. Die folgenden Schritte ermöglichen die Erkennung des vSphere Kubernetes-Clusters in der NetApp Console.

a. In der NetApp Console **Inventar > Kubernetes** auswählen und die Details des vSphere Kubernetes-Clusters angeben.

b. Wählen Sie den Agenten aus, der in derselben Umgebung bereitgestellt ist, in der sich der vSphere Kubernetes-Cluster befindet.

c. Die bereitgestellten Befehle können kopiert werden, um Trident Protect zu installieren, und von einem Rechner ausgeführt werden, der mit Ihrem vSphere Kubernetes-Cluster verbunden ist.

d. Nach erfolgreicher Installation von Trident Protect auf **Erkennen** klicken. NetApp Console erkennt den vSphere Kubernetes-Cluster und alle seine Ressourcen über den Agenten und stellt sie in der Benutzeroberfläche dar.



1 | Create a trident-protect namespace :

```
kubectl create namespace trident-protect
```

2 | Create a Kubernetes Secret : Use the provided client_id and client_secret to create the occmauthcreds secret

```
kubectl create secret generic occmauthcreds \
--namespace=trident-protect \
--from-literal=client_id=PZBjqkcEVZeeg-fJECfCRm42rO1EOKrq \
--from-literal=client_secret=5-XbW8MoCnQKb7ImJSIdTC0pef5W4klpmTqeLTeDU5s8ArZWYJuup7OuNdgyBJbz
```

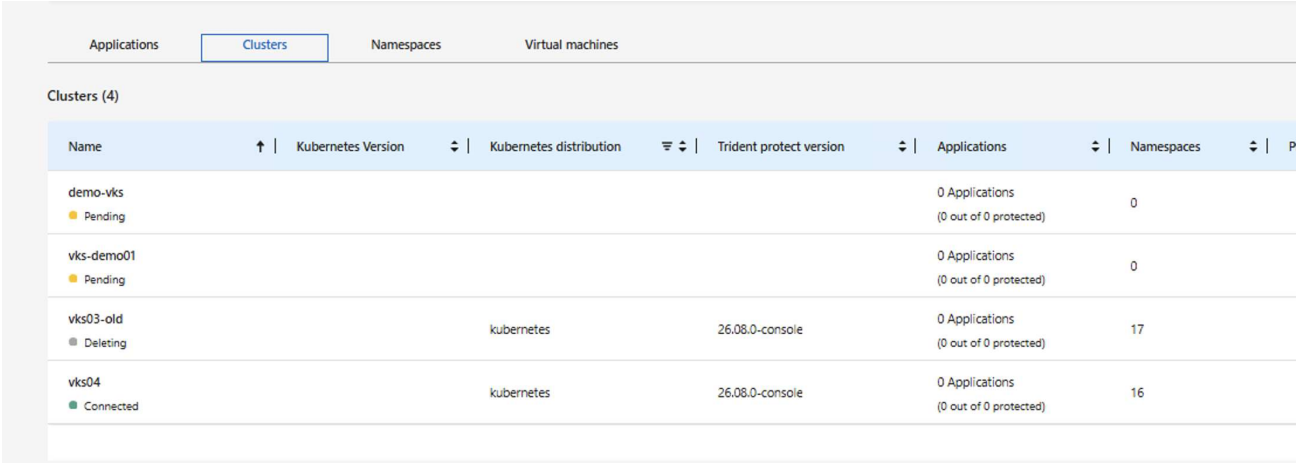
3 | Add/update helm repo :

```
helm repo add --force-update netapp-trident-protect https://netapp.github.io/trident-protect-helm-chart
```

4 | Install or Upgrade Trident Protect and Trident Protect Connector

```
helm upgrade --install trident-protect \
netapp-trident-protect/trident-protect-console \
--version 100.2608.0-console \
--namespace trident-protect \
--set clusterName=vks04 \
--set clusterID=966e65b1-c36a-441b-b23b-99db28c33151 \
--set trident-protect.cbs.accountID=account-B6oEAubL \
--set trident-protect.cbs.agentID=kgl3b3JQQR7e0NxidFg1wH3XockAb8p7clients \
--set trident-protect.cbs.proxySecretName=occmauthcreds \
--set trident-protect.cbs.proxyHostIP=https://10.192.113.198
```

```
vksbastion@vks:~/vks04$ kubectl get pods -n trident-protect
NAME                                                READY   STATUS    RESTARTS   AGE
trident-protect-connector-86bf87f7d7-76llf          1/1     Running   0          29s
trident-protect-controller-manager-8567cf76b9-lxxqz   1/1     Running   0          29s
vksbastion@vks:~/vks04$
```



4. **ONTAP S3 als Backup-Ziel einrichten**

   In diesem Beispiel wird ONTAP S3 Objektspeicher als Sicherungsziel verwendet. AWS S3, Azure Blob, Google Cloud Storage oder StorageGRID können ebenfalls als Sicherungsziele verwendet werden.

   Weitere Einzelheiten sind unter "NetApp Console-Dokumentation" zu finden.

   Um ONTAP S3 als Backup-Ziel in der NetApp Console hinzuzufügen, werden die folgenden Informationen von Ihrem ONTAP Cluster benötigt.

```
S3 Endpoint:
```

```
Access Key:
Secret Key:
Bucket Name:
```

Um diese Werte zu erhalten, sind im ONTAP Cluster mit dem System Manager die folgenden Schritte auszuführen.

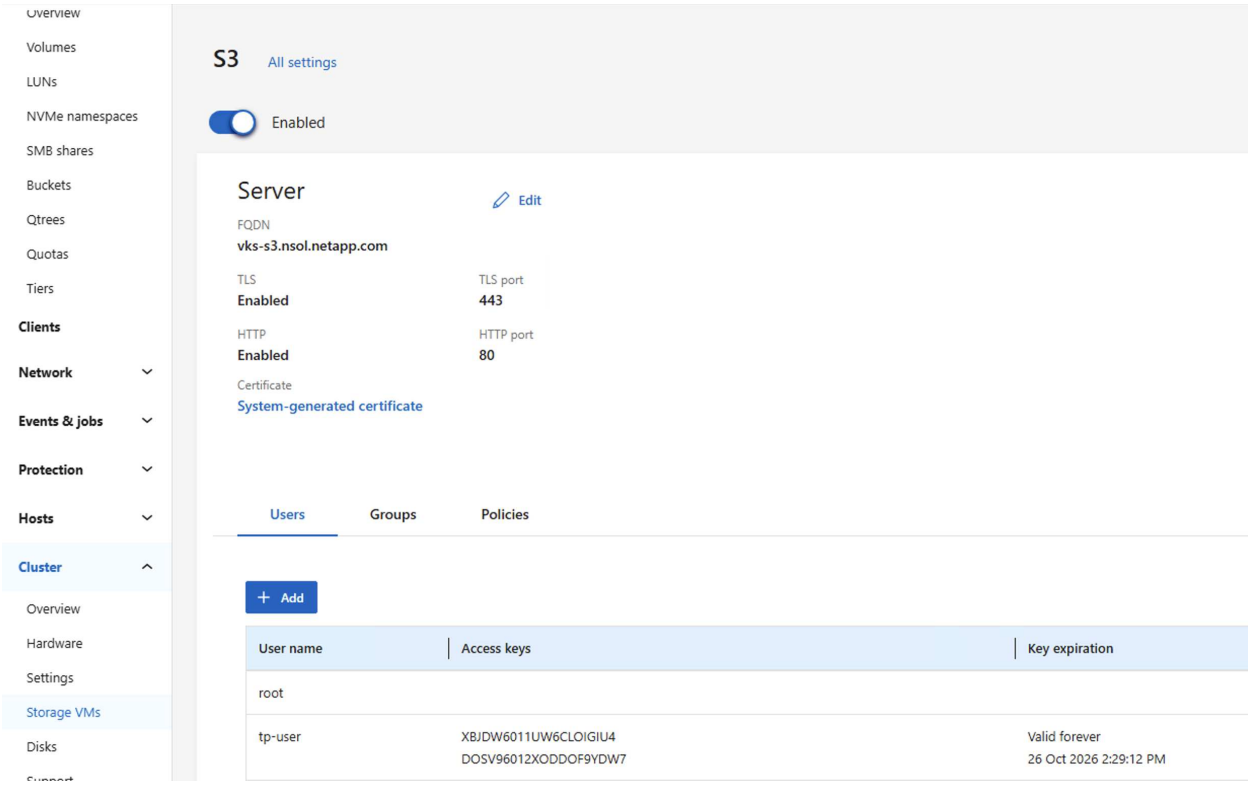
a. Im ONTAP Cluster eine S3-fähige SVM erstellen, um die Sicherungsdaten zu speichern. Die S3-Endpunkt-URL für diese SVM ist zu notieren.

b. In den S3-Einstellungen der SVM einen Benutzer erstellen und die erforderlichen Zugriffsrechte zuweisen. Der Zugriffsschlüssel und der geheime Schlüssel dieses Benutzers sollten notiert werden.

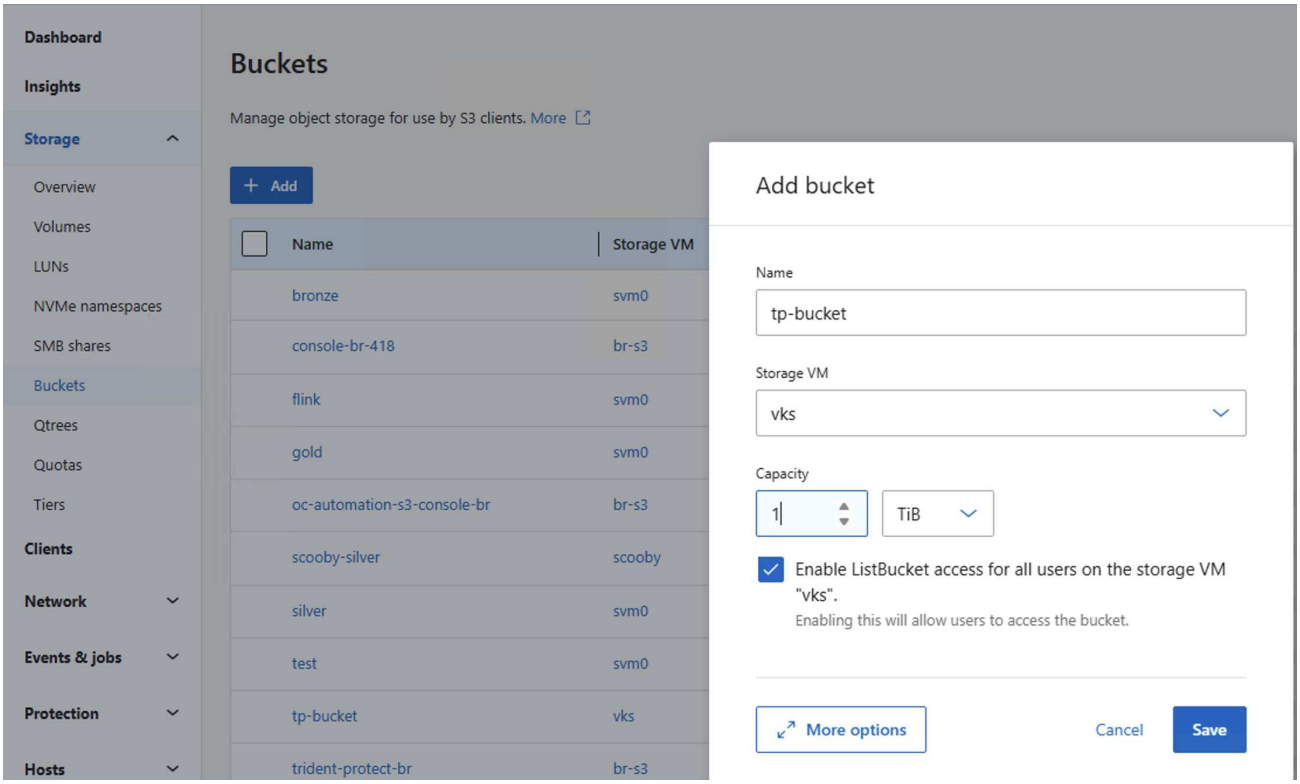
> Trident Protect erfordert, dass der S3-Benutzer mindestens `PutObject`, `GetObject`, `ListBucket` und `DeleteObject` Berechtigungen besitzt. Ohne diese Berechtigungen schlagen AppVault Backup- und Wiederherstellungsvorgänge mit Zugriffsverweigerungsfehlern fehl. Einzelheiten finden sich unter "Trident Protect AppVault Dokumentation".

▼ **Beispiel anzeigen**

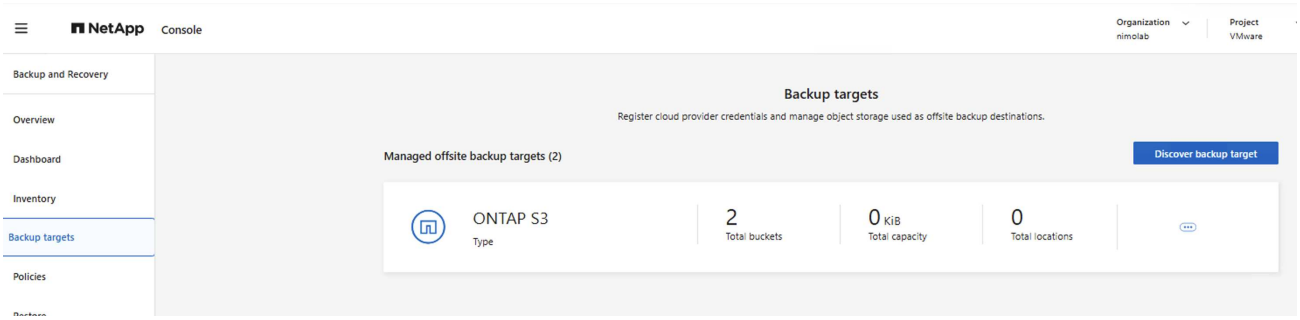
i. **S3-Benutzer erstellen und Zugriffsschlüssel und Secret Key herunterladen**

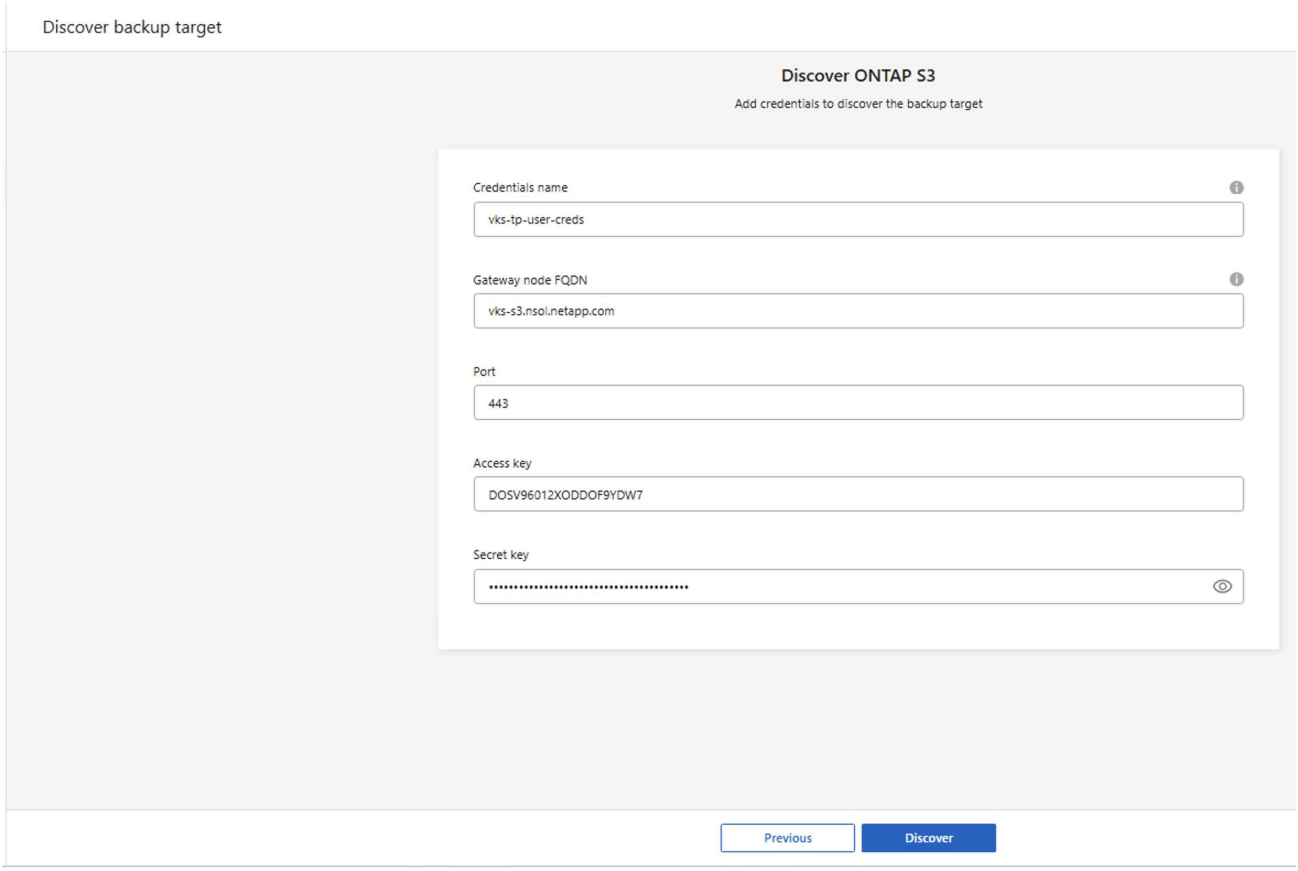


A. Erstellen Sie einen Bucket in der S3-fähigen SVM. Der Bucket-Name ist zu notieren, um ihn beim Hinzufügen des ONTAP S3 Objektspeichers als Backup-Ziel in der NetApp Console zu verwenden.



Den ONTAP S3 Objektspeicher in der NetApp Console als Sicherungsziel mit dem S3-Endpunkt, Zugriffsschlüssel, geheimen Schlüssel und Bucket-Namen hinzufügen.





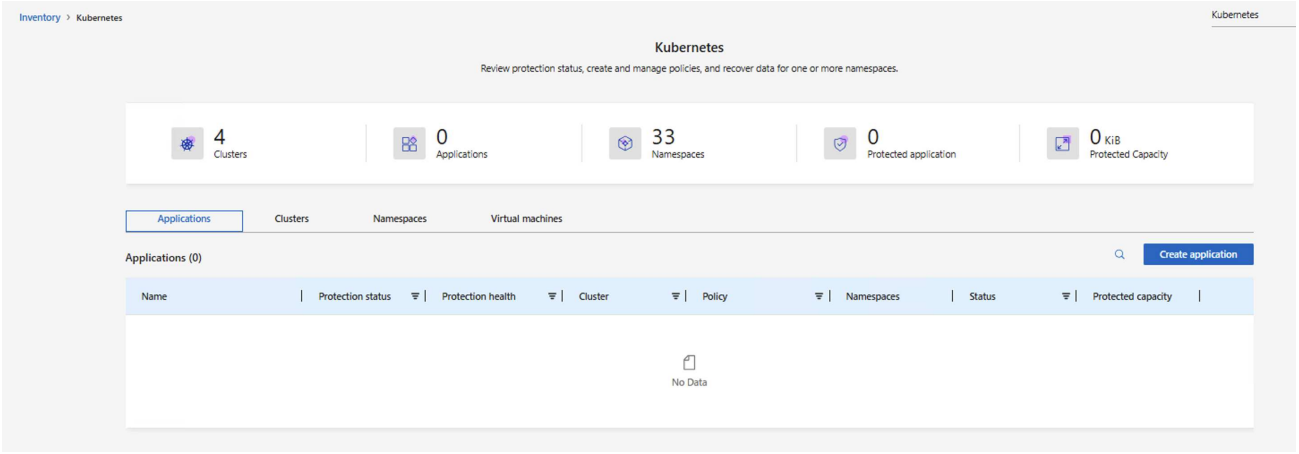
5. **In der NetApp Console eine Anwendung für die Container-Workloads erstellen und mit Backups schützen**

> Für das Erstellen und Schützen einer Kubernetes-Anwendung ist die Rolle Backup and Recovery super admin oder Backup and Recovery backup admin erforderlich. Für das Wiederherstellen einer Kubernetes-Anwendung ist die Rolle Backup and Recovery super admin oder Backup and Recovery restore admin erforderlich.

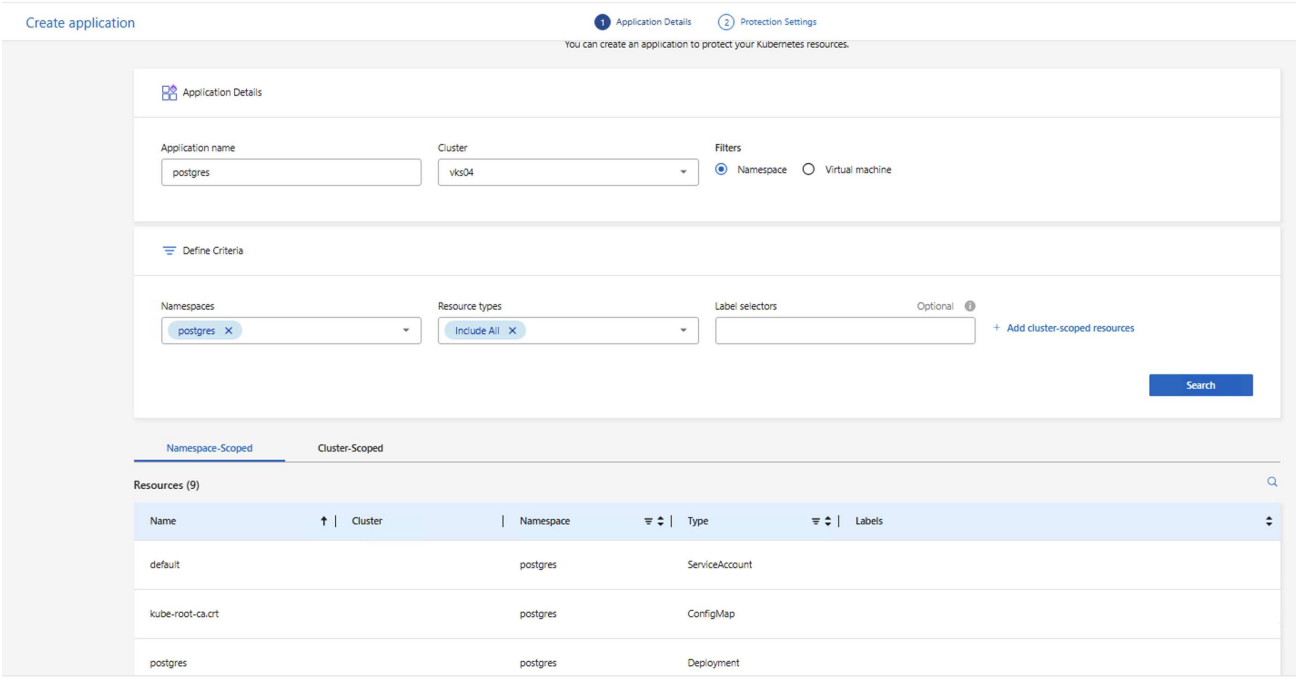
▼ **Beispiel anzeigen**

In diesem Schritt wird in der NetApp Console eine Anwendung für die Containeranwendungen im vSphere Kubernetes-Cluster erstellt, die geschützt werden müssen. Mit Namespaces kann eine Anwendung erstellt werden, um alle Containeranwendungen in einem Namespace im vSphere Kubernetes-Cluster zu schützen.

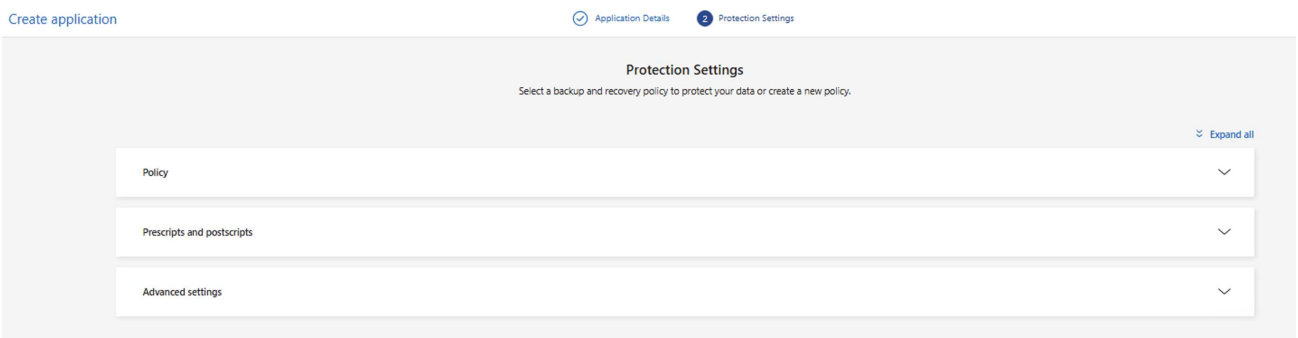
Sie benötigen außerdem eine Datensicherungsstrategie, die Sie der Anwendung zuordnen. Es kann eine neue Datensicherungsstrategie erstellt oder eine vorhandene verwendet werden.



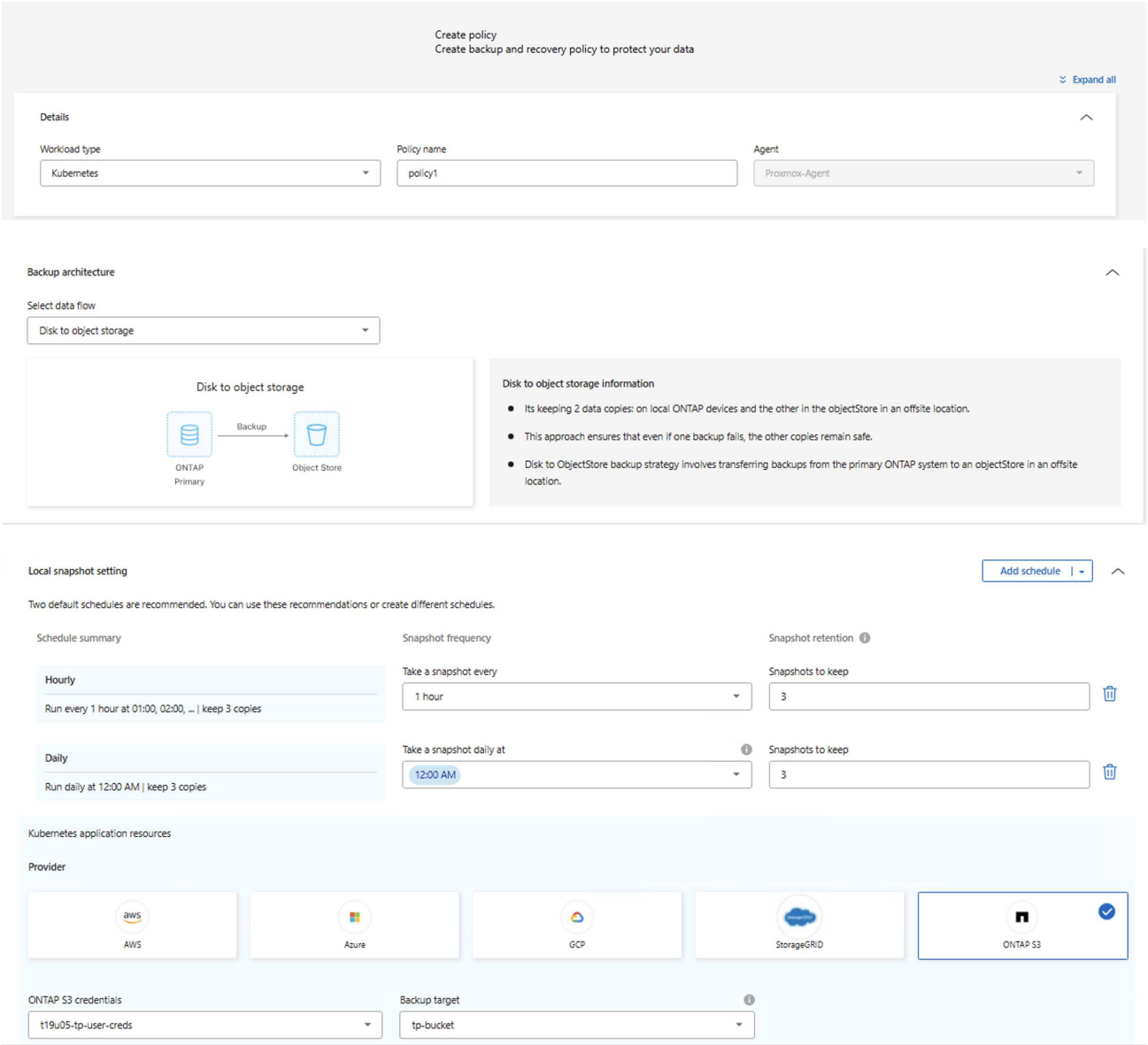
Anwendungsdetails werden bereitgestellt und Kriterien für die Anwendung definiert. In diesem Beispiel wird die Anwendung für alle Containeranwendungen in einem Namespace im vSphere Kubernetes-Cluster erstellt. Mit **Suchen** wird die Liste der Namespace-bezogenen Ressourcen angezeigt, die Teil der Anwendung sind. Mit **Weiter** wird fortgefahren.

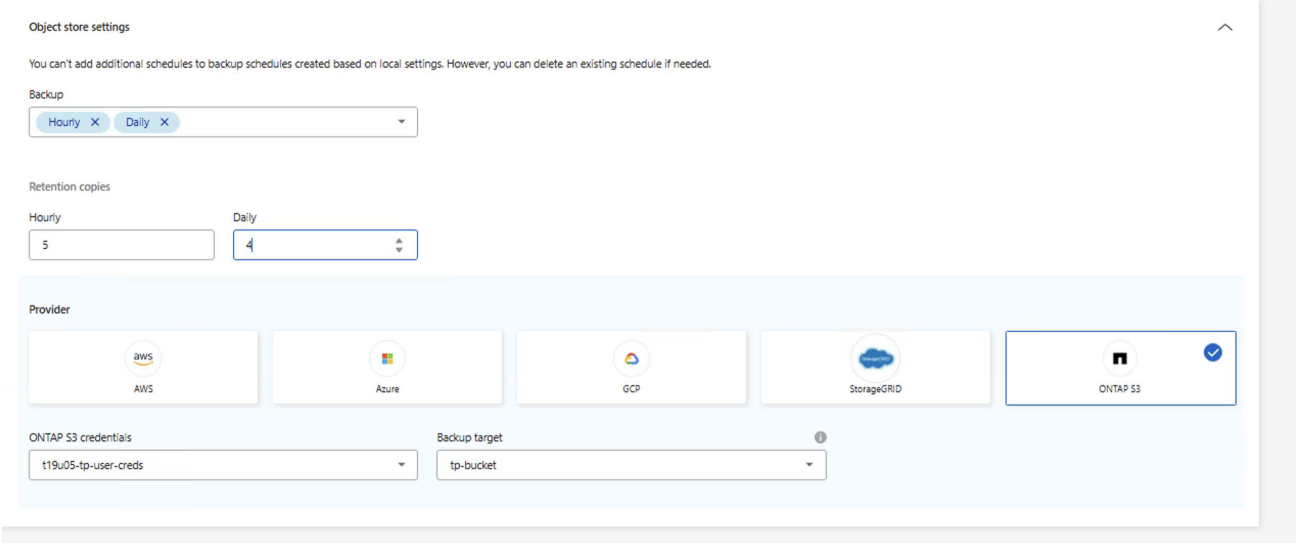


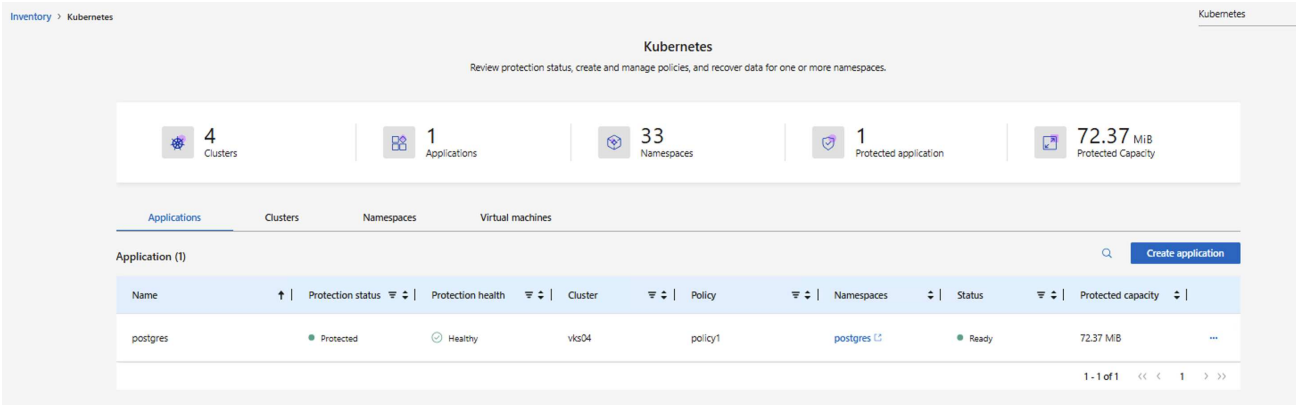
Als Nächstes sind die Schutzeinstellungen für die Anwendung anzugeben. Es kann eine neue Datensicherungsstrategie erstellt oder eine vorhandene verwendet werden. In diesem Beispiel wird eine neue Datensicherungsstrategie für die Anwendung erstellt und ihr zugeordnet. **Policy** erweitern und auf **Create new policy** klicken.



Ein Name für die Richtlinie kann angegeben und die Backup-Architektur ausgewählt werden. In diesem Beispiel wird Festplatte zu Objektspeicher gewählt. Der lokale Snapshot-Zeitplan wird definiert, der steuert, wie oft Snapshots auf dem primären Speicher erstellt werden. Anschließend werden die Backup-Einstellungen für den Objektspeicher separat konfiguriert: Das Backup-Ziel (in diesem Beispiel ONTAP S3) wird ausgewählt, der Übertragungszeitplan festgelegt, der steuert, wann Snapshot-Daten in den Objektspeicher kopiert werden, und die Anzahl der Aufbewahrungskopien angegeben. Der Übertragungszeitplan ist unabhängig vom Snapshot-Zeitplan, daher werden Ressourcen gemäß dem Übertragungszeitplan in den Objektspeicher kopiert und nicht unmittelbar nach jeder Snapshot-Erstellung. Auch die maximale Anzahl an Wiederholungsversuchen und das Wiederholungsintervall können bei Bedarf angepasst werden. Mit **Erstellen** wird fortgefahren. Die Anwendung wird erkannt und die Datensicherungsstrategie wird ihr zugeordnet.





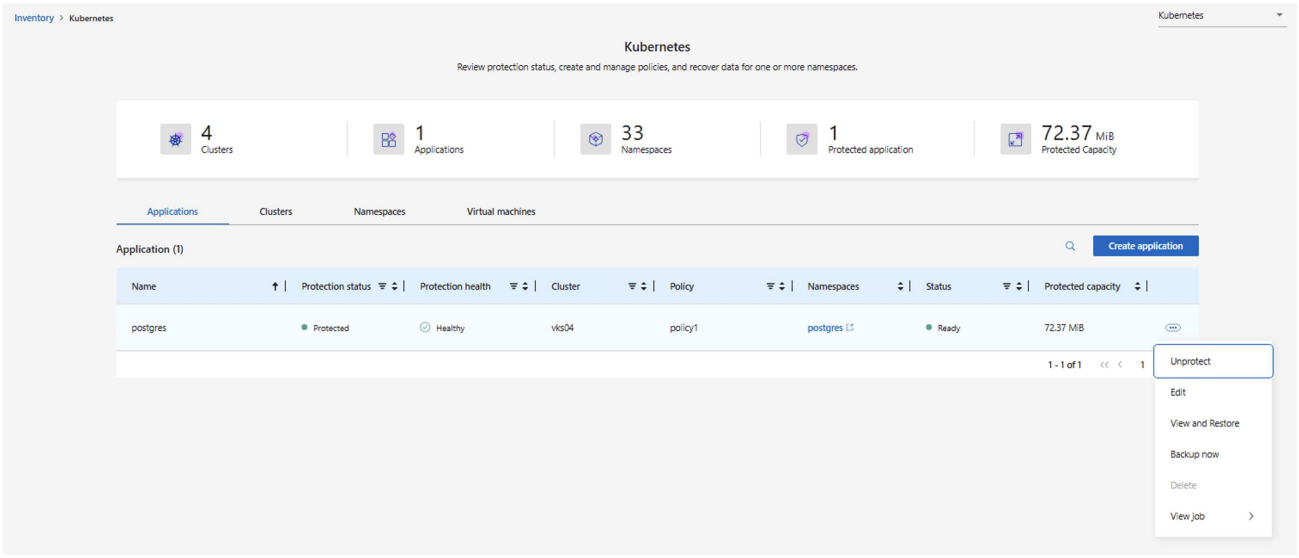


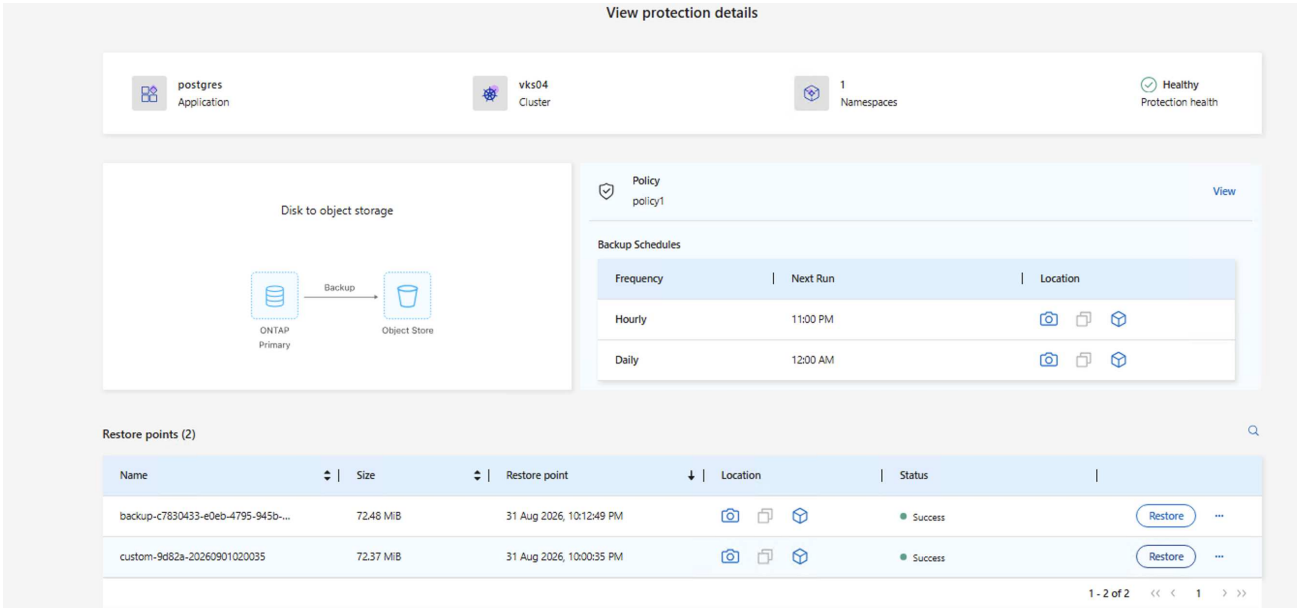
6. **Anwendung aus einem Backup wiederherstellen**

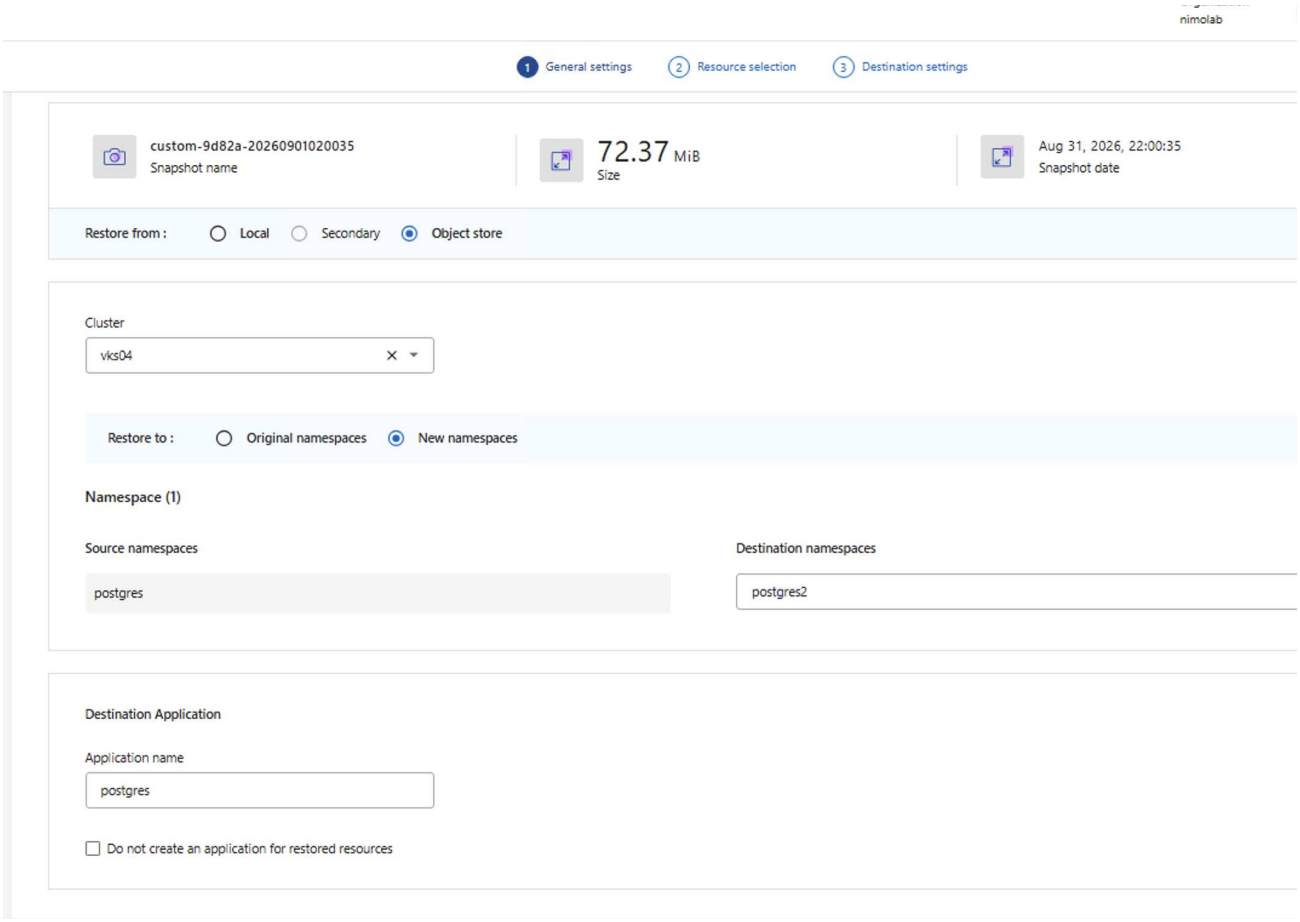
> Für die Wiederherstellung einer Kubernetes-Anwendung ist die Rolle Backup and Recovery super admin oder Backup and Recovery restore admin erforderlich.

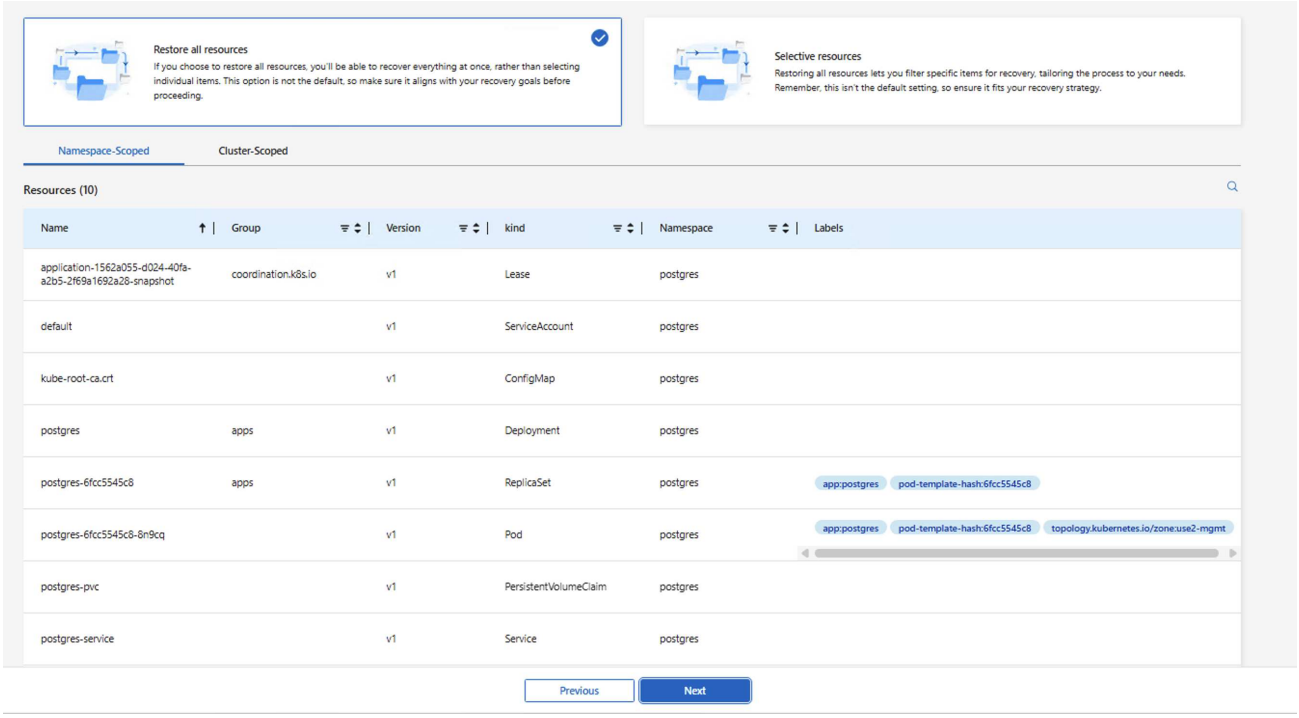
▼ Beispiel anzeigen

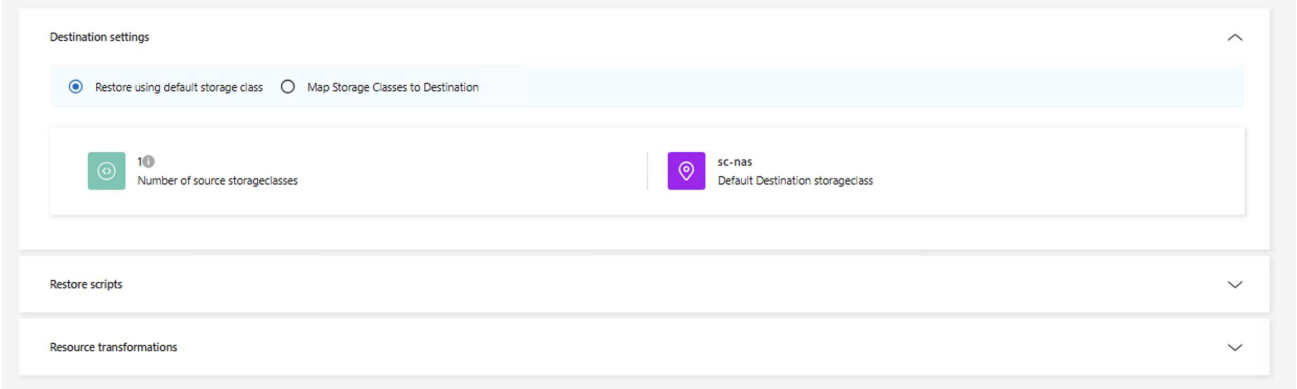
- Sie können jeden verfügbaren Wiederherstellungspunkt auswählen, also einen lokalen Snapshot, ein Objektspeicher-Backup oder ein Backup von einem sekundären Ziel, um die Anwendung wiederherzustellen.
- Sie können die Anwendung in ihrem ursprünglichen Namensraum oder in einem anderen Namensraum wiederherstellen.
- Sie können auswählen, welche Ressourcen in die Wiederherstellung einbezogen werden.
- Sie können die Speicherklasse auswählen, die für die wiederhergestellten Anwendungsressourcen verwendet werden soll.


Inventory › Kubernetes
Kubernetes
Kubernetes
Review protection status, create and manage policies, and recover data for one or more namespaces.
4 Clusters
1 Applications
33 Namespaces
1 Protected application
72.37 MiB Protected Capacity
Applications
Clusters
Namespaces
Virtual machines
Application (1)
Create application
Name
Protection status
Protection health
Cluster
Policy
Namespaces
Status
Protected capacity
postgres
Protected
Healthy
vks04
policy1
postgres
Ready
72.37 MiB
1 - 1 of 1
1
Unprotect
Edit
View and Restore
Backup now
Delete
View job

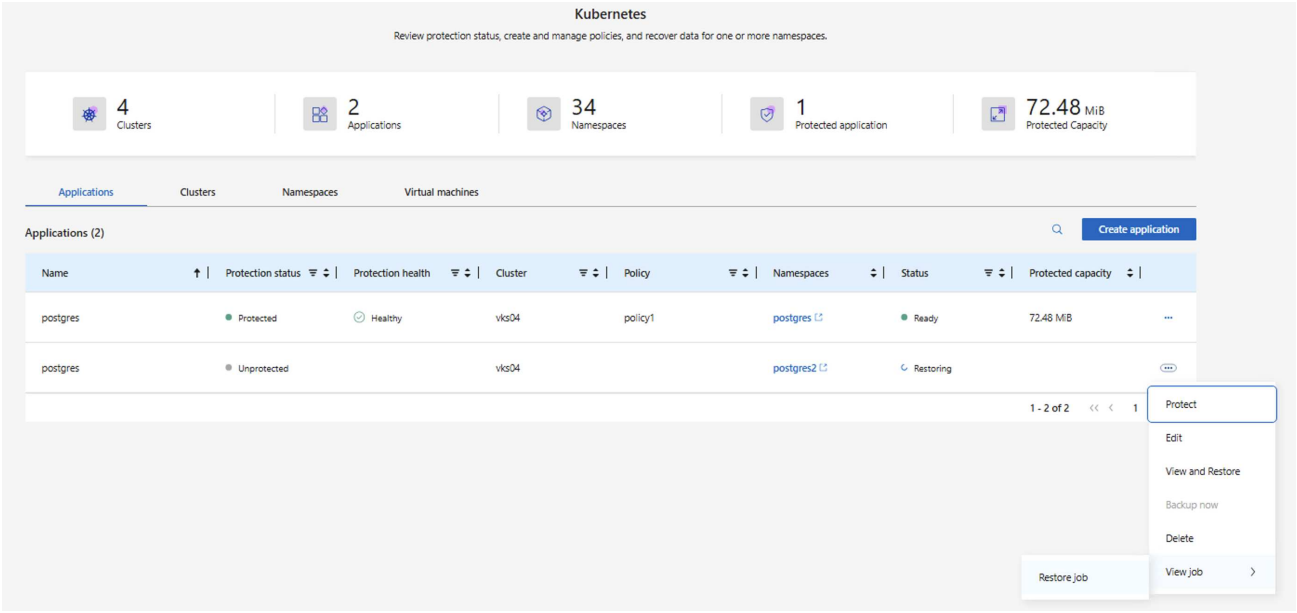

View protection details
postgres
Application
vks04
Cluster
1
Namespaces
Healthy
Protection health
Disk to object storage
Backup
ONTAP
Primary
Object Store
Policy
policy1
View
Backup Schedules
Frequency
Next Run
Location
Hourly
11:00 PM
Daily
12:00 AM
Restore points (2)
Name
Size
Restore point
Location
Status
backup-c7830433-e0eb-4795-945b-...
72.48 MiB
31 Aug 2026, 10:12:49 PM
Success
Restore
custom-9d82a-20260901020035
72.37 MiB
31 Aug 2026, 10:00:35 PM
Success
Restore
1 - 2 of 2
1

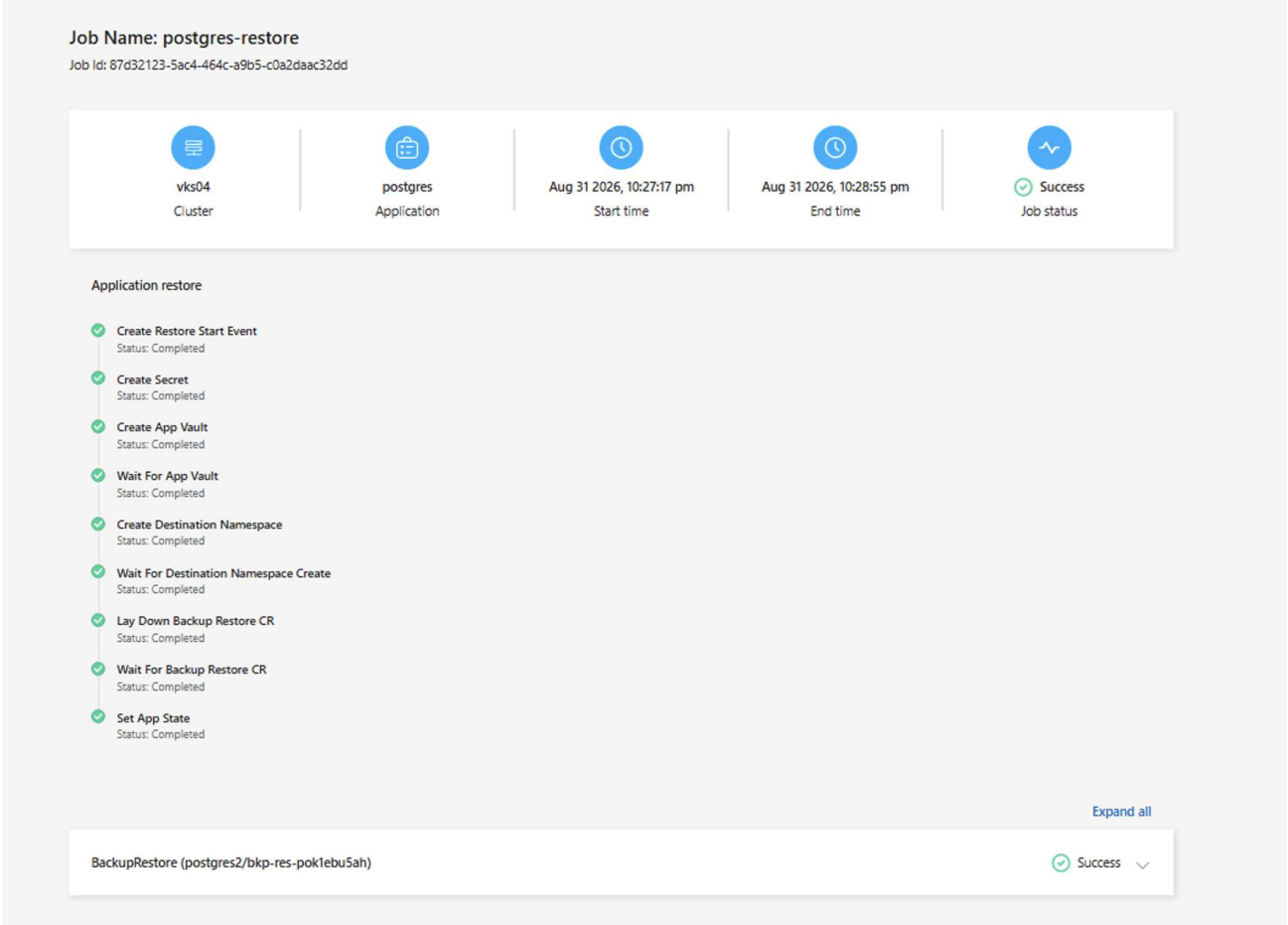

nimolab
1 General settings
2 Resource selection
3 Destination settings
custom-9d82a-20260901020035
Snapshot name
72.37 MiB
Size
Aug 31, 2026, 22:00:35
Snapshot date
Restore from :
Local
Secondary
Object store
Cluster
vks04
Restore to :
Original namespaces
New namespaces
Namespace (1)
Source namespaces
postgres
Destination namespaces
postgres2
Destination Application
Application name
postgres
Do not create an application for restored resources

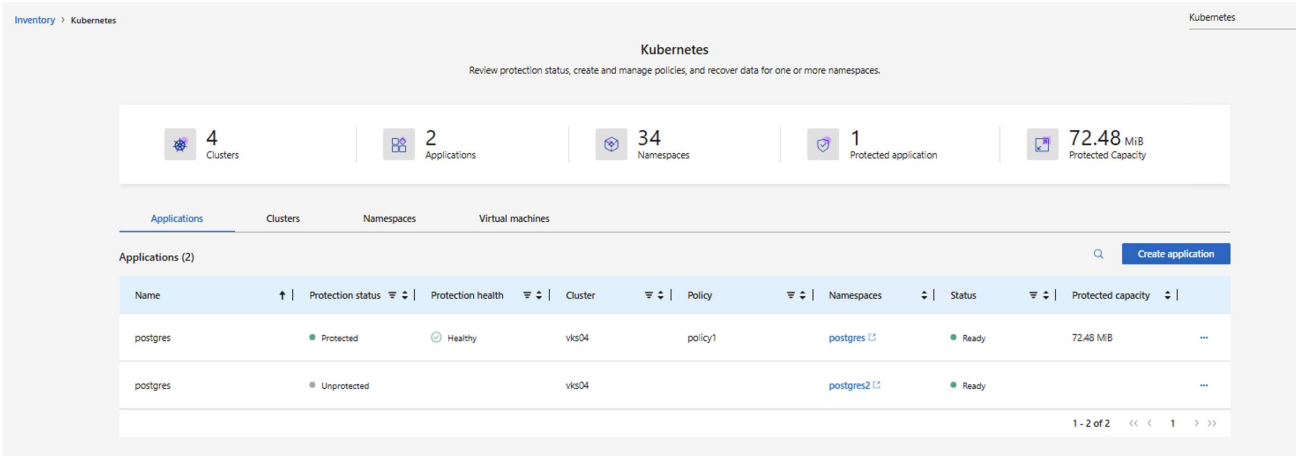

Restore all resources
If you choose to restore all resources, you'll be able to recover everything at once, rather than selecting individual items. This option is not the default, so make sure it aligns with your recovery goals before proceeding.
Selective resources
Restoring all resources lets you filter specific items for recovery, tailoring the process to your needs. Remember, this isn't the default setting, so ensure it fits your recovery strategy.
Namespace-Scoped
Cluster-Scoped
Resources (10)
Name
Group
Version
kind
Namespace
Labels
application-1562a055-d024-40fa-a2b5-2f69a1692a28-snapshot
coordination.k8s.io
v1
Lease
postgres
default
v1
ServiceAccount
postgres
kube-root-ca.crt
v1
ConfigMap
postgres
postgres
apps
v1
Deployment
postgres
postgres-6fcc5545c8
apps
v1
ReplicaSet
postgres
app:postgres
pod-template-hash:6fcc5545c8
postgres-6fcc5545c8-8n9cq
v1
Pod
postgres
app:postgres
pod-template-hash:6fcc5545c8
topology.kubernetes.io/zone:use2-mgmt
postgres-pvc
v1
PersistentVolumeClaim
postgres
postgres-service
v1
Service
postgres
Previous
Next



Im Abschnitt „Monitor“ der NetApp Console lässt sich der Fortschritt des Wiederherstellungsjobs beobachten. Nach Abschluss des Wiederherstellungsjobs sind die wiederhergestellten Ressourcen im vSphere Kubernetes-Cluster sichtbar.







Containeranwendungen in einem vSphere Kubernetes Service-Cluster mit Trident Protect sichern und wiederherstellen

Containeranwendungen in einem vSphere Kubernetes Service (VKS) Cluster mithilfe von Trident Protect Snapshots und Backups sichern und wiederherstellen. Dieses Verfahren umfasst die Erstellung eines AppVault mit ONTAP S3 Objektspeicher, die Konfiguration von Trident Protect zum Erfassen von Anwendungsdaten einschließlich Kubernetes-Ressourcenobjekten und persistenten Volumes sowie die Wiederherstellung der Daten bei Bedarf.

Container-Anwendungen, die in einem VKS-Cluster ausgeführt werden, werden als Kubernetes-Workloads in

Worker-Node-Namespaces verwaltet. Es ist wichtig, sowohl die Anwendungsmetadaten als auch die persistenten Volumes zu schützen, damit sie im Falle eines Verlusts oder einer Beschädigung wiederhergestellt werden können.

Die persistenten Volumes von VKS-Anwendungen können durch ONTAP Storage, der in den VKS-Cluster integriert ist, mit "Trident CSI" unterstützt werden. Dieses Verfahren verwendet "Trident Protect" zur Erstellung von Anwendungssnapshots, deren Metadaten im ONTAP Objektspeicher gespeichert werden, während die Volume-Snapshot-Daten im Storage-Backend verbleiben, sowie Backups, deren Daten in den ONTAP Objektspeicher kopiert werden. Eine Wiederherstellung ist sowohl aus einem Snapshot als auch aus einem Backup möglich.

Trident Protect ermöglicht Snapshots, Backups, Wiederherstellung und Notfallwiederherstellung von Anwendungen auf einem Kubernetes-Cluster. Zu den mit Trident Protect schutzfähigen Daten gehören Kubernetes-Ressourcenobjekte, die mit den Anwendungen und deren persistenten Volumes verknüpft sind.

Im Folgenden sind die Versionen der verschiedenen Komponenten aufgeführt, die in den Beispielen dieses Abschnitts verwendet werden.

- VMware Cloud Foundation (VCF) 9.1 mit vSphere Kubernetes Service
- "Trident 26.06"
- "Trident Protect 26.06"
- "ONTAP 9.16"

Trident Protect auf dem vSphere Kubernetes-Cluster installieren

▼ **Installieren Sie Trident Protect**

Helm wird verwendet, um Trident Protect auf dem vSphere Kubernetes Service Cluster zu installieren. Die Beispiele in diesem Abschnitt verwenden `tridentctl-protect`, die Trident Protect CLI. Die Installation erfolgt gemäß den Anweisungen in der "Trident Protect CLI Dokumentation". Weitere Methoden zur Installation von Trident Protect sind in der "Trident Protect Dokumentation" dokumentiert.

Zuerst den `trident-protect`Namespace erstellen und kennzeichnen. Das `enforce=privileged`Label ist erforderlich, da Trident Protect privilegierte Workloads ausführt. Ohne dieses Label verhindert der Kubernetes Pod Security Admission Controller den Start der Trident Protect Pods.

```
kubectl create namespace trident-protect
kubectl label namespace trident-protect pod-
security.kubernetes.io/enforce=privileged
```

Dann das Helm-Repository hinzufügen und Trident Protect im Namespace installieren.

```
helm repo add netapp-trident-protect https://netapp.github.io/trident-
protect-helm-chart
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --version 100.2606.0 --namespace
trident-protect
```

Falls Trident Protect Pods aufgrund eines `PodSecurity` Zulassungsfehlers nicht starten,

wurden die Pod-Sicherheitslabels möglicherweise nicht auf den `trident-protect` Namespace vor der Installation angewendet. Mit `--overwrite` können sie erneut angewendet werden, anschließend lässt sich überprüfen, ob die Pods starten.

```
kubectl label namespace trident-protect pod-
security.kubernetes.io/enforce=privileged --overwrite
```

```
kubectl get pods -n trident-protect
```

```
vksbastion@vks:~/demo-vks$ kubectl get pods -n trident-protect
NAME                                                  READY   STATUS    RESTARTS   AGE
autosupportbundle-e4cb1693-6640-4fcd-bddf-b413ff41f37c-znvsw   1/1     Running   0          16h
trident-protect-controller-manager-5f64ff594f-cl8cp           1/1     Running   0          3h50m
vksbastion@vks:~/demo-vks$
```

```
vksbastion@vks:~/demo-vks$ tridentctl-protect version -n trident-protect
26.06.0
vksbastion@vks:~/demo-vks$
```

App Vault für Objektspeicher erstellen

▼ **Erstellen AppVault**

Bevor Snapshots und Backups für eine Anwendung erstellt werden, muss der Objektspeicher in Trident Protect konfiguriert werden. Dies erfolgt durch das Erstellen eines AppVault CR. Nur Administratoren können ein AppVault CR erstellen und konfigurieren.

AppVault-Objekte sind die Kubernetes Custom Resource-Darstellung eines Speicher-Buckets. Ein AppVault CR enthält die Konfigurationen, die erforderlich sind, damit ein Bucket in Datensicherungsoperationen wie Backups, Snapshots, Wiederherstellungsvorgängen und SnapMirror-Replikation verwendet werden kann.

Die folgenden Schritte erstellen eine für ONTAP S3 konfigurierte AppVault CR: . Erstellen eines S3-Objektspeicherservers in der SVM im ONTAP Cluster. . Erstellen eines Buckets im Objektspeicherserver. . Erstellen eines S3-Benutzers in der SVM. Den Zugriffsschlüssel und den geheimen Schlüssel an einem sicheren Ort aufbewahren.

+ HINWEIS: Trident Protect erfordert, dass der S3-Benutzer mindestens `PutObject`, `GetObject`, `ListBucket` und `DeleteObject` Berechtigungen besitzt. Ohne diese Berechtigungen schlagen AppVault Backup- und Wiederherstellungsvorgänge mit Zugriffsverweigerungsfehlern fehl. Weitere Informationen finden sich unter "Trident Protect AppVault Dokumentation". . Im VKS-Cluster ein Secret zum Speichern der ONTAP S3-Anmeldeinformationen erstellen. . Ein AppVault Objekt für ONTAP S3 erstellen.

Trident Protect AppVault für ONTAP S3 konfigurieren

Das untenstehende Manifest verwendet HTTPS mit aktivierter Zertifikatsvalidierung, was für den Produktivbetrieb erforderlich ist. Wenn Ihr ONTAP S3-Endpunkt ein von einer öffentlichen Zertifizierungsstelle signiertes Zertifikat verwendet, das vom Cluster bereits als vertrauenswürdig eingestuft wird, ist keine zusätzliche Zertifikatskonfiguration notwendig. Wenn ein selbstsigniertes oder internes Zertifikat einer Zertifizierungsstelle verwendet wird, ist das benutzerdefinierte Stammzertifikat der Zertifizierungsstelle im `rootCA` Feld wie im Manifest gezeigt bereitzustellen. Am Ende dieses Abschnitts findet sich eine ausschließlich

für Labore vorgesehene Variante, falls eine HTTP-Konfiguration für isolierte Tests außerhalb des Produktivbetriebs benötigt wird.

```
# alias tp='tridentctl-protect'

# cat appvault-secret.yaml
apiVersion: v1
data:
  accessKeyID: <base64 encoded access key>
  secretAccessKey: <base64 encoded secret access key>
# Alternatively, remove the data section above and use:
# stringData:
#   accessKeyID: "<access key of S3>"
#   secretAccessKey: "<secret access key of S3>"
kind: Secret
metadata:
  name: ontap-s3-appvault-secret1
  namespace: trident-protect
type: Opaque

# cat appvault.yaml
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: ontap-s3-appvault1
  namespace: trident-protect
spec:
  providerConfig:
    azure:
      accountName: ""
      bucketName: ""
      endpoint: ""
    gcp:
      bucketName: ""
      projectID: ""
    s3:
      bucketName: trident-protect
      endpoint: <lif for S3 access>
      secure: "true"
      skipCertValidation: "false"
      # If your ONTAP S3 endpoint uses a certificate signed by a public
CA
      # already trusted by the cluster, no additional certificate config
is needed.
      # If it uses a self-signed or internal CA certificate, provide the
      # root CA certificate:
```

```
      # rootCA: <root CA certificate>
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: ontap-s3-appvault-secret1
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: ontap-s3-appvault-secret1
  providerType: OntapS3

# kubectl create -f appvault-secret.yaml -n trident-protect
# kubectl create -f appvault.yaml -n trident-protect
```

Nur für Laborzwecke (HTTP, keine Zertifikatsprüfung)

Nur die folgenden `s3`Einstellungen in einer isolierten Laborumgebung verwenden, in der keine sensiblen Daten vorhanden sind. Diese Einstellungen nicht in der Produktion verwenden.

```
    s3:
      bucketName: trident-protect
      endpoint: <lif for S3 access>
      secure: "false"
      skipCertValidation: "true"
```

```
vksbastion@vks:~/demo-vks/tp$ kubectl get appvault -n trident-protect
NAME                STATE       ERROR   MESSAGE   AGE
ontap-s3-appvault1  Available                     3d14h
vks-appvault        Available                     5d15h
vksbastion@vks:~/demo-vks/tp$
```

Eine Trident Protect Anwendung erstellen

▼ **Eine Trident Protect Anwendung erstellen**

Dieses Beispiel behandelt den Datenschutz für die Beispielanwendung postgres, die im Namespace postgres installiert ist. Einzelheiten zur Installation finden sich unter "VKS-Workloads mit NetApp Storage bereitstellen".

> ⓘ Die Beispiel-PostgreSQL-PVCs fordern den RWO-Zugriffsmodus an. Der Zugriffsmodus muss im PVC-Manifest explizit angegeben werden; Trident wählt keinen Zugriffsmodus automatisch anhand des Speicherprotokolls aus. RWX wird für NAS-basierte PVCs unterstützt, und SAN-basierte PVCs können RWX mit zusätzlicher Konfiguration unterstützen. Weitere Informationen finden sich in der "Trident Protect Dokumentation".

Im Beispiel enthält der Namespace postgres eine Anwendung, und alle Ressourcen des Namespace werden beim Erstellen der Trident Protect Application einbezogen.

```
# alias tp='tridentctl-protect'
# tp create app postgres-app --namespaces postgres -n postgres --dry-run
> postgres-app.yaml

# cat postgres-app.yaml
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: postgres-app
spec:
  includedNamespaces:
  - namespace: postgres
  resourceFilter: {}
# kubectl create -f postgres-app.yaml -n postgres
```

```
vksbastion@vks:~/demo-vks/tp$ kubectl get application -n postgres
NAME           PROTECTION STATE   AGE
postgres-app   None               3d13h
```

Die App wird durch das Erstellen eines Backups geschützt.

▼ **Backups erstellen**

On-Demand-Backup erstellen

Erstellen Sie eine Sicherung für die zuvor erstellte postgres-app, die alle Ressourcen im postgres-Namespace umfasst. Geben Sie den Namen des appvault an, in dem die Sicherungen gespeichert werden.

```
# cat postgres-backup-on-demand.yaml
apiVersion: protect.trident.netapp.io/v1
kind: Backup
metadata:
  name: postgres-backup-on-demand
spec:
  appVaultRef: ontap-s3-appvault1
  applicationRef: postgres-app
  cleanupSnapshot: true
  replicateSnapshot: false
```

```
kubectl create -f postgres-backup-on-demand.yaml -n postgres
```

```
vksbastion@vks:~/demo-vks/tp$ kubectl create -f postgres-backup-on-demand.yaml -n postgres
backup.protect.trident.netapp.io/postgres-backup-on-demand created
vksbastion@vks:~/demo-vks/tp$ kubectl get backups -n postgres
NAME                        APP            RECLAIM POLICY   STATE     ERROR   AGE
postgres-backup-on-demand   postgres-app   Retain           Running           12s
vksbastion@vks:~/demo-vks/tp$ kubectl get backups -n postgres -w
NAME                        APP            RECLAIM POLICY   STATE     ERROR   AGE
postgres-backup-on-demand   postgres-app   Retain           Running           29s
postgres-backup-on-demand   postgres-app   Retain           Running           82s
postgres-backup-on-demand   postgres-app   Retain           Running           82s
postgres-backup-on-demand   postgres-app   Retain           Running           82s
postgres-backup-on-demand   postgres-app   Retain           Running           83s
postgres-backup-on-demand   postgres-app   Retain           Running           83s
postgres-backup-on-demand   postgres-app   Retain           Running           83s
postgres-backup-on-demand   postgres-app   Retain           Completed          83s
```

```
vksbastion@vks:~/demo-vks/tp$ kubectl get application -n postgres
NAME           PROTECTION STATE   AGE
postgres-app   Partial            3d13h
```

Backups nach Zeitplan erstellen

Erstellen Sie einen Zeitplan für die Backups, wobei die Granularität und die Anzahl der aufzubewahrenden Backups festgelegt werden.

```
# tp create schedule backup-schedule1 --app postgres-app --appvault
ontap-s3-appvault1 --granularity Hourly --minute 45 --backup-retention 1
-n postgres --dry-run>postgres-backup-schedule1.yaml

#cat postgres-backup-schedule1.yaml
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  name: backup-schedule1
  namespace: postgres
spec:
  appVaultRef: ontap-s3-appvault1
  applicationRef: postgres-app
  backupRetention: "1"
  dayOfMonth: ""
  dayOfWeek: ""
  enabled: true
  granularity: Hourly
  hour: ""
  minute: "45"
  recurrenceRule: ""
  replicationRetention: "0"
  runImmediately: false
  snapshotRetention: "0"
# kubectl create -f postgres-backup-schedule1.yaml -n postgres
```

```
vksbastion@vks:~/demo-vks/tp$ kubectl create -f postgres-backup-schedule1.yaml -n postgres
schedule.protect.trident.netapp.io/backup-schedule1 created
```

```
vksbastion@vks:~/demo-vks/tp$ kubectl get schedule -n postgres
NAME               ENABLED   GRANULARITY   STATE   ERROR   AGE
backup-schedule1   true      Hourly                        10m
```

Wiederherstellung aus Backup

▼ **Wiederherstellung aus Backups**

Wiederherstellung der Anwendung im selben Namensraum

In diesem Beispiel enthält das Backup postgres-backup-on-demand das Backup für die postgres-app.

Bevor die Anwendung gelöscht wird, sollte überprüft werden, dass sich die Sicherung, von der wiederhergestellt werden soll, im `Completed` Zustand befindet. Eine laufende oder fehlgeschlagene Sicherung kann nicht zur Wiederherstellung der Anwendung verwendet werden.

```
kubectl get backup -n postgres
```

Nachdem bestätigt wurde, dass der Backup-Status `Completed` ist, die postgres-Anwendung löschen und sicherstellen, dass die PVCs und Pod-Objekte aus dem Namespace "postgres" entfernt werden.

```
vksbastion@vks:~/demo-vks/tp$ kubectl get all -n postgres
NAME                            READY   STATUS    RESTARTS   AGE
pod/postgres-6fcc5545c8-6gw9h   1/1     Running   0          3d13h

NAME                       TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/postgres-service   ClusterIP   10.101.183.142   <none>        5432/TCP   3d13h

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/postgres   1/1     1            1           3d13h

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/postgres-6fcc5545c8   1         1         1       3d13h
vksbastion@vks:~/demo-vks/tp$ kubectl delete deployment/postgres -n postgres
deployment.apps "postgres" deleted from postgres namespace
vksbastion@vks:~/demo-vks/tp$ kubectl get all -n postgres
NAME                       TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/postgres-service   ClusterIP   10.101.183.142   <none>        5432/TCP   3d13h
```

```
vksbastion@vks:~/demo-vks/tp$ kubectl get pvc -n postgres
NAME           STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   V
OLUMEATTRIBUTESCLASS   AGE
postgres-pvc   Bound    pvc-43f275d5-6063-4751-98d5-3a36b07d0bf0   10Gi       RWO            sc-nas         <
unset>                 3d14h
```

```
vksbastion@vks:~/demo-vks/tp$ kubectl get all,pvc -n postgres
No resources found in postgres namespace.
```

Nun wird ein Backup-in-Place-Wiederherstellungsobjekt erstellt.

```
# tp create bir postgres-app-restore --backup postgres/postgres-backup-
on-demand -n postgres --dry-run>postgres-app-bir.yaml
```

```
# cat postgres-app-bir.yaml
apiVersion: protect.trident.netapp.io/v1
kind: BackupInplaceRestore
metadata:
  annotations:
    protect.trident.netapp.io/max-parallel-restore-jobs: "25"
  name: postgres-app-restore
  namespace: postgres
spec:
  appArchivePath: postgres-app_314bbaf6-2ce3-4065-b3c1-
ab85c7bb7c7c/backups/postgres-backup-on-demand_63efc9d1-92d7-45ce-84fe-
846ce7db7b29
  appVaultRef: ontap-s3-appvault1
  cleanUpAdditionalExecHooks: true
  cleanUpArchivedExecHooks: false
  resourceFilter: {}
  runArchivedExecHooks: true

# kubectl create -f postgres-app-bir.yaml -n postgres
```

```
vksbastion@vks:~/demo-vks/tp$ kubectl create -f postgres-app-bir.yaml -n postgres
backupinplacerestore.protect.trident.netapp.io/postgres-app-restore created
```

Es wird überprüft, ob die PostgreSQL-Anwendungsbereitstellung, der Service, das Pod und die PVCs wiederhergestellt sind.

```
vksbastion@vks:~/demo-vks/tp$ kubectl get all,pvc -n postgres
NAME                            READY   STATUS    RESTARTS   AGE
pod/postgres-6fcc5545c8-9pv2m   1/1     Running   0          2m1s

NAME                       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/postgres-service   ClusterIP   10.107.38.167   <none>        5432/TCP   2m1s

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/postgres   1/1     1            1           2m1s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/postgres-6fcc5545c8   1         1         1       2m1s

NAME                                 STATUS   VOLUME                                     CAPACITY   ACCESS MO
DES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
persistentvolumeclaim/postgres-pvc   Bound    pvc-191af706-64ac-4f09-921a-ff9bf6d86a68   10Gi       RWO
      sc-nas         <unset>                 2m6s
```

Wiederherstellung der Anwendung in einem anderen Namensraum

Zuerst einen neuen Namespace erstellen, in dem die App wiederhergestellt werden soll, in diesem Beispiel postgres2. Ein stündliches Backup, das durch den Zeitplan erstellt wurde, ist jetzt für die postgres-app verfügbar. Dieses Backup kann verwendet werden, um die Anwendung im neuen Namespace postgres2 wiederherzustellen.

```
vksbastion@vks:~/demo-vks/tp$ kubectl get backup -n postgres
NAME                              APP            RECLAIM POLICY   STATE       ERROR   AGE
hourly-cbd11-20260831104500   postgres-app   Retain           Completed           14m
postgres-backup-on-demand     postgres-app   Retain           Completed           51m
```

```
# tp create backuprestore --appvault ontap-s3-appvault1 --path postgres-
app_314bbaf6-2ce3-4065-b3c1-ab85c7bb7c7c/backups/hourly-cbd11-
20260831124500_2b57bda9-5cb0-4c8f-ae7a-20f94c23c3b9 --namespace-mapping
postgres:postgres2 -n postgres2 --dry-run>postgres-app-postgres2-br.yaml
```

Um den Pfad für die Sicherung zu erhalten, verwenden Sie den Befehl `kubectl get backups <backup-name> -n postgres -o jsonpath='{.status.appArchivePath}'`.

```
vksbastion@vks:~/demo-vks/tp$ kubectl get backups -n postgres
NAME                          APP            RECLAIM POLICY   STATE       ERROR   AGE
hourly-cbd11-20260831124500   postgres-app   Retain           Completed           13m
postgres-backup-on-demand     postgres-app   Retain           Completed           170m
vksbastion@vks:~/demo-vks/tp$ tp get backups -n postgres -o yaml | grep hourly-cbd11-20260831124500
    name: hourly-cbd11-20260831124500
    appArchivePath: postgres-app_314bbaf6-2ce3-4065-b3c1-ab85c7bb7c7c/backups/hourly-cbd11-20260831124500_2b5
7bda9-5cb0-4c8f-ae7a-20f94c23c3b9
vksbastion@vks:~/demo-vks/tp$
```

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupRestore
metadata:
  annotations:
    protect.trident.netapp.io/max-parallel-restore-jobs: "25"
  name: postgres-app-z2woek
  namespace: postgres2
spec:
  appArchivePath: postgres-app_314bbaf6-2ce3-4065-b3c1-
ab85c7bb7c7c/backups/hourly-cbd11-20260831124500_2b57bda9-5cb0-4c8f-
ae7a-20f94c23c3b9
  appVaultRef: ontap-s3-appvault1
  cleanUpAdditionalExecHooks: true
  cleanUpArchivedExecHooks: false
  namespaceMapping:
  - destination: postgres2
    source: postgres
  resourceFilter: {}
  runArchivedExecHooks: true
  skipApplicationCreation: false

# kubectl create -f postgres-app-postgres2-br.yaml -n postgres2
```

```
vksbastion@vks:~/demo-vks/tp$ tp create backuprestore --appvault ontap-s3-appvault1 --path postgres-app_314bb
af6-2ce3-4065-b3c1-ab85c7bb7c7c/backups/hourly-cbd11-20260831124500_2b57bda9-5cb0-4c8f-ae7a-20f94c23c3b9 --na
mespace-mapping postgres:postgres2 -n postgres2 --dry-run>postgres-app-postgres2-br.yaml
vksbastion@vks:~/demo-vks/tp$
vksbastion@vks:~/demo-vks/tp$ kubectl create -f postgres-app-postgres2-br.yaml -n postgres2
backuprestore.protect.trident.netapp.io/postgres-app-btguyt created
```

Es sollte überprüft werden, ob die Postgres-Anwendungsobjekte und PVCs im neuen Namespace postgres2

erstellt wurden.

```
vksbastion@vks:~/demo-vks/tp$ kubectl get all,pvc -n postgres2
NAME                            READY   STATUS    RESTARTS   AGE
pod/postgres-6fcc5545c8-858dt   1/1     Running   0          72s

NAME                       TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
service/postgres-service   ClusterIP   10.109.5.180   <none>        5432/TCP   72s

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/postgres   1/1     1            1           72s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/postgres-6fcc5545c8   1         1         1       72s

NAME                                 STATUS   VOLUME                                     CAPACITY   ACCESS MO
DES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
persistentvolumeclaim/postgres-pvc   Bound    pvc-5893851c-c152-439f-a147-c4ae3581cc6f   10Gi       RWO
      sc-nas         <unset>                 78s
vksbastion@vks:~/demo-vks/tp$
```

Die App mit Snapshots schützen

▼ Snapshots erstellen

Erstellen eines On-Demand-Snapshots Ein Snapshot für die App wird erstellt und der AppVault angegeben, in dem die Snapshot-Metadaten gespeichert werden. Die Volume-Snapshot-Daten selbst werden nicht in den Objektspeicher kopiert, sondern verbleiben im Storage-Backend.

```
# tp create snapshot postgres-app-snapshot-ondemand --app postgres-app
--appvault ontap-s3-appvault1 -n postgres --dry-run>postgres-app
-snapshot-ondemand.yaml

# cat postgres-app-snapshot-ondemand.yaml
apiVersion: protect.trident.netapp.io/v1
kind: Snapshot
metadata:
  name: postgres-app-snapshot-ondemand
  namespace: postgres
spec:
  appVaultRef: ontap-s3-appvault1
  applicationRef: postgres-app
  cleanupSnapshot: false
  completionTimeout: 0s
  volumeSnapshotsCreatedTimeout: 0s
  volumeSnapshotsReadyToUseTimeout: 0s

# kubectl create -f postgres-app-snapshot-ondemand.yaml
snapshot.protect.trident.netapp.io/postgres-app-snapshot-ondemand
created
```

```
vksbastion@vks:~/demo-vks/tp$ kubectl create -f postgres-app-snapshot-ondemand.yaml
snapshot.protect.trident.netapp.io/postgres-app-snapshot-ondemand created
vksbastion@vks:~/demo-vks/tp$
vksbastion@vks:~/demo-vks/tp$ kubectl get snapshots -n postgres
NAME                              APP            RECLAIM POLICY   STATE       ERROR   AGE
postgres-app-snapshot-ondemand    postgres-app   Delete           Completed           20s
vksbastion@vks:~/demo-vks/tp$
```

Zeitplan für Snapshots erstellen Ein Zeitplan für die Snapshots wird erstellt. Die Granularität und die Anzahl der aufzubewahrenden Snapshots werden angegeben.

```
# tp create schedule snapshot-schedule1 --app postgres-app --appvault
ontap-s3-appvault1 --granularity Hourly --minute 55 --snapshot-retention
1 -n postgres --dry-run>postgres-app-snapshot-schedule1.yaml

# cat postgres-app-snapshot-schedule1.yaml
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  name: snapshot-schedule1
  namespace: postgres
spec:
  appVaultRef: ontap-s3-appvault1
  applicationRef: postgres-app
  backupRetention: "0"
  dayOfMonth: ""
  dayOfWeek: ""
  enabled: true
  granularity: Hourly
  hour: ""
  minute: "55"
  recurrenceRule: ""
  replicationRetention: "0"
  runImmediately: false
  snapshotRetention: "1"

# kubectl create -f postgres-app-snapshot-schedule1.yaml
schedule.protect.trident.netapp.io/snapshot-schedule1 created
```

```
vksbastion@vks:~/demo-vks/tp$ kubectl create -f postgres-app-snapshot-schedule1.yaml -n postgres
schedule.protect.trident.netapp.io/snapshot-schedule1 created
vksbastion@vks:~/demo-vks/tp$ kubectl get schedule -n postgres
NAME                 ENABLED   GRANULARITY   STATE   ERROR   AGE
backup-schedule1     true      Hourly                        3h37m
snapshot-schedule1   true      Hourly                        10s
vksbastion@vks:~/demo-vks/tp$ kubectl get snapshots -n postgres -w
NAME                            APP            RECLAIM POLICY   STATE       ERROR   AGE
postgres-app-snapshot-ondemand  postgres-app   Delete           Completed           32m
hourly-f1dd9-20260831135500     postgres-app   Delete                               0s
hourly-f1dd9-20260831135500     postgres-app   Delete                               0s
hourly-f1dd9-20260831135500     postgres-app   Delete                               0s
hourly-f1dd9-20260831135500     postgres-app   Delete                               0s
hourly-f1dd9-20260831135500     postgres-app   Delete                               0s
hourly-f1dd9-20260831135500     postgres-app   Delete                               0s
hourly-f1dd9-20260831135500     postgres-app   Delete                               0s
hourly-f1dd9-20260831135500     postgres-app   Delete                               0s
hourly-f1dd9-20260831135500     postgres-app   Delete                               0s
hourly-f1dd9-20260831135500     postgres-app   Delete                               0s
hourly-f1dd9-20260831135500     postgres-app   Delete           Running             0s
hourly-f1dd9-20260831135500     postgres-app   Delete           Running             0s
hourly-f1dd9-20260831135500     postgres-app   Delete           Running             0s
hourly-f1dd9-20260831135500     postgres-app   Delete           Running             0s
^Cvksbastion@vks:~/demo-vks/tp$ kubectl get snapshots -n postgres
NAME                            APP            RECLAIM POLICY   STATE       ERROR   AGE
hourly-f1dd9-20260831135500     postgres-app   Delete           Completed           15s
postgres-app-snapshot-ondemand  postgres-app   Delete           Completed           33m
```

Wiederherstellung aus Snapshot

▼ **Wiederherstellung aus Snapshot**

Wiederherstellung der Anwendung aus einem Snapshot im selben Namespace

Bevor Sie die Anwendung löschen, vergewissern Sie sich, dass sich der Snapshot, von dem Sie die Anwendung wiederherstellen möchten, im `Completed` Status befindet. Ein Snapshot, der sich noch in Bearbeitung befindet oder fehlgeschlagen ist, kann nicht zur Wiederherstellung der Anwendung verwendet werden.

```
kubectl get snapshot -n postgres
```

Nachdem bestätigt wurde, dass der Snapshot-Status `Completed` korrekt ist, werden die Anwendungsobjekte und die PVCs für Postgres aus dem Postgres-Namespace gelöscht.

```
vksbastion@vks:~/demo-vks/tp$ kubectl get deployment,svc,pvc -n postgres
NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/postgres   1/1     1            1           3h14m

NAME                       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/postgres-service   ClusterIP   10.107.38.167   <none>        5432/TCP   3h14m

NAME                                STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
VOLUMEATTRIBUTESCLASS   AGE
persistentvolumeclaim/postgres-pvc  Bound    pvc-191af706-64ac-4f09-921a-ff9bf6d86a68   10Gi       RWO            sc-nas
<unset>                 3h14m
vksbastion@vks:~/demo-vks/tp$ kubectl delete deployment/postgres -n postgres
deployment.apps "postgres" deleted from postgres namespace
vksbastion@vks:~/demo-vks/tp$ kubectl delete service/postgres-service -n postgres
service "postgres-service" deleted from postgres namespace
vksbastion@vks:~/demo-vks/tp$ kubectl delete pvc/postgres-pvc -n postgres
persistentvolumeclaim "postgres-pvc" deleted from postgres namespace
vksbastion@vks:~/demo-vks/tp$ kubectl get deployment,svc,pvc -n postgres
No resources found in postgres namespace.
vksbastion@vks:~/demo-vks/tp$ |
```

Ein Snapshot-in-Place-Restore-Objekt aus dem Snapshot erstellen.

```
# tp create sir postgres-restore-from-snapshot --snapshot
postgres/hourly-f1dd9-20260831135500 -n postgres --dry-run > postgres-
sir.yaml

# cat postgres-sir.yaml
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotInplaceRestore
metadata:
  name: postgres-restore-from-snapshot
  namespace: postgres
spec:
  appArchivePath: postgres-app_314bbaf6-2ce3-4065-b3c1-
ab85c7bb7c7c/snapshots/20260831135500_hourly-f1dd9-
20260831135500_36c2bd2a-9405-4424-88a7-75e6bbc5b315
  appVaultRef: ontap-s3-appvault1
  cleanUpAdditionalExecHooks: true
  cleanUpArchivedExecHooks: false
  resourceFilter: {}
  runArchivedExecHooks: true

# kubectl create -f postgres-sir.yaml
snapshotinplacerestore.protect.trident.netapp.io/postgres-restore-from-
snapshot created
```

Es sollte sichergestellt sein, dass die Objekte und PVCs der Anwendung im postgres-Namespace erstellt wurden.

```
vksbastion@vks:~/demo-vks/tp$ kubectl get all,pvc -n postgres
NAME                            READY   STATUS    RESTARTS   AGE
pod/postgres-6fcc5545c8-wrppb   1/1     Running   0          43s

NAME                       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/postgres-service   ClusterIP   10.101.142.54   <none>        5432/TCP   43s

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/postgres   1/1     1            1           43s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/postgres-6fcc5545c8   1         1         1       43s

NAME                                  STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
VOLUMEATTRIBUTESCLASS   AGE
persistentvolumeclaim/postgres-pvc   Bound    pvc-40f55cd3-ba5c-40b6-94ba-52a4e6767b33   10Gi       RWO            sc-nas
<unset>                 49s
vksbastion@vks:~/demo-vks/tp$
```

Wiederherstellung der Anwendung aus einem Snapshot in einen anderen Namensraum

Die Anwendung im Namespace postgres2, die zuvor aus dem Backup wiederhergestellt wurde, wird gelöscht.

```
vksbastion@vks:~/demo-vks/tp$ kubectl get all,pvc -n postgres2
NAME                            READY   STATUS    RESTARTS   AGE
pod/postgres-6fcc5545c8-858dt   1/1     Running   0          65m

NAME                       TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
service/postgres-service   ClusterIP   10.109.5.180   <none>        5432/TCP   65m

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/postgres   1/1     1            1           65m

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/postgres-6fcc5545c8   1         1         1       65m

NAME                                  STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
VOLUMEATTRIBUTESCLASS   AGE
persistentvolumeclaim/postgres-pvc    Bound    pvc-5893851c-c152-439f-a147-c4ae3581cc6f   10Gi       RWO            sc-nas
<unset>                 65m
vksbastion@vks:~/demo-vks/tp$ kubectl delete deployment/postgres -n postgres2
deployment.apps "postgres" deleted from postgres2 namespace
vksbastion@vks:~/demo-vks/tp$ kubectl delete service/postgres-service -n postgres2
service "postgres-service" deleted from postgres2 namespace
vksbastion@vks:~/demo-vks/tp$ kubectl delete pvc/postgres-pvc -n postgres2
persistentvolumeclaim "postgres-pvc" deleted from postgres2 namespace
vksbastion@vks:~/demo-vks/tp$ kubectl get all,pvc -n postgres2
No resources found in postgres2 namespace.
vksbastion@vks:~/demo-vks/tp$
```

Das Snapshot-Wiederherstellungsobjekt wird aus dem Snapshot erstellt, und die Namespace-Zuordnung wird bereitgestellt.

```
# tp create sr postgres-sr --snapshot postgres/hourly-f1dd9-
20260831135500 --namespace-mapping postgres:postgres2 -n postgres2 --dry
-run>postgres-sr.yaml

# cat postgres-sr.yaml
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: postgres-sr
  namespace: postgres2
spec:
  appArchivePath: postgres-app_314bbaf6-2ce3-4065-b3c1-
ab85c7bb7c7c/snapshots/20260831135500_hourly-f1dd9-
20260831135500_36c2bd2a-9405-4424-88a7-75e6bbc5b315
  appVaultRef: ontap-s3-appvault1
  cleanUpAdditionalExecHooks: true
  cleanUpArchivedExecHooks: false
  namespaceMapping:
  - destination: postgres2
    source: postgres
  resourceFilter: {}
  runArchivedExecHooks: true
  skipApplicationCreation: false

# kubectl create -f postgres-sr.yaml
snapshotrestore.protect.trident.netapp.io/postgres-sr created
```

```
vksbastion@vks:~/demo-vks/tp$ kubectl get snapshotrestore  -n postgres2
NAME          STATE       ERROR   AGE
postgres-sr   Completed           44s
```

Es sollte überprüft werden, ob die Objekte und PVCs der Anwendung im Namespace postgres2 wiederhergestellt sind.

```
vksbastion@vks:~/demo-vks/tp$ kubectl get all,pvc -n postgres2
NAME                            READY   STATUS    RESTARTS   AGE
pod/postgres-6fcc5545c8-ql8h4   1/1     Running   0          3m29s

NAME                       TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/postgres-service   ClusterIP   10.107.103.215   <none>        5432/TCP   3m29s

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/postgres   1/1     1            1           3m29s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/postgres-6fcc5545c8   1         1         1       3m29s

NAME                                  STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
VOLUMEATTRIBUTESCLASS   AGE
persistentvolumeclaim/postgres-pvc   Bound    pvc-11e41614-4706-4bc7-ab77-da57b3baf754   10Gi       RWO            sc-nas
<unset>                 3m33s
vksbastion@vks:~/demo-vks/tp$
```