vSphere Kubernetes サービスと NetApp ストレージ

NetApp container solutions

NetApp
September 01, 2026

目次

vSphere Kubernetes サービスと NetApp ストレージ

NetApp ONTAPを使用したVMware vSphere Kubernetes Serviceについて学びます

VMware vSphere Kubernetes Service（VKS）は、VMwareが提供するマネージドKubernetesサービスであり、組織がKubernetesを使用してコンテナ化されたアプリケーションをデプロイ、管理、スケールすることを可能にします。NetApp ONTAPストレージと組み合わせることで、VKSはクラウドネイティブワークロードにエンタープライズクラスの永続性、パフォーマンス、およびデータ管理を提供します。

VKSはVMware Cloud Foundation（VCF）に組み込まれており、上流のCNCF Kubernetesに準拠しているため、より広範なKubernetesエコシステムおよびツールとの互換性が確保されています。

vSphere Kubernetes Serviceの主な機能

1. マネージド**Kubernetes**クラスター**:** vSphere Kubernetes Serviceは、Kubernetesクラスターのデプロイと管理を簡素化します。クラスターの作成、アップグレード、スケーリング、監視といったタスクを自動化します。
2. **VMware** インフラストラクチャとの統合： VKS は VMware vSphere、NSX-T、およびその他の VMware ソリューションとシームレスに統合し、組織が既存の VMware インフラストラクチャを Kubernetes ワークロードに活用できるようにします。
3. マルチクラウドおよびハイブリッドクラウドのサポート： vSphere Kubernetes Serviceは、プライベートクラウド、パブリッククラウド（AWS、Azure、Google Cloudなど）、およびハイブリッドクラウド環境へのデプロイをサポートしており、組織がアプリケーションを管理する上での柔軟性を提供します。
4. 組み込みセキュリティ： VKSには、ロールベースのアクセス制御（RBAC）、ポリシー適用、およびネットワークセキュリティのためのVMware NSXとの統合などのセキュリティ機能が含まれています。
5. **Kubernetes**エコシステムのサポート： VKSは、Kubernetesの完全なアップストリームAPIをサポートしており、環境間での移植性と、サービスメッシュ用のIstio、監視用のPrometheus、その他のCNCF認定ツールなどのツールを含む、より広範なKubernetesエコシステムとの互換性を保証します。組織は、標準的なKubernetesのスキルと統合機能をVKSクラスターに直接適用できます。
6. 開発者向けツール： VKS は、CI/CD パイプライン、GitOps プラクティス、および kubectl や Helm などのツールを含む標準的な Kubernetes 開発者ワークフローをサポートしており、開発チームは独自のインターフェースを学習することなく、コンテナ化されたアプリケーションを構築・デプロイできます。
7. 拡張性と高可用性： VKSは、アプリケーションの高い可用性とパフォーマンスを維持するために、自動スケーリングやフォールトトレランスなどの機能を提供します。
8. 可観測性と監視： VKSは、標準的なKubernetes互換の監視ツールと統合し、クラスターおよびアプリケーションのパフォーマンスに関するリアルタイムの洞察を提供します。

vSphere Kubernetes Serviceのユースケース

1. アプリケーションの最新化： 組織は、レガシーアプリケーションをコンテナ化し、VKS によって管理される Kubernetes クラスターにデプロイすることで、レガシーアプリケーションを最新化できます。

2. **DevOps Enablement:** VKS は、自動化、CI/CD 統合、開発者向けツールを提供することで、DevOps ワークフローをサポートします。
3. ハイブリッドクラウド展開： 企業はVKSを使用して、オンプレミスとクラウド環境にまたがるKubernetesクラスターを展開し、一貫した運用を実現できます。
4. マイクロサービスアーキテクチャ： VKSはマイクロサービスベースのアプリケーション開発をサポートしており、チームが分散システムを構築および拡張できます。

ストレージと**vSphere Kubernetes Service**の利用開始

vSphere Kubernetes Service クラスタのインストール

vSphere Kubernetes Service（VKS）クラスタをVCF 9.1環境にインストールし、ワークロードに適したストレージユーティリティを使用してワーカーノードを構成します。

タスク概要

NFSユーティリティは、作成したクラスターのワーカーノードに自動的にインストールされます。iSCSIまたはNVMeユーティリティがインストールされたワーカーノードを作成する場合は、カスタムOVAイメージを作成し、そのイメージをワーカーノードに使用する必要があります。カスタムイメージはDockerを使用して作成でき、 `vcf` コマンドラインツールを使用して、このイメージからOVAファイルを作成できます。このOVAファイルは、vSphereのコンテンツライブラリにアップロードできます。VKSクラスタを作成する際、コンテンツライブラリからこのイメージをノードプール用に選択できます。

手順

1. VKSクラスターを作成します。

   VKSクラスタは、デフォルトのワーカーノードイメージ、または独自に作成したカスタムイメージのいずれかを使用して作成できます。デフォルトのワーカーノードイメージにはNFSユーティリティがプリインストールされていますが、カスタムイメージにはiSCSIおよびNVMeユーティリティを含めることができます。以下のオプションが利用可能です：

   - **NAS**のみ（デフォルト）： NFSユーティリティがプリインストールされたデフォルトのワーカーノードイメージを使用して VKS クラスタを作成します。
   - **SAN** 有効（カスタム イメージ）： iSCSI および NVMe ユーティリティを含むカスタム Docker イメージを作成し、 `vcf` コマンドラインツールを使用して OVA ファイルを生成し、OVA を vSphere コンテンツ ライブラリにアップロードして、そのカスタム イメージをノード プールに選択して VKS クラスターを作成します。

     ▼ デフォルトのワーカーノードイメージを使用した**NAS**専用の**vSphere Kubernetes Service**クラスタを作成する手順を表示する

     デフォルトのワーカーノードイメージを使用してVKSクラスタを作成します。画面の指示に従って、クラスター名、ノードプール構成、その他の設定を指定してください。

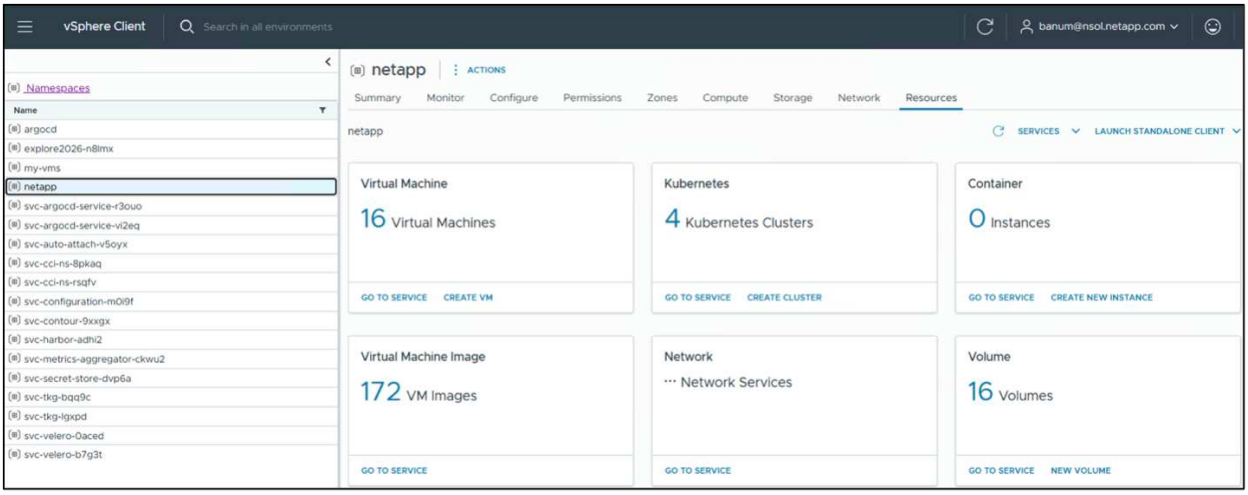
     vSphere Client から、vSphere Kubernetes クラスターを作成する名前空間に移動します。その名前空間の「リソース」タブで、Kubernetes セクションの「クラスターの作成」をクリックするか、「サービスへ移動」をクリックしてから「クラスターの作成」をクリックします。

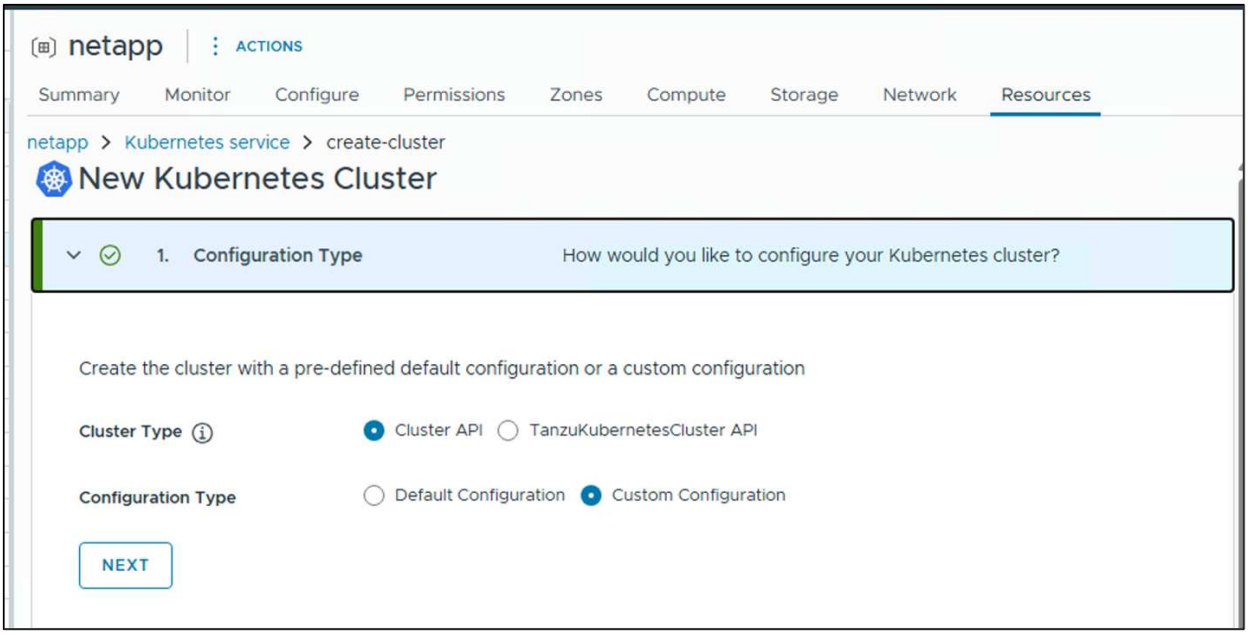
     クラスターの詳細をフォームに入力してください。

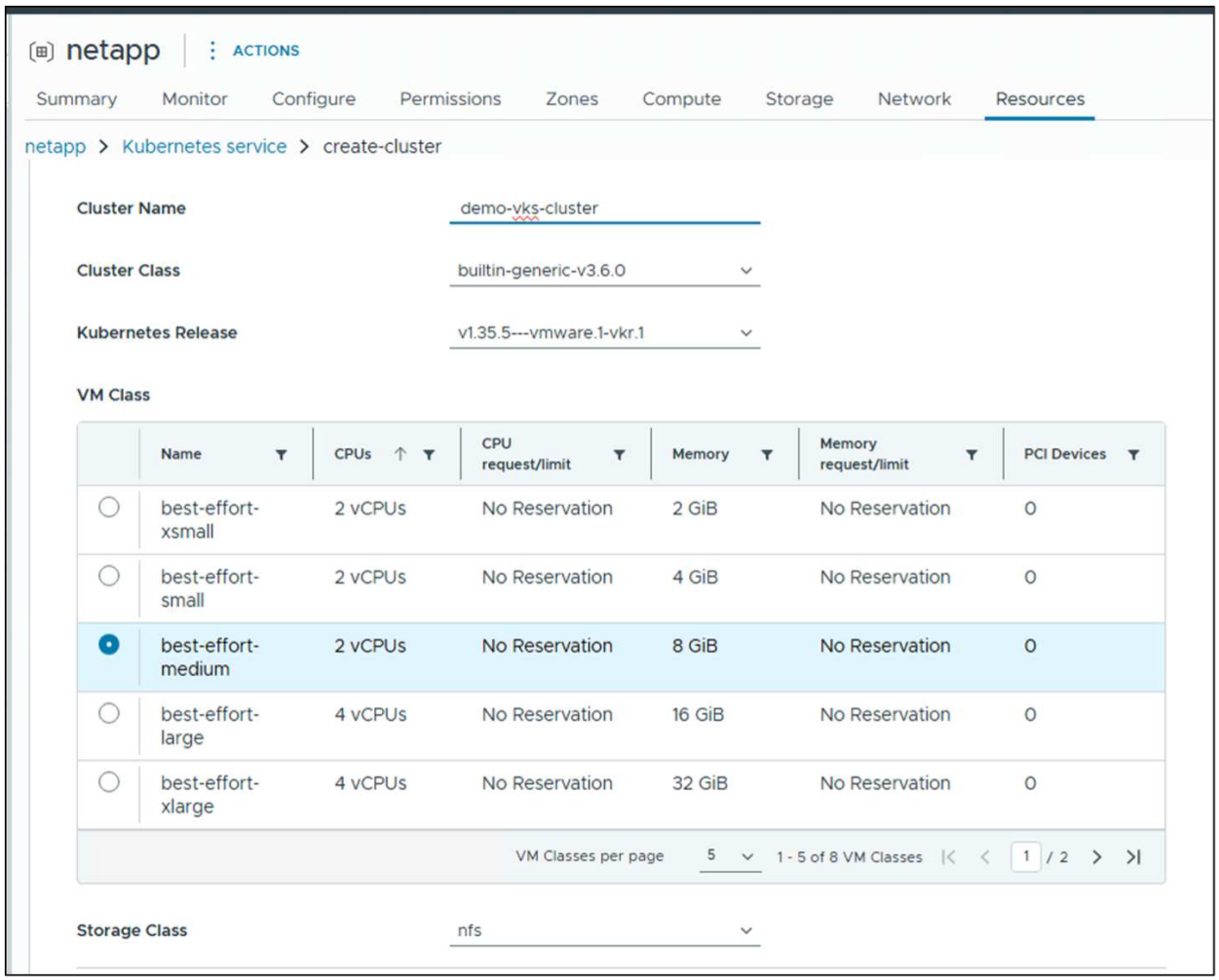
     - セクション1の「構成タイプ」で、「カスタム」を選択します。

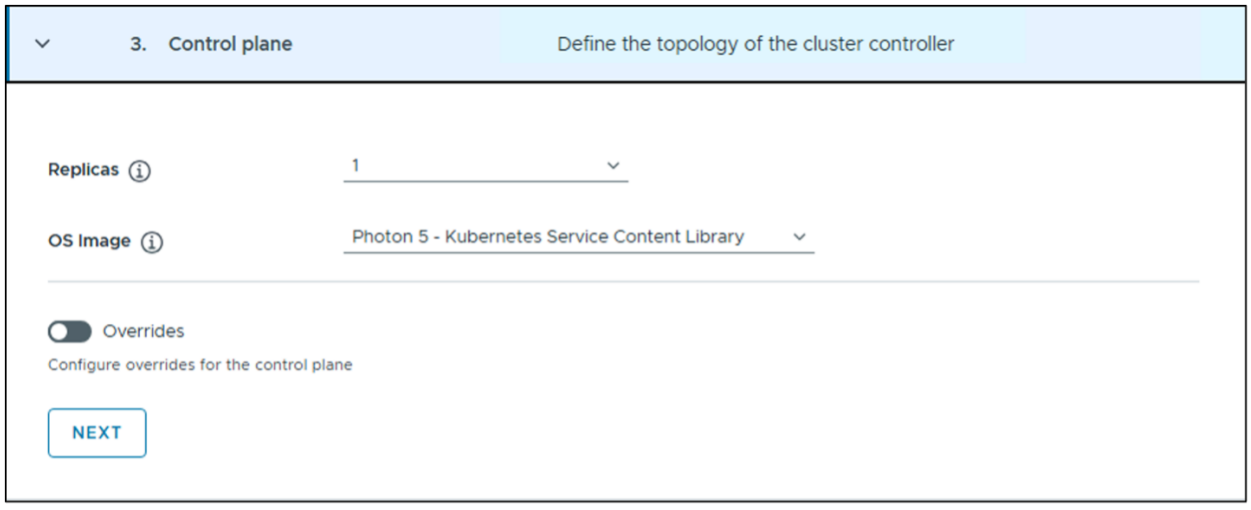
- セクション2の「一般設定」で、クラスターの名前を指定し、その他の設定はすべてデフォルト値のままにしてください。
- セクション3のデフォルト設定をすべて受け入れてください。
- セクション 4「ノード プール」で、ノード プールの横にある省略記号（…）をクリックし、「編集」をクリックします。レプリカ数を 3 に増やし、OS イメージには最新の Ubuntu バージョンを選択します。「次へ」をクリックし、次に「確認と確定」をクリックします。

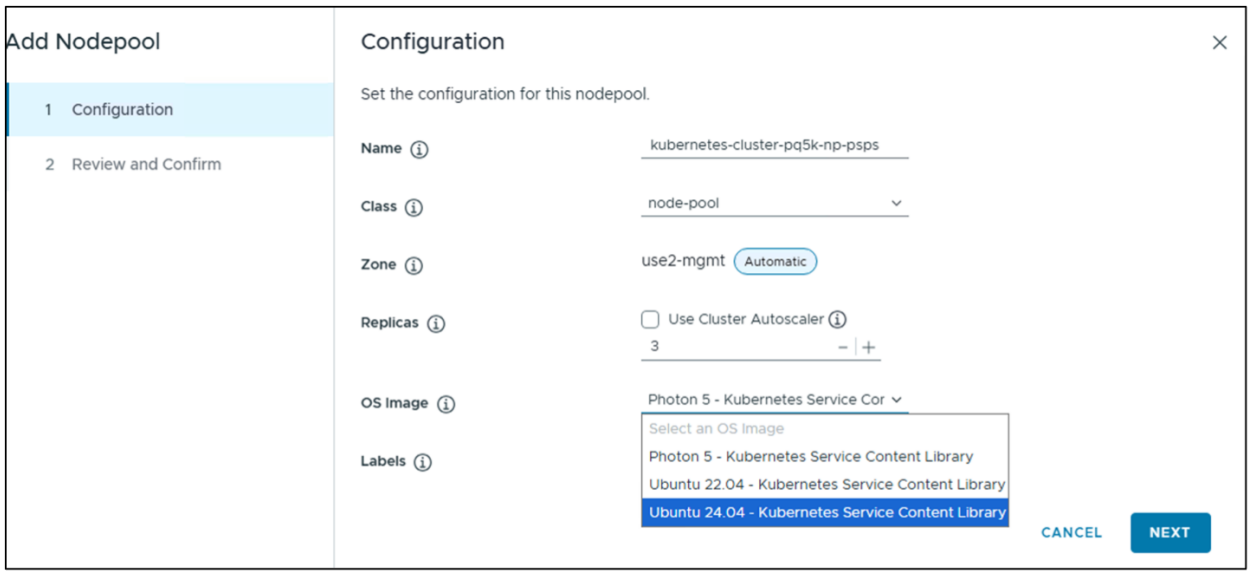
クラスターの詳細を確認したら、「完了」をクリックします。vSphere Kubernetes クラスターは数分以内に作成されます。

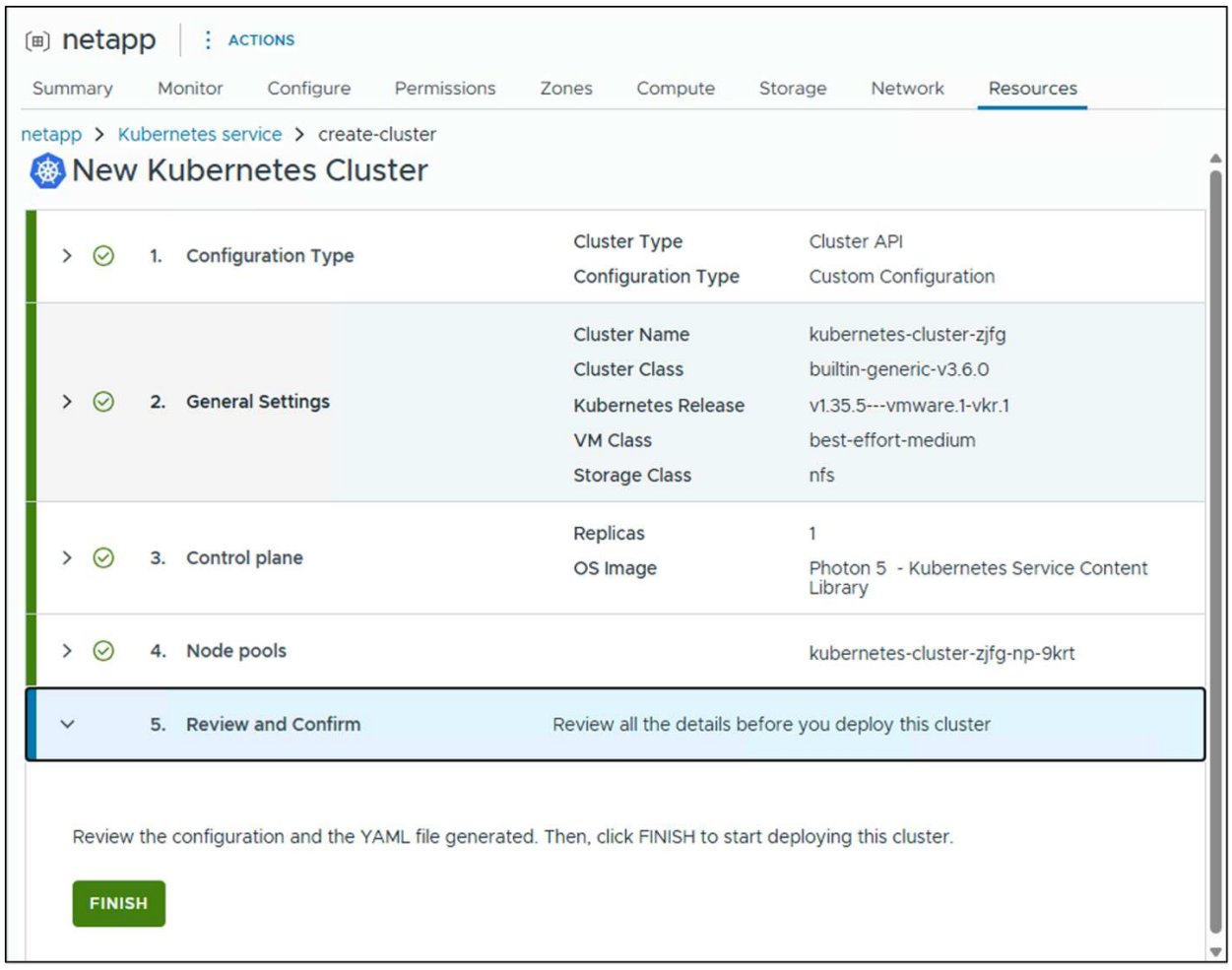


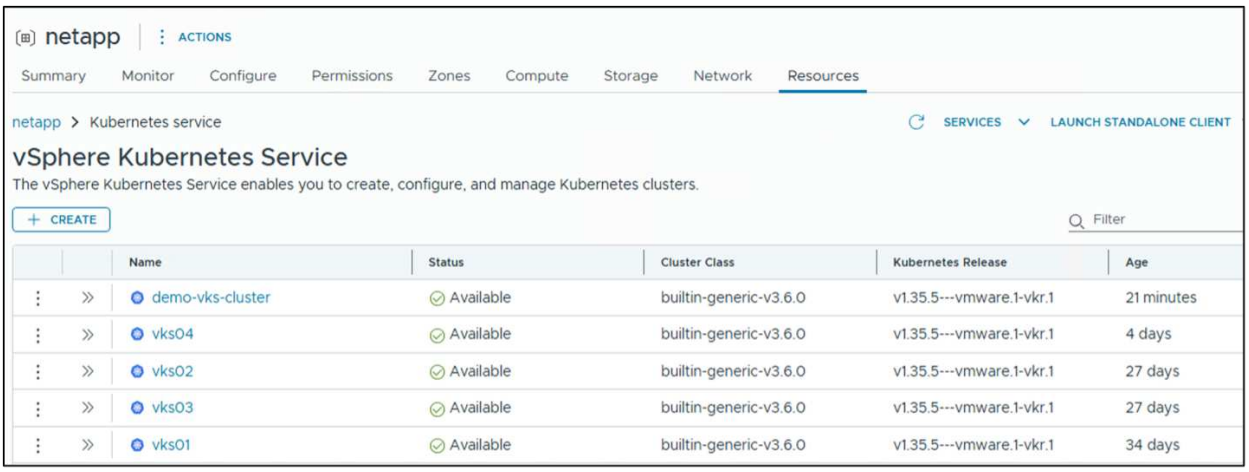



netapp
ACTIONS
Summary
Monitor
Configure
Permissions
Zones
Compute
Storage
Network
Resources
netapp > Kubernetes service > create-cluster
Cluster Name
demo-vks-cluster
Cluster Class
builtin-generic-v3.6.0
Kubernetes Release
v1.35.5---vmware.1-vkr.1
VM Class
Name
CPUs
CPU request/limit
Memory
Memory request/limit
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



▼ **SAN対応ワーカーノード用のカスタムOVAイメージを作成する手順を表示する**

必要なユーティリティがインストールされた Docker イメージを作成します。以下の例は、iSCSI およびマルチパスユーティリティがインストールされた Ubuntu 24.04 ベースの Docker イメージを作成する方法を示しています。

```
#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"]
```

以下の方法でDockerイメージをビルドします：

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

Docker レジストリが利用できない場合は、次のコマンドを実行してローカルレジストリを起動し、そこにイメージをプッシュしてください：

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

イメージにタグを付けてプッシュします。

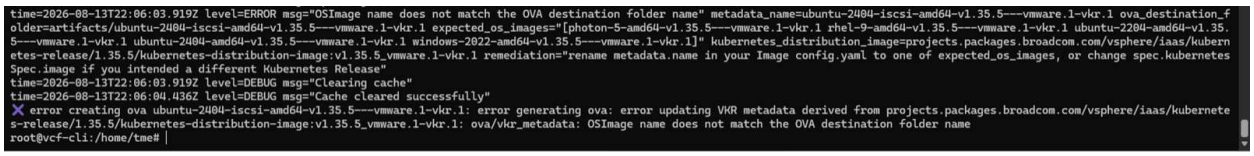
```
docker tag ubuntu-2404:san localhost:5000/ubuntu-2404:san
docker push localhost:5000/ubuntu-2404:san
```

イメージ設定ファイルを作成し（`ubuntu2404vkr135-san.yml`として保存）、イメージを参照します：

```
#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
```

メタデータ名は、事前に承認されたリストの中から選択する必要があります。そうでない場合、OVAファイルを作成しようとしたときに、以下に示すようなエラーが表示されます。事前承認リストは、 `kr bake`コマンドのVCF CLIヘルプで確認できます。



VCF CLIを使用してOVAファイルを作成します。

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

SCP を使用して OVA ファイルを、vSphere コンテンツライブラリにアップロードできるマシンに転送します。

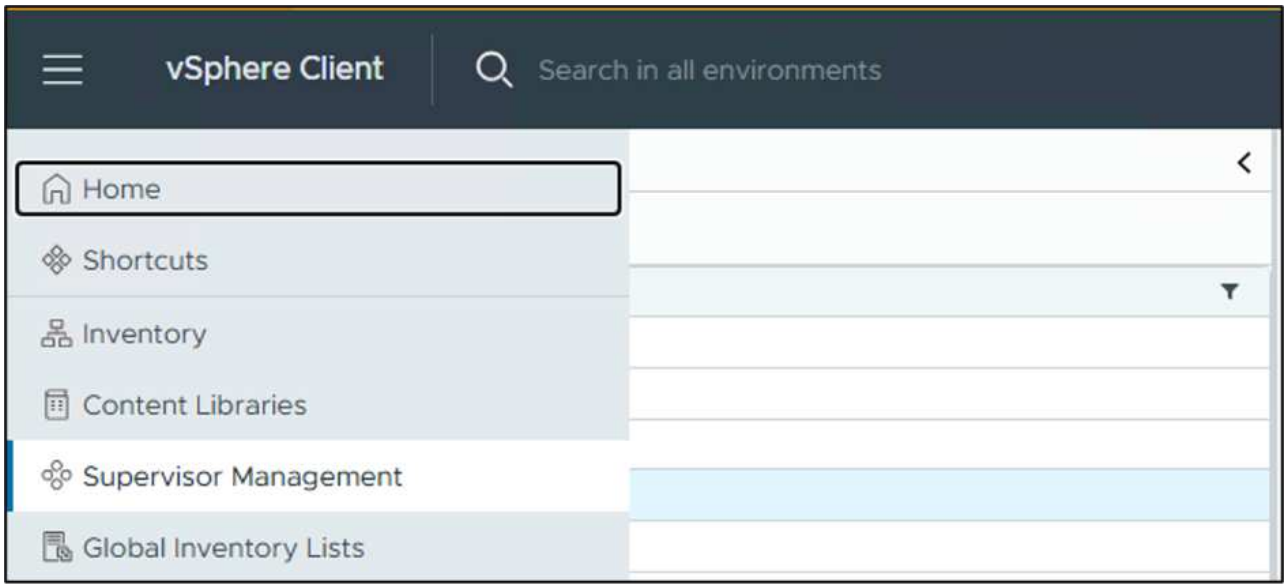
2. クラスターのインストール後、vCenter でクラスターを見つけて、kubeconfig ファイルをダウンロードします。

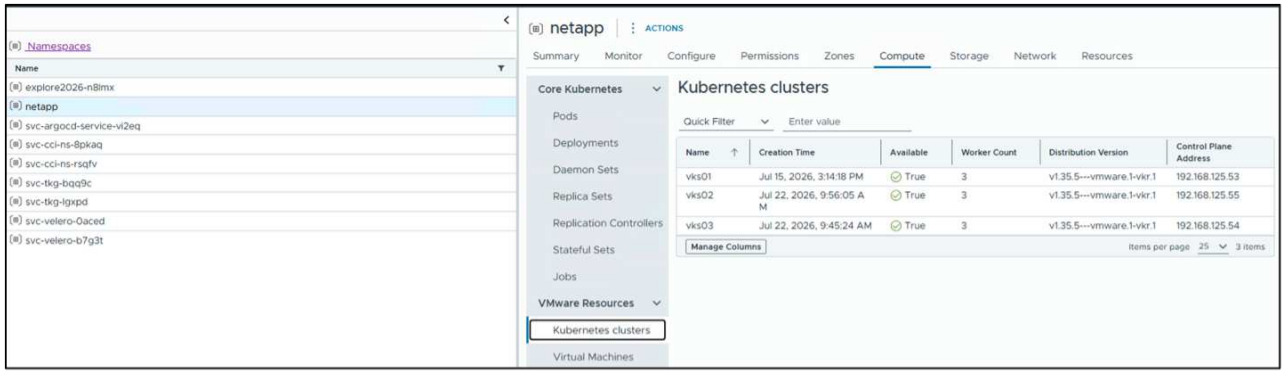
    a. メインメニューから **Supervisor Management** をクリックし、クラスターが作成された名前空間を選

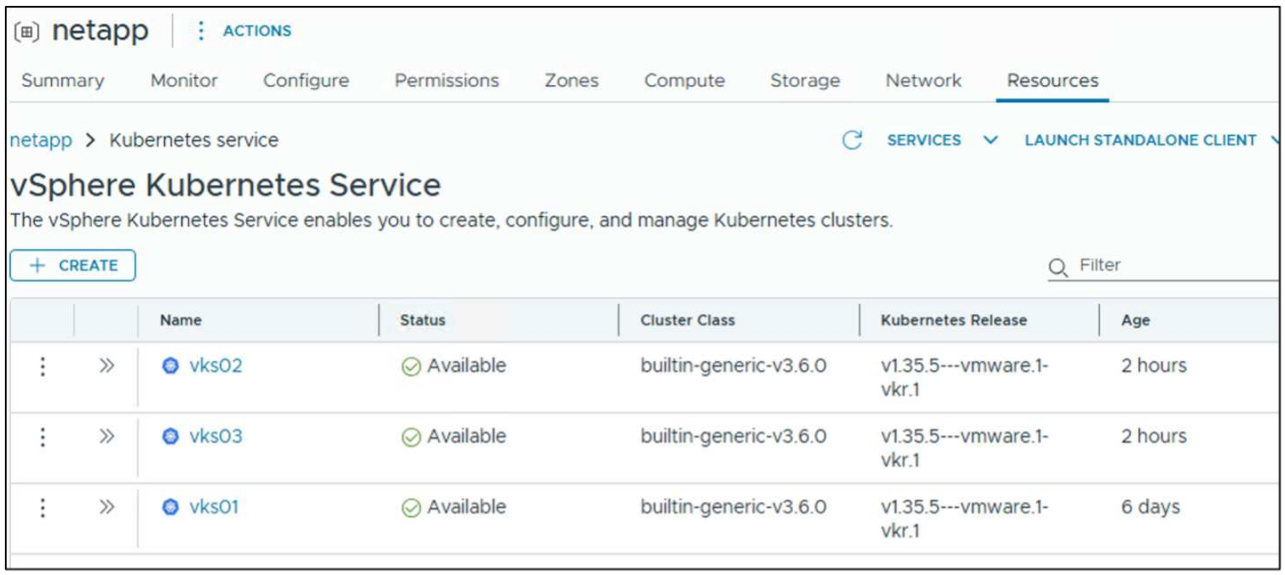
択します。

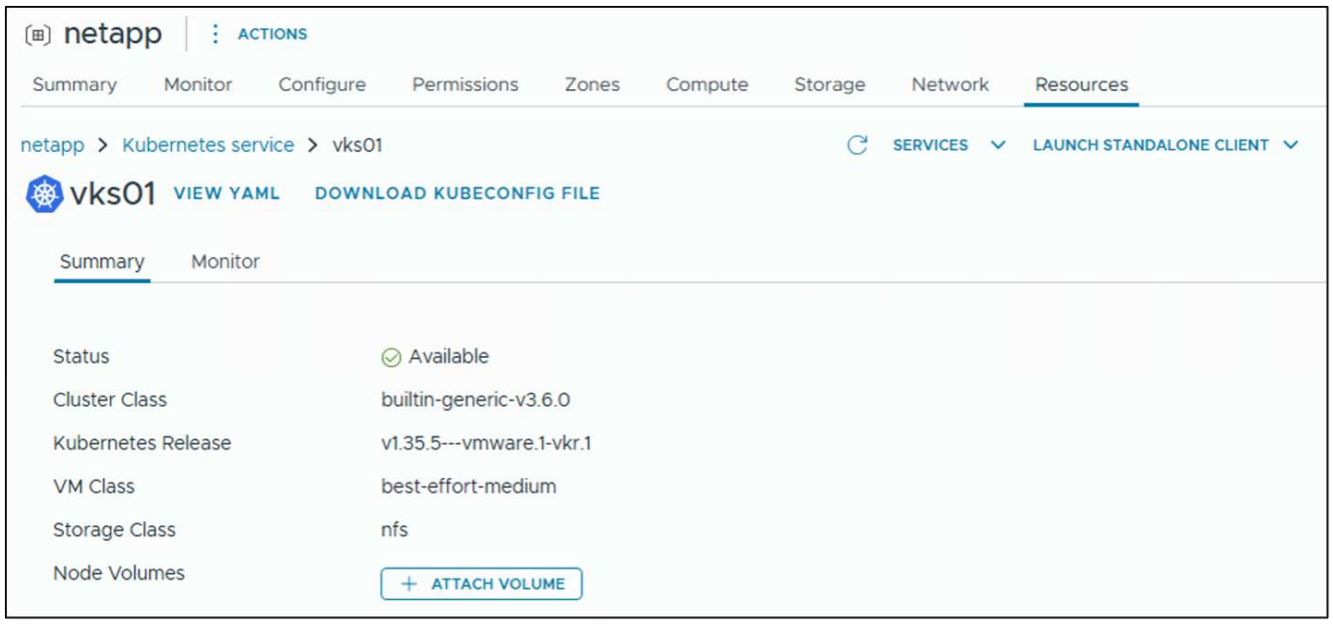
b. 「コンピューティング」タブを選択し、次に「Kubernetes クラスター」を選択します。名前空間内のすべてのクラスターが表示されます。

c. 「リソース」タブに移動し、必要な Kubernetes クラスターを選択して、その kubeconfig ファイルをダウンロードします。









3. 踏み台ホストから、kubeconfig ファイルを使用してクラスターに接続し、CLI から管理します。

```
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
```

ビデオデモ

以下のビデオでは、vSphere Kubernetes クラスターを作成する2つの方法を説明します。

スーパーバイザークラスタ上に vSphere Kubernetes クラスタを作成する

VCF Automationを使用してvSphere Kubernetesクラスターを作成する

vSphere Kubernetes Service向けのNetAppストレージについて説明します

NetAppをvSphere Kubernetes Service （VKS）のストレージとして使用することは、NetAppの堅牢なストレージソリューションを活用してKubernetesワークロードのパフォーマンス、拡張性、信頼性を向上させる強力な組み合わせです。NetApp Tridentは、Kubernetes向けのオープンソースの動的ストレージプロビジョナーであり、vSphere Kubernetes Serviceのワークロードとストレージを統合するために使用されます。

VKSでNetAppストレージを使用する主なメリット

1. 動的ストレージプロビジョニング： NetApp Trident はストレージボリュームの動的なプロビジョニングを可能にし、Kubernetes ポッドが要件に基づいてその場でストレージを要求できるようにします。
2. 永続ストレージ： NetApp ソリューションは永続的なストレージ機能を提供し、ポッドの再起動や障害発生時にもデータの耐久性と可用性を確保します。

3. ハイパフォーマンス： NetAppのストレージソリューション（ONTAPなど）は、低レイテンシのハイパフォーマンスストレージを提供しており、Kubernetes上で動作するI/O負荷の高いアプリケーションに最適です。
4. 拡張性： NetAppストレージシステムは、増大するKubernetesワークロードの需要に合わせて拡張でき、大規模な展開をサポートできます。
5. データ管理： NetApp は、スナップショット、クローニング、データレプリケーションなどの高度なデータ管理機能を提供しており、バックアップ、ディザスタリカバリ、開発ワークフローに役立ちます。
6. マルチクラウドおよびハイブリッドクラウドのサポート： NetAppのストレージソリューションは、オンプレミス、パブリッククラウド、およびハイブリッドクラウド環境に対応し、ストレージ管理における柔軟性と一貫性を提供します。

NetApp ONTAP統合の概要

NetApp ONTAPは、データ重複排除、圧縮、暗号化などの機能を備えたエンタープライズクラスのストレージを提供します。NetApp Tridentを使用して、NetAppストレージをKubernetesと統合します。NetApp Tridentは、オンプレミスのONTAP、AWSのFSx for NetApp ONTAP、AzureのAzure NetApp Files、Google CloudのGoogle Cloud NetApp Volumesなど、さまざまなNetAppストレージプラットフォームをサポートしています。

Trident Protect を使用して、Kubernetes ワークロードのデータ保護とディザスタリカバリを処理します。Trident Protect を使用すると、スナップショットやクローンを作成し、異なるストレージバックエンド間でデータをレプリケートできるため、障害発生時にもデータが安全に保護され、復旧可能になります。

Trident の主な機能

CSI準拠 最新のKubernetesストレージ機能および標準との互換性を確保します。宣言型構成 ユーザーがYAMLファイルでストレージ要件を定義できるようにし、その後Tridentによって自動的に管理されます。ドライバーTridentはONTAPがサポートするNFS、CIFS/SMB、iSCSI、FC、NVMe/TCPプロトコルのドライバーをサポートします。ハイブリッドおよびマルチクラウドサポート Tridentはオンプレミスの ONTAPストレージおよびハイパースケーラーにおけるマネージドストレージサービス（具体的にはAWSのFSx for NetApp ONTAP、AzureのAzure NetApp Files、Google CloudのGoogle Cloud NetApp Volumes）のドライバーをサポートします。高度なデータ管理 ONTAPのスナップショットおよびクローニング機能を活用し、効率的なデータ保護とテスト環境および開発環境の迅速なプロビジョニングを実現します。

Trident の詳細については、"Tridentドキュメント"を参照してください。

Trident Protect

Trident Protect は Trident とは別にインストールされます。デプロイ前に、Kubernetes プラットフォーム、ストレージバックエンド、スナップショットコンポーネント、およびオブジェクトストレージが"Trident Protectの要件"を満たしていることを確認してください。

Trident Protect の主な機能

自動バックアップ Trident Protect は、自動化されたポリシーベースの Kubernetes アプリケーションのバックアップを実現し、手動による介入なしに定期的にバックアップが行われるようにします。

スナップショット管理 ONTAPのスナップショット機能を活用して、コンテナアプリケーションおよびVMのポイントインタイムコピーを頻繁に作成し、信頼性の高い復旧メカニズムを提供します。

バックアップリポジトリの暗号化 Trident Protect は、Restic および Kopia バックアップリポジトリのパスワードベースの暗号化をサポートしています。ストレージ層とネットワーク層で、永続ボリューム暗号化と転送

中暗号化をそれぞれ個別に設定してください。

コンプライアンスと監査 組織がコンプライアンス要件を満たし、データ保護慣行の定期的な監査を実施するのに役立つツールと機能を提供します。

ディザスタリカバリ ONTAPのレプリケーション機能と統合することで、堅牢なディザスタリカバリソリューションを提供し、壊滅的な事態が発生した場合でも事業継続性を確保します。

Trident Protect の詳細については、"Trident Protect ドキュメント"を参照してください。

NetApp ONTAPのサポートされているハードウェアモデルとプロトコル

NetApp ONTAP ソフトウェアは、さまざまなハードウェアモデル上で動作し、それぞれが異なる性能および容量要件に対応するように設計されています。Trident は、All Flash FAS （AFF）、All SAN Array （ASA）、および ASA R2 システムを含むさまざまな NetApp ONTAP システムをサポートする堅牢な機能を拡張します。このサポートにより、組織はコンテナ化された環境においてハイパフォーマンス ストレージ ソリューションの潜在能力を最大限に活用できるようになります。

オールフラッシュ **FAS** （**AFF**）システム AFF システムは優れたパフォーマンスと低レイテンシを実現し、高負荷アプリケーションに最適です。

All SAN Array （**ASA**） システム ASA システムは専用の SAN 環境向けに設計されており、堅牢なパフォーマンスと信頼性を提供します。

ASA R2 Systems ASA R2システムは、SAN環境向けに強化された機能と性能を提供します。

AFX NetApp AFX は、エンタープライズ AI ワークロード向けに設計された、目的に即した分散型オールフラッシュ ストレージ アーキテクチャです。ハイパフォーマンス NAS を提供し、コンピューティングと容量を独立してスケーリングできます。NetApp AFX ストレージシステムは、ストレージ レイヤの実装において、他の ONTAP ベースのシステム（ASA、AFF、FAS）とは異なります。AFX は高性能と拡張性を実現する分散ファイルシステムを使用しますが、他の ONTAP ベースのシステムは従来型のストレージ アーキテクチャを使用します。

1. AFF でサポートされているプロトコル：NFS、CIFS/SMB、iSCSI、FC、NVMe/TCP
2. ASA でサポートされているプロトコル：iSCSI、FC、NVMe/TCP
3. ASA R2 でサポートされているプロトコル：iSCSI、FC、NVMe/TCP
4. AFX でサポートされているプロトコル：Trident 25.10 以降、NFS プロトコルのみがサポートされており、SMB プロトコルはサポートされていません。

ONTAP ストレージ向け **Trident** ドライバー

Tridentには、バックエンドストレージにボリュームをプロビジョニングできる以下のCSIドライバが含まれています。

サポートされているハードウェアモデルにおける**NAS**プロトコルの場合

1. ontap-nas：1 つの PVC が 1 つの PV にマッピングされ、これは専用の FlexVol volume です。
2. ontap-nas-economy: プロビジョニングされた各 PV は qtree であり、FlexVol volume あたりの qtree 数は設定可能です（デフォルトは 200、50 ～ 300 の間で設定可能）。

   1つのPVCは1つのPVに対応し、これは共有FlexVol volume上のqtreeです。

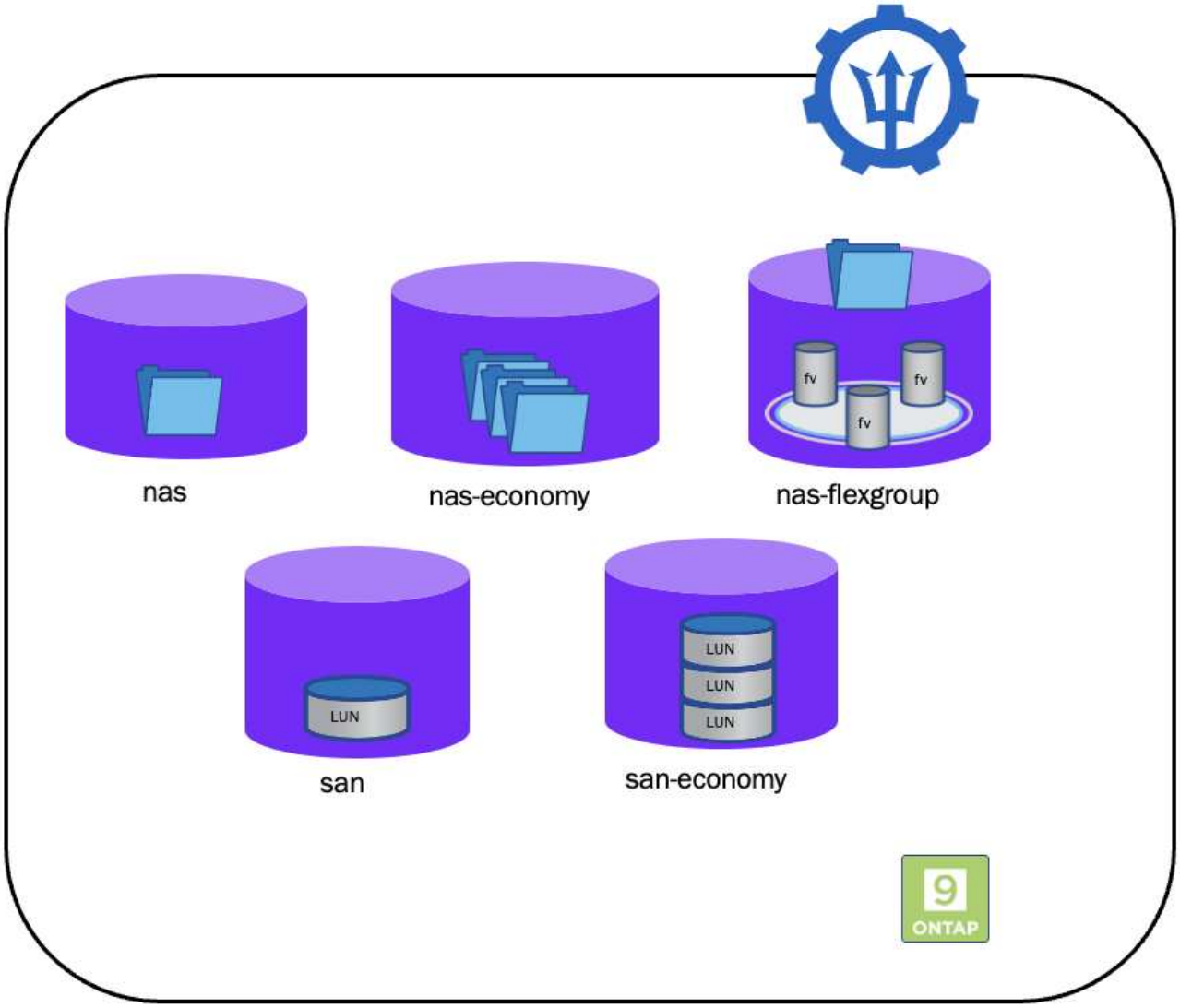
仮想マシンのすべてのディスクをqtreeとして、単一のボリューム内に配置できます。ただし、スナップショット、クローン、レプリケーション機能はTridentではサポートされていないことにご注意ください。これらの機能がストレージ管理者によって管理される場合、PVの粒度には対応していません。

1. ontap-nas-flexgroup: プロビジョニングされた各 PV は完全な ONTAP FlexGroup であり、SVM に割り当てられたすべてのアグリゲートが使用されます。

サポートされているハードウェアモデルにおける**SAN**プロトコルの場合

1. ontap-san: 1 つの PVC が 1 つの PV にマッピングされます。これは専用の FlexVol volume 上の LUN です。
2. ontap-san-economy: 1 つの PVC が 1 つの PV にマッピングされ、これは共有 FlexVol volume 上の LUN です。共有 FlexVol volume には複数の LUN を設定できます（デフォルトは 100、FlexVol volume あたり 50～200 LUN/PV で設定可能）。

仮想マシンのすべてのディスクをLUNとして、単一のボリューム内に配置できます。ただし、このドライバーはスナップショットとクローンはサポートしていますが、レプリケーションはサポートしていません。ストレージ管理者がレプリケーションを管理する場合、PV単位の粒度には対応していません。



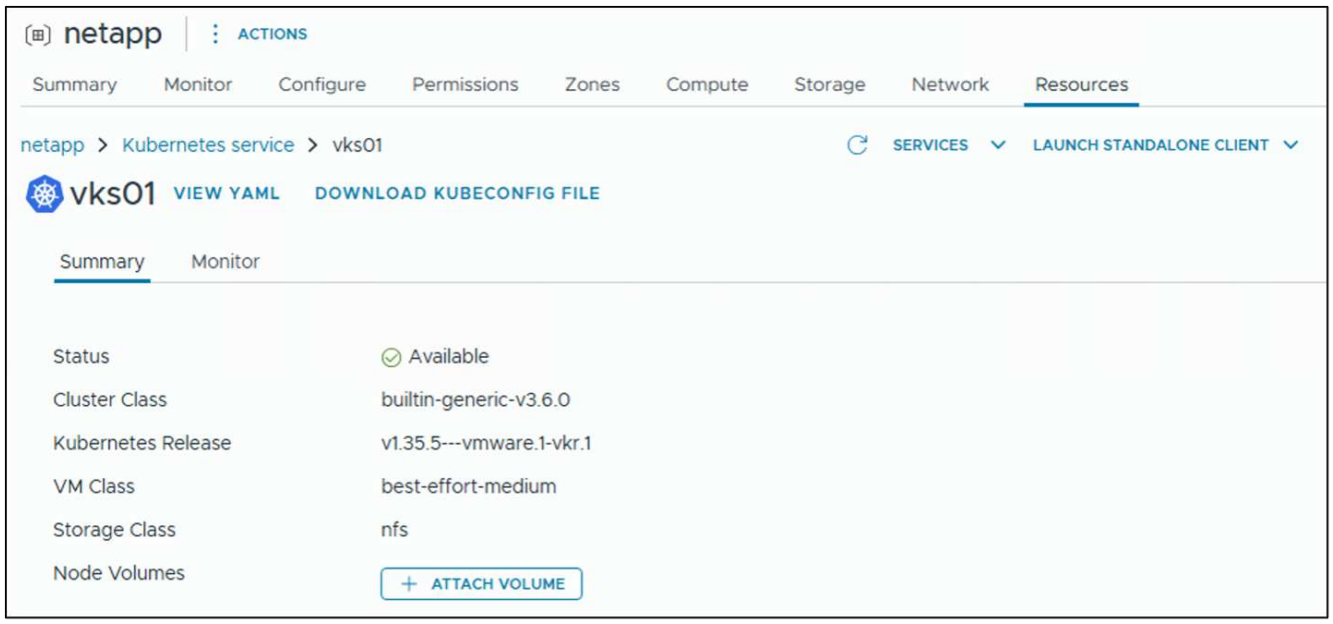
Trident ドライバを通じて公開されている機能と、ストレージ管理者が適用する必要がある機能の完全なリストについては、Trident ドキュメントの "ONTAP バックエンドドライバ" セクションを参照してください。

VKSクラスターへのNetApp Tridentのインストール

NetApp Trident を動的ストレージ プロビジョニング ツールとして vSphere Kubernetes Service クラスターにインストールし、NetApp ONTAP ストレージを Kubernetes ワークロードと統合します。

開始する前に

- ONTAPクラスターが構成され、VMware Kubernetes環境に接続されていることを確認してください。
- ワークロードでiSCSIまたはNVMeプロトコルをストレージに使用する場合は、VKSクラスタにiSCSIまたはNVMeツールがインストールされていることを確認してください。
- 必ず `kubectl`を、kubeconfigファイルを使用してVKSクラスターに接続できるマシンにインストールしてください。



手順

1. Helm チャートを使用して Trident Operator をインストールするには、Trident ドキュメントの手順に従ってください：

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

   Helm インストールを実行する前に、 `trident` 名前空間を作成してラベルを付けてください。Trident は特権ワークロードを実行するため、 `enforce=privileged` ラベルが必要です。これがないと、Kubernetes ポッドセキュリティアドミッションコントローラーが Trident ポッドの起動をブロックします。

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

   ▼ 例を表示

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

   デバッグログを有効にするには、以下を追加します `--set tridentDebug=true` ：

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

   Trident Operatorを使用して、Tridentを手動でインストールすることもできます。Tridentドキュメントの手順を確認してください："Trident オペレーターを手動でデプロイする"

`trident`名前空間に必要なPodセキュリティラベルが付いていることを確認してから、Tridentを手動でインストールしてください。

2. すべての Trident ポッドがクラスター内で実行されていることを確認してください。

▼ 例を表示

```
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$
```

Trident ポッドが `PodSecurity`アドミッションエラーで起動しない場合、インストール前にポッドのセキュリティラベルが `trident`ネームスペースに適用されていない可能性があります。 `--overwrite`を使用して再適用し、ポッドが起動することを確認してください。

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

ストレージバックエンドと環境用の**StorageClass**設定ファイルを準備する

1. 必要な接続の詳細とストレージパラメータを定義して、ONTAP の Trident バックエンドを設定します。
2. Kubernetes で、NetApp ストレージバックエンドにマッピングするストレージクラスを定義します。これらのクラスは、パフォーマンス特性やレプリケーションポリシーなどのパラメータを指定します。

   ワークロードの要件に応じて、以下のプロトコルに対応する Trident バックエンドとストレージクラスを定義します：

   - NAS（ontap-nas ドライバ）
   - NAS Economy（ontap-nas-economy ドライバ）
   - iSCSI、FC、またはNVMeプロトコル用のSAN（ontap-sanドライバ）
   - iSCSI 用 SAN Economy (ontap-san-economy ドライバ)

NAS

TridentBackendConfigとStorageClassを作成して、ONTAP NASでNFSベースの永続ストレージプロビジョニングを有効にします。バックエンド構成には、Kubernetes Secret に保存されている認証情報が含まれており、ONTAP SVM と管理 LIF を参照します。

ユーザー名とパスワード認証を使用したバックエンドシークレットとバックエンド設定ファイルのサンプル（ `tbc-nas.yaml` として保存）：

```
# 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
```

クライアント証明書認証を使用するバックエンドシークレットとバックエンド構成ファイルのサンプル（ `tbc-nas.yaml`として保存）：

```
# 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定義（`sc-nas.yaml`として保存）：

`mountOptions`は、NFSボリュームの追加マウントオプションを指定するオプションパラメータです。`nconnect`オプションを使用すると、単一のNFSマウントに対して複数のTCP接続を並列で確立できるため、高スループットのワークロードのパフォーマンスを向上させることができます。デフォルト値は1ですが、ONTAPシステムで許可される最大値まで増やすことができます。

ONTAPは、nconnect対応クライアント上の単一のNFSマウントに対して、最大16個の接続をサポートします。Tridentはマウントオプションを直接Kubernetesワーカーノードに渡すため、ワーカーノードのNFSクライアントとONTAPの両方でサポートされている値を選択してください。次の例では、4つの接続を使用します。

```
# 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
```

qtrees を使用した NFS ベースの永続的なストレージ プロビジョニングを有効にするために、ONTAP NAS Economy ドライバー用の TridentBackendConfig と StorageClass を作成します。バックエンド構成には、Kubernetes Secret に保存された認証情報が含まれており、お客様の ONTAP SVM と管理 LIF を参照します。

ユーザー名とパスワード認証を使用したバックエンドシークレットとバックエンド設定ファイルのサンプル（ `tbc-nas-economy.yaml` として保存）：

```
# 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定義（ `sc-nas-economy.yaml`として保存）：

```
# 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
```

iSCSIを使用したONTAP SAN用のTridentBackendConfigとStorageClassを作成して、iSCSIベースのブロック ストレージ プロビジョニングを有効にします。バックエンド構成ではontap-sanドライバーを使用し、Kubernetes Secretに保存されている資格情報を含めます。省略した場合、 `sanType`パラメーターはデフォルトでiSCSIプロトコルになります。

ユーザー名とパスワード認証を使用するバックエンドシークレットとバックエンド設定ファイル（保存先： `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
```

クライアント証明書認証を使用するバックエンドシークレットとバックエンド構成ファイル（ `tbc-iscsi.yaml`として保存）：

```
# 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定義（`sc-iscsi.yaml`として保存）：

```
# 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
```

NVMe ベースのブロック ストレージ プロビジョニングを有効にするために、ONTAP SAN with NVMe 用の TridentBackendConfig と StorageClass を作成します。バックエンド構成では、ontap-san ドライバを使用し、Kubernetes Secret に格納された認証情報が含まれます。 `sanType` パラメータは `nvme` に設定

する必要があります。

ユーザー名とパスワード認証を使用するバックエンドシークレットとバックエンド設定ファイル（保存先： `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
```

クライアント証明書認証を使用するバックエンドシークレットとバックエンド構成ファイル（ `tbc-nvme.yaml`として保存）：

```
# 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定義（`sc-nvme.yaml`として保存）：

```
# 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
```

SANドライバおよびNASドライバでサポートされているアクセスモードとボリュームモードに関する情報、およびYAMLファイルの追加サンプルについては、Tridentのドキュメントを参照してください。

- "NASドライバを使用したサンプル"
- "SANドライバを使用したサンプル"

VolumeSnapshotClass 設定ファイルを作成する

VolumeSnapshotClass の定義を作成します。この構成により、永続ボリュームに対するスナップショットベースの操作が可能になります。

VolumeSnapshotClass定義（`snapshot-class.yaml`として保存）：

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

構成ファイルをクラスターに適用します

前の手順で作成した構成ファイルを `kubectl`を使用してvSphere Kubernetes Serviceクラスタに適用します。これにより、クラスターに必要なシークレット、バックエンド構成、ストレージクラス、およびスナップショットクラスが作成されます。

手順

1. 設定したプロトコルごとにTridentBackendConfigとStorageClassファイルを適用します。

```
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. リソースが正常に作成されたことを確認してください。

   TridentBackendConfigオブジェクトを確認：

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

   StorageClassオブジェクトを確認します：

```
kubectl get storageclass
```

   VolumeSnapshotClassを確認：

```
kubectl get volumesnapshotclass
```

デフォルトの **Trident** ストレージクラスとスナップショットクラスを設定する

vSphere Kubernetes Serviceクラスターで、TridentのStorageClassおよびVolumeSnapshotClassをデフォルトとして設定します。

手順

1. デフォルトのTrident StorageClassを設定します。

   Trident-backed StorageClassをクラスターのデフォルトとして設定すると、ストレージクラスが指定されていない場合にPersistentVolumeClaimsが自動的にそれを使用します。

   デフォルトとして設定されている StorageClass が1つだけであることを確認してください。別の StorageClass がすでにデフォルトに設定されている場合は、そのアノテーションを `false`に設定してください。

   CLIから：

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

2. デフォルトのTrident VolumeSnapshotClassを設定します。

   Tridentを基盤とするVolumeSnapshotClassをクラスターのデフォルトとして設定することで、永続ボリュームに対してスナップショットベースの操作が有効になります。これにより、スナップショットクラスが指定されていない場合に、VolumeSnapshotsが自動的にTrident CSIドライバーを使用するようになります。

   デフォルトとして設定されている VolumeSnapshotClass が1つだけであることを確認してください。別の VolumeSnapshotClass がすでにデフォルトに設定されている場合は、その注釈を `false`に設定してください。

   CLIから：

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

ワークロードのストレージを要求するには、**Kubernetes Persistent Volume Claims** を使用します。

Tridentは、バックエンドのNetAppストレージから必要なボリュームを自動的にプロビジョニングします。VKSクラスタでPVCを使用してワークロードをデプロイする例については、"永続ストレージを使用してワークロードをデプロイする"を参照してください。

コンテナアプリケーションを **NetApp** 永続ストレージでデプロイする

NetApp ONTAP 永続ストレージを使用して、vSphere Kubernetes Service クラスターにコンテナ化されたアプリケーションをデプロイします。次の例では、NAS（ontap-nas）または SAN iSCSI（ontap-san）ストレージのいずれかを使用して PostgreSQL データベースをデプロイする方法を示します。

以下のYAML設定では、マニフェスト内でPOSTGRES_USER / POSTGRES_PASSWORDを直接設定します。本番環境では、データベース認証情報などの機密情報を管理するためにKubernetes Secretsを使用することをお勧めします。

NASストレージ（**ontap-nas**ドライバ）を使用して**PostgreSQL**をデプロイする

ontap-nasドライバによってプロビジョニングされたNFS永続ボリュームをバックエンドとするPostgreSQLコンテナアプリケーションをデプロイできます。次の例は、PersistentVolumeClaim（PVC）とPostgreSQLのデプロイメントを作成する方法を示しています。

```
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
```

SANストレージ（**ontap-san iSCSI**ドライバ）を使用して**PostgreSQL**をデプロイする

以下のYAMLを使用して、ontap-sanドライバによってプロビジョニングされたiSCSI永続ボリュームをバックエンドとするPostgreSQLコンテナアプリケーションをデプロイします。`Recreate`デプロイメント戦略では、新しいポッドが開始される前にポッドが完全に終了することが保証されます。これは、ReadWriteOnce iSCSIボリュームに必要であり、ポッドの再起動後のデータ破損を防止します。この構成は、ポッドの削除と再作成をまたいだデータの永続性をサポートし、Tridentボリュームのスナップショット、バックアップおよびリストア操作と互換性があります。

```
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
```

データベースのデプロイメントを検証する

アプリケーションをデプロイした後、ポッドが実行状態になっていること、およびデータベースが正常に動作していることを確認してください。以下のコマンドを使用して、ポッド、PVC、およびサービスの状態を確認します。ポッドが稼働していること、およびPVCが適切なボリュームにバインドされていることを確認してください。

```
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:~$
```

使用するプロトコルに応じて、バックエンドストレージはボリューム、qtree、またはLUNです。ONTAP システムでプロビジョニングされたストレージを確認するには、PersistentVolume（PV）を記述して内部ボリューム名を取得し、それを ONTAP CLI コマンドで使用してストレージの詳細を取得します。次のコマンドを使用して PV を記述し、内部ボリューム名を取得します：

```
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>
```

図 **1.** サンプルの提示

```
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>
```

出力から内部ボリューム名をコピーし、次のコマンドを実行して ONTAP CLI からストレージの詳細を取得します。

NASボリュームの場合：

```
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
```

SAN LUN の場合は、次のコマンドを実行して ONTAP CLI から LUN の詳細を取得します：

```
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::>
```

NASまたはNAS Economyストレージクラスで `nconnect`マウントオプションを使用した場合、ワーカーノードからストレージサーバーへのアクティブなTCP接続数を確認できます。

まず、Trident バックエンド構成を一覧表示してバックエンド名を確認し、それを記述して vserver 名とデータ LIF を取得します。

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

次に ONTAP クラスターにログインし、上記の出力からデータ LIF を置き換えて以下のコマンドを実行します：

```
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.
```

図 **2.** サンプルの提示

ポッドが実行状態になったら、データベースに接続してデータ操作を確認します。

手順

1. PostgreSQL サービスのポートをローカルマシンに転送します：

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

2. 別のターミナルから、psql を使用してデータベースに接続します：

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

3. データベースとテーブルを作成し、テーブルにデータを入力します。

```
CREATE DATABASE employee;
```

新しいデータベースに切り替える（psqlメタコマンド）：

```
\c employee
```

テーブルを作成し、行を挿入して、結果をクエリします。

```
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;
```

ビデオデモ

以下のビデオでは、vSphere Kubernetes クラスターへの Trident のインストールと、Trident を使用した永続ストレージを備えた PostgreSQL データベースのデプロイについて説明します。

vSphere Kubernetes Service クラスター上でワークロードをデプロイする（Tridentを使用したONTAP永続ストレージ）

データ保護

NetApp Consoleを使用した、vSphere Kubernetes Serviceクラスターにおけるコンテナアプリケーションのバックアップとリストア

vSphere Kubernetes Service （VKS）クラスターのコンテナアプリケーションをNetApp Consoleを使用してバックアップおよびリストアします。この手順では、ONTAP S3オブジェクトストレージをバックアップターゲットとして使用し、NetApp ConsoleのBackup and Recovery Serviceを通じてバックアップとリストア操作を作成および管理する方法について説明します。

NetApp Console は、オンプレミスおよびクラウド環境全体にわたるストレージとデータサービスの一元管理を提供します。統合されたコントロール、運用の簡素化、および安全な管理を実現します。"console.netapp.com" のユーザーインターフェースからアクセスできるデータサービスと管理機能がいくつかあります。NetApp Console の詳細については、"NetApp Console のドキュメント" を参照してください。

このセクションでは、Kubernetesワークロード（具体的にはvSphere Kubernetes Serviceクラスター上のコンテナアプリケーション）向けのNetApp Backup and Recoveryデータサービスに焦点を当てます。NetApp Backup and Recoveryサービスを使用すると、単一のインターフェイスからKubernetesアプリケーションを検出、管理、作成し、保護ポリシーに関連付けることができます。完全なガイドについては、"NetApp Backup and Recovery ドキュメント"を参照してください。

バックアップとリカバリのワークフロー

1. **Console** エージェントがあることを確認してください

   VKSクラスタが稼働している環境に、コンソールエージェントがデプロイされていることを確認してください。コンソールエージェントのインストールの詳細については、"NetApp Console Setupドキュメント" を参照してください。

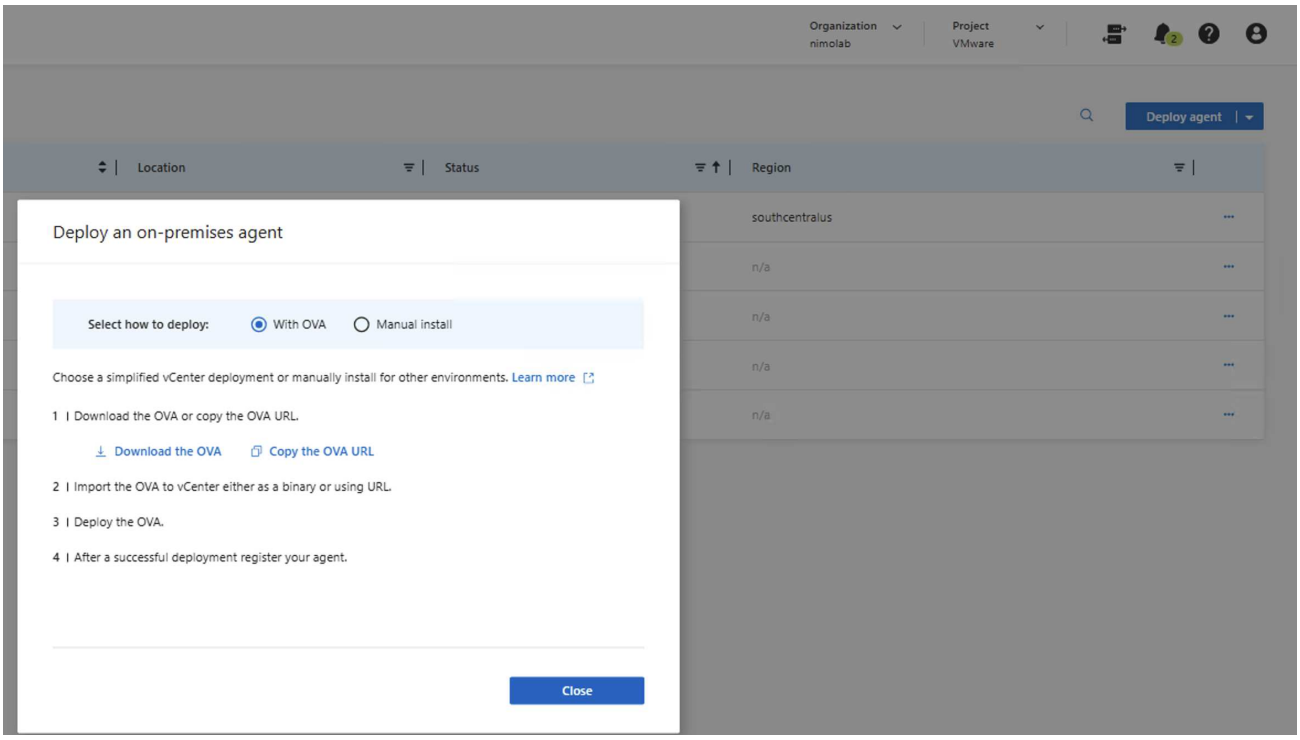
   ▼ 例を表示

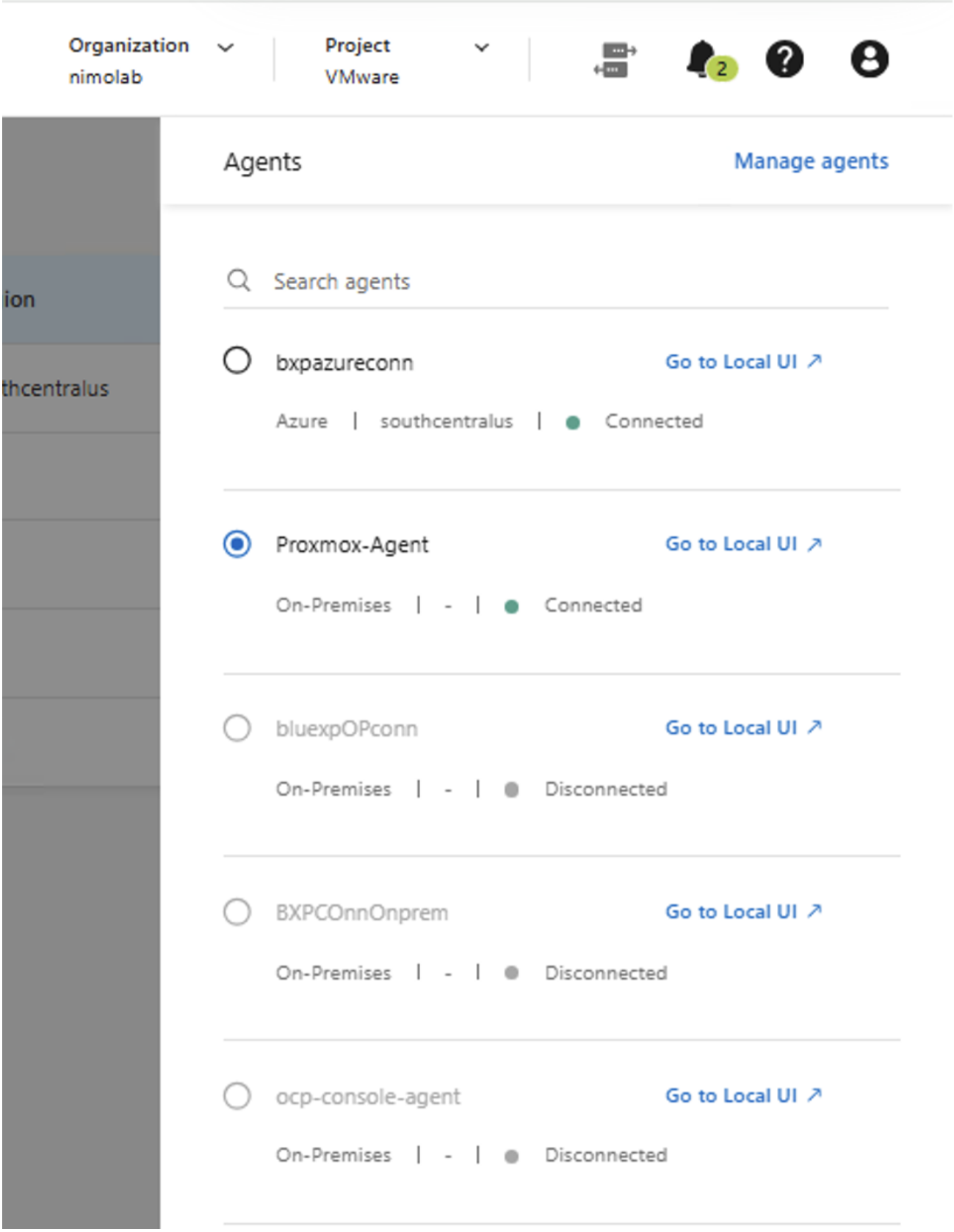
   NetApp Console で、スクリーンショットに示すように「Deploy Agent」をクリックし、VKS クラスタが稼働している環境にエージェントをデプロイします。

   この例では、オンプレミス **> OVA** を使用 を選択し、OVA URL をコピーして、その URL を vCenter で使用してコンソールエージェントをデプロイします。

   URLにエージェントのIPアドレスを使用して登録してください。詳細な手順については、"ドキュメン

ト"を参照してください。エージェントが登録されると、以下のスクリーンショットに示すように NetApp Console で確認できます。





2. **ONTAP** クラスターを **NetApp Console** に追加してバックアップとリカバリを有効にする

   ワークロードを保護するには、プライマリ ONTAP クラスターを NetApp Console に追加し、Backup and Recovery サービスを有効にする必要があります。手順については、"NetApp Backup and Recoveryのセットアップ"を参照してください。

3. **NetApp Console** で **vSphere Kubernetes** クラスターを検出する

▼ 例を表示

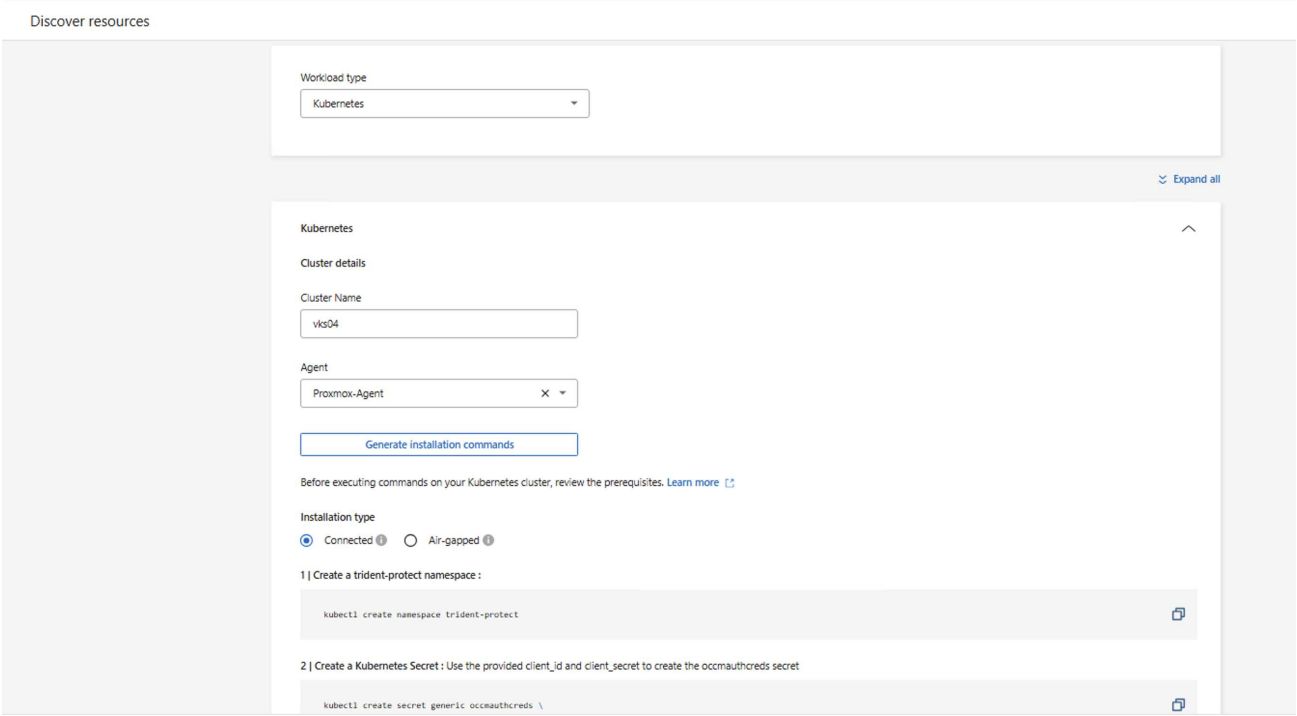
この手順を実行するには、vSphere Kubernetesクラスターが実行されており、同じ環境にConsoleエージェントがデプロイされている必要があります。次の手順に従って、NetApp ConsoleでvSphere Kubernetesクラスターを検出します。

a. NetApp Console で、インベントリ **> Kubernetes** を選択し、vSphere Kubernetes クラスターの詳細を入力します。

b. vSphere Kubernetes クラスターが配置されているのと同じ環境にデプロイされたエージェントを選択します。

c. 提供されたコマンドをコピーしてTrident Protectをインストールし、vSphere Kubernetesクラスターへ接続されたマシンから実行します。

d. Trident Protect のインストールが正常に完了したら、**Discover** をクリックします。NetApp Console は、エージェントを介して vSphere Kubernetes クラスターとそのすべてのリソースを検出し、UI に表示します。



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$
```

Applications | Clusters | Namespaces | Virtual machines

Clusters (4)

Name	Kubernetes Version	Kubernetes distribution	Trident protect version	Applications	Namespaces
demo-vks Pending				0 Applications (0 out of 0 protected)	0
vks-demo01 Pending				0 Applications (0 out of 0 protected)	0
vks03-old Deleting		kubernetes	26.08.0-console	0 Applications (0 out of 0 protected)	17
vks04 Connected		kubernetes	26.08.0-console	0 Applications (0 out of 0 protected)	16

4. **ONTAP S3**をバックアップターゲットとして設定する

   この例では、バックアップ先としてONTAP S3オブジェクトストレージを使用します。AWS S3、Azure Blob、Google Cloud Storage、またはStorageGRIDをバックアップターゲットとして使用することもできます。

   詳細については、"NetApp Console のドキュメント"を参照してください。

   ONTAP S3をNetApp Consoleのバックアップターゲットとして追加するには、ONTAPクラスターから次の情報が必要になります。

```
S3 Endpoint:
Access Key:
Secret Key:
Bucket Name:
```

これらの値を取得するには、System Manager を使用して ONTAP クラスタで次の手順を実行します。

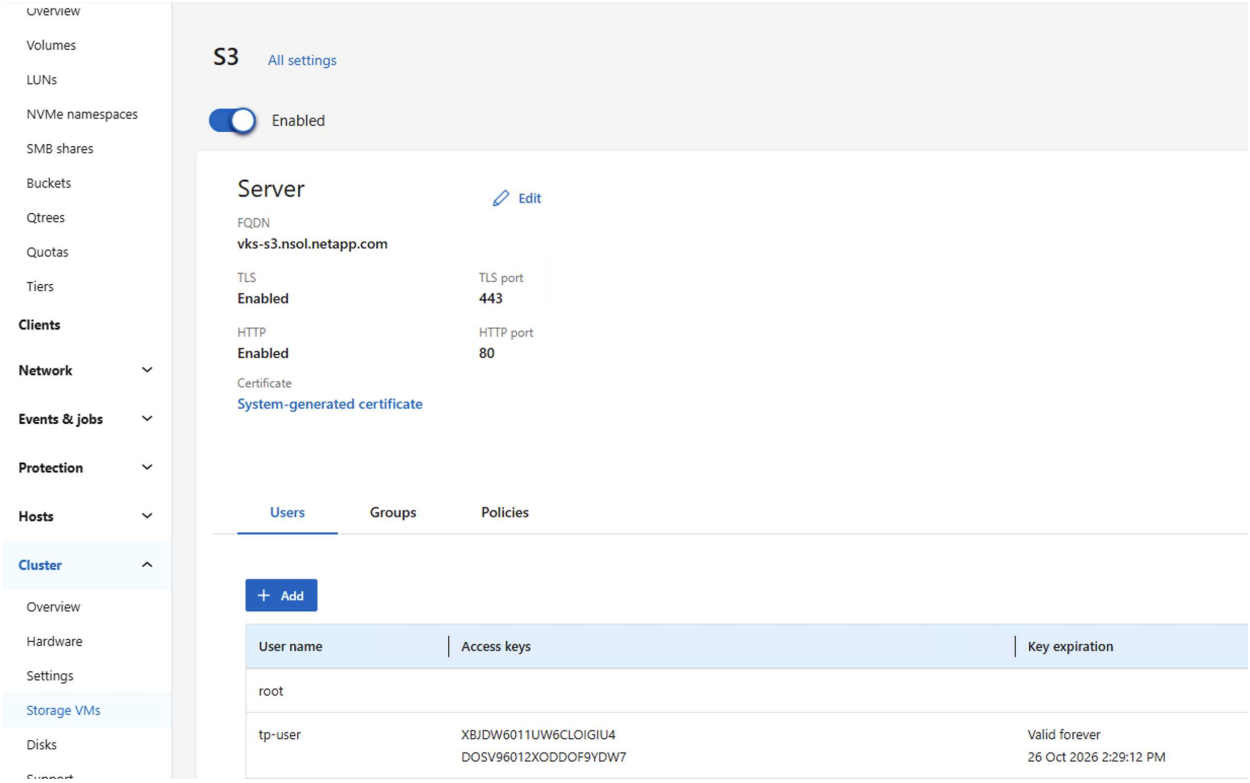
a. バックアップデータを保存するために、ONTAPクラスタにS3対応のSVMを作成します。このSVMのS3エンドポイントURLをメモしておいてください。

b. SVMのS3設定で、ユーザーを作成し、必要なアクセス権限を割り当てます。このユーザーのアクセスキーとシークレットキーを控えておいてください。

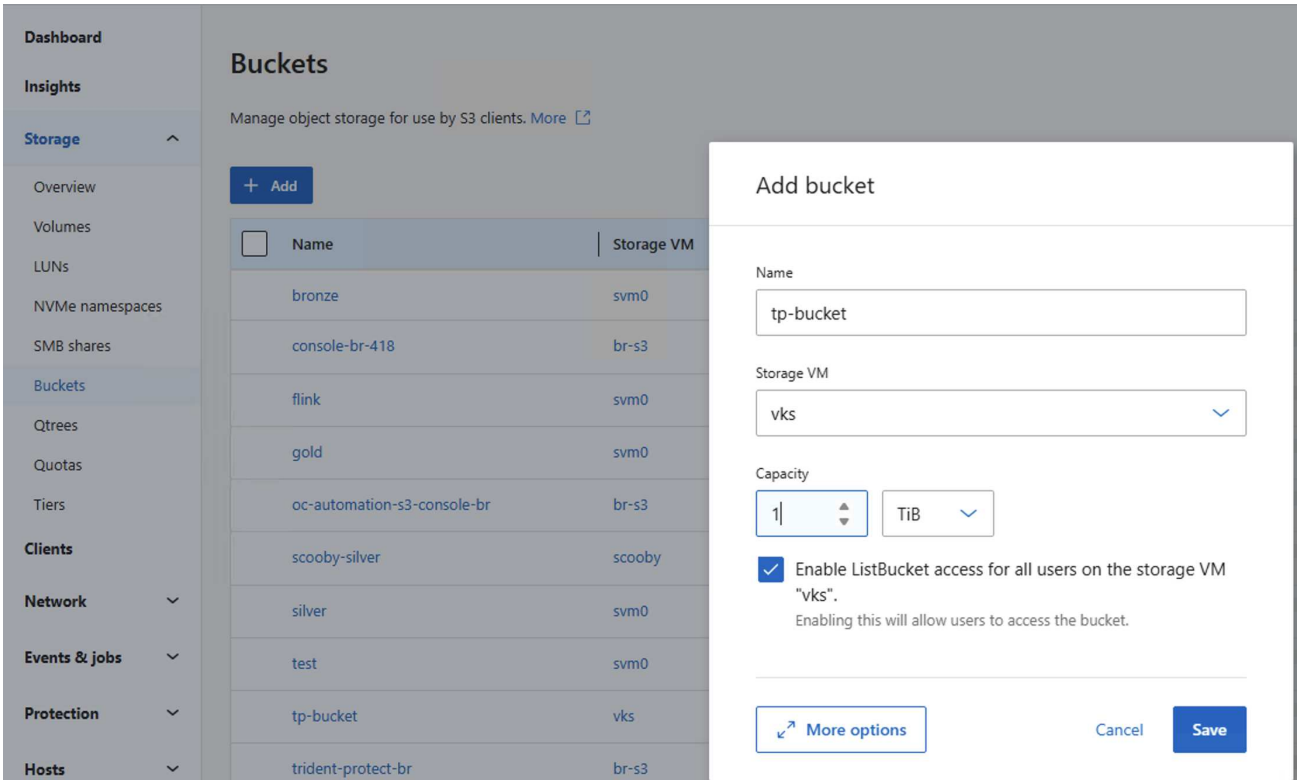
> Trident Protectでは、S3ユーザーに少なくとも `PutObject`、 `GetObject`、 `ListBucket`、および `DeleteObject`の権限が必要です。これらの権限がない場合、AppVaultのバックアップおよびリストア操作はアクセス拒否エラーで失敗します。詳細については、"Trident Protect AppVault ドキュメント"を参照してください。

▼ 例を表示

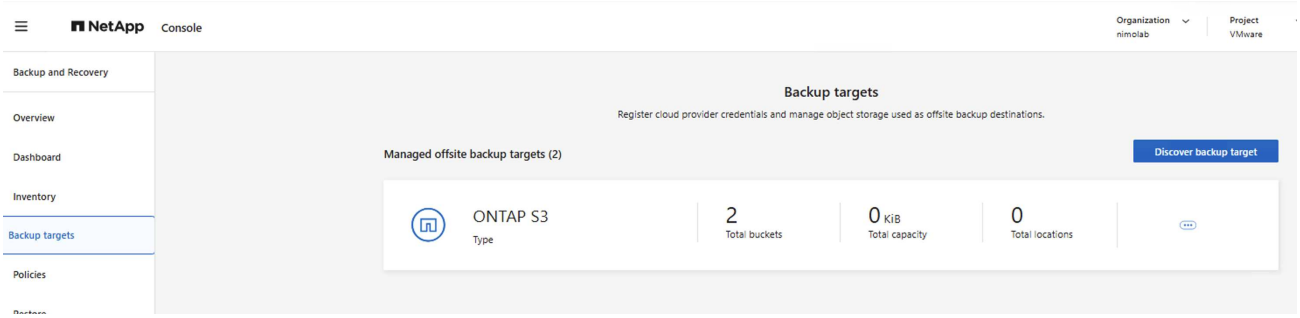
i. **S3**ユーザーを作成し、アクセスキーとシークレットキーをダウンロードします

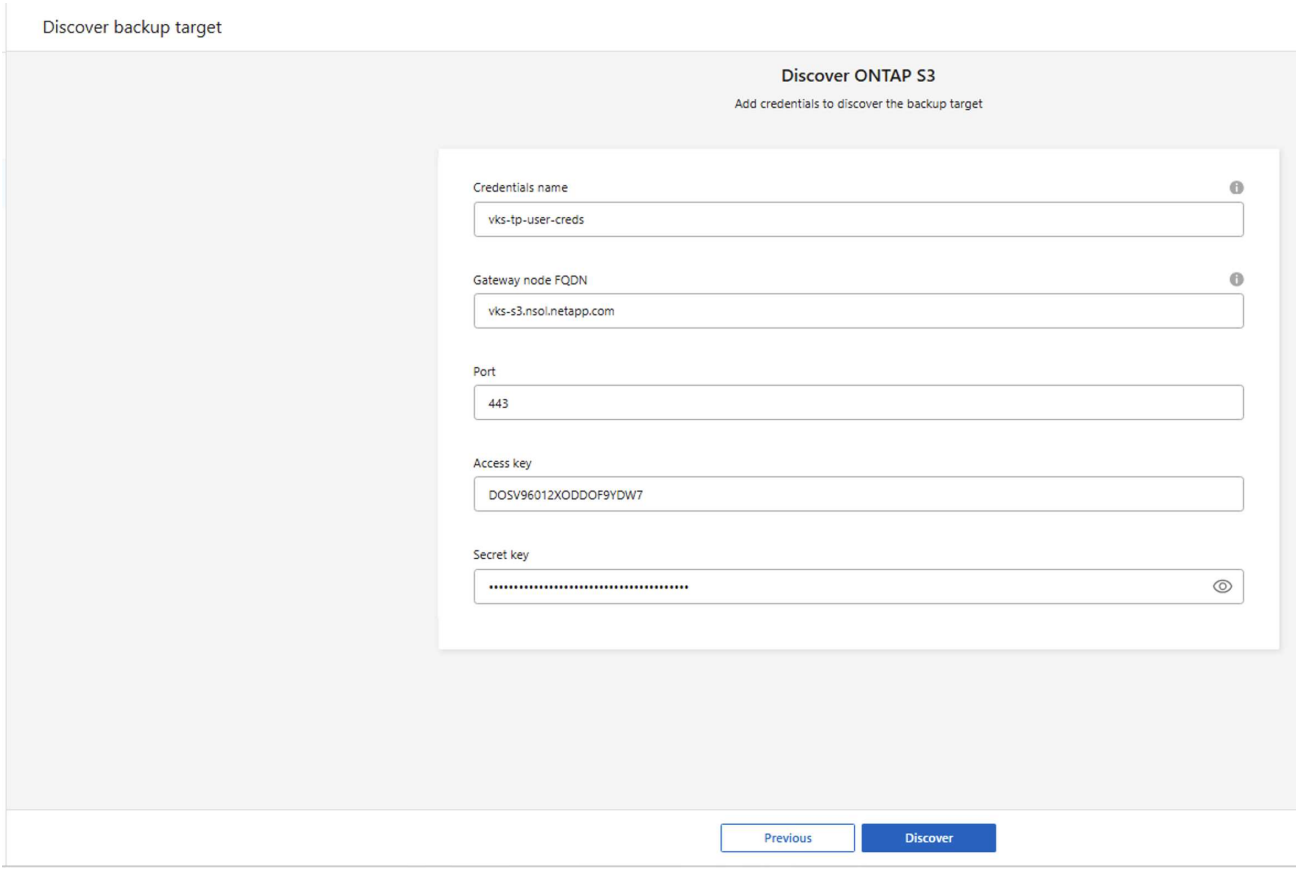


A. S3対応のSVMにバケットを作成します。NetApp ConsoleでONTAP S3オブジェクト ストレージをバックアップ ターゲットとして追加する際に使用するバケット名をメモしておきます。



S3エンドポイント、アクセスキー、シークレットキー、バケット名を使用して、ONTAP S3オブジェクトストレージをバックアップターゲットとして NetApp Console に追加します。





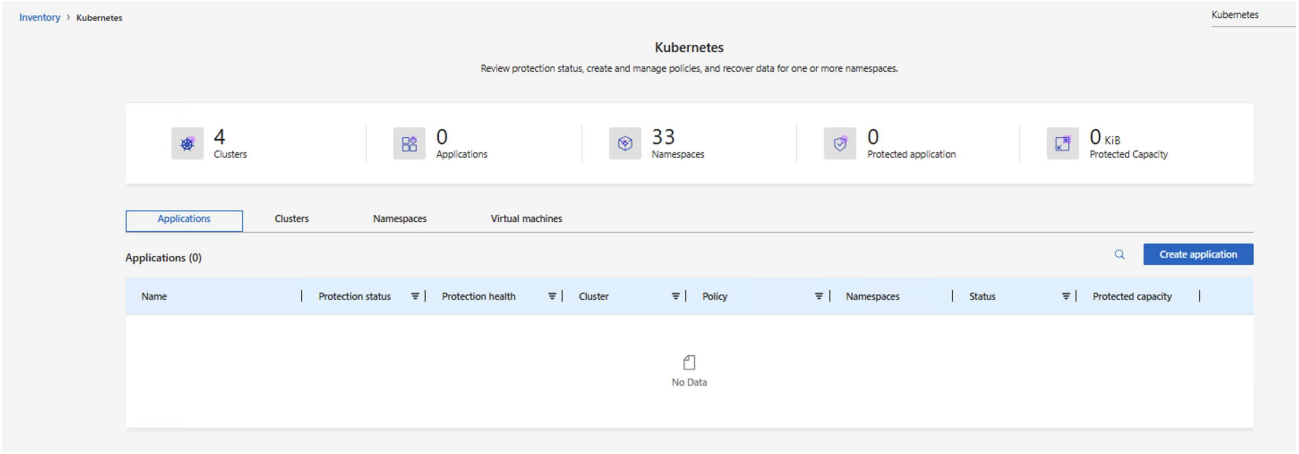
5. **NetApp Console** で、コンテナワークロード用のアプリケーションを作成し、バックアップを使用して保護します

> Kubernetesアプリケーションを作成および保護するには、バックアップおよびリカバリのスーパー管理者、またはバックアップおよびリカバリのバックアップ管理者ロールが必要です。Kubernetesアプリケーションを復元するには、バックアップおよびリカバリのスーパー管理者、またはバックアップおよびリカバリの復元管理者ロールが必要です。

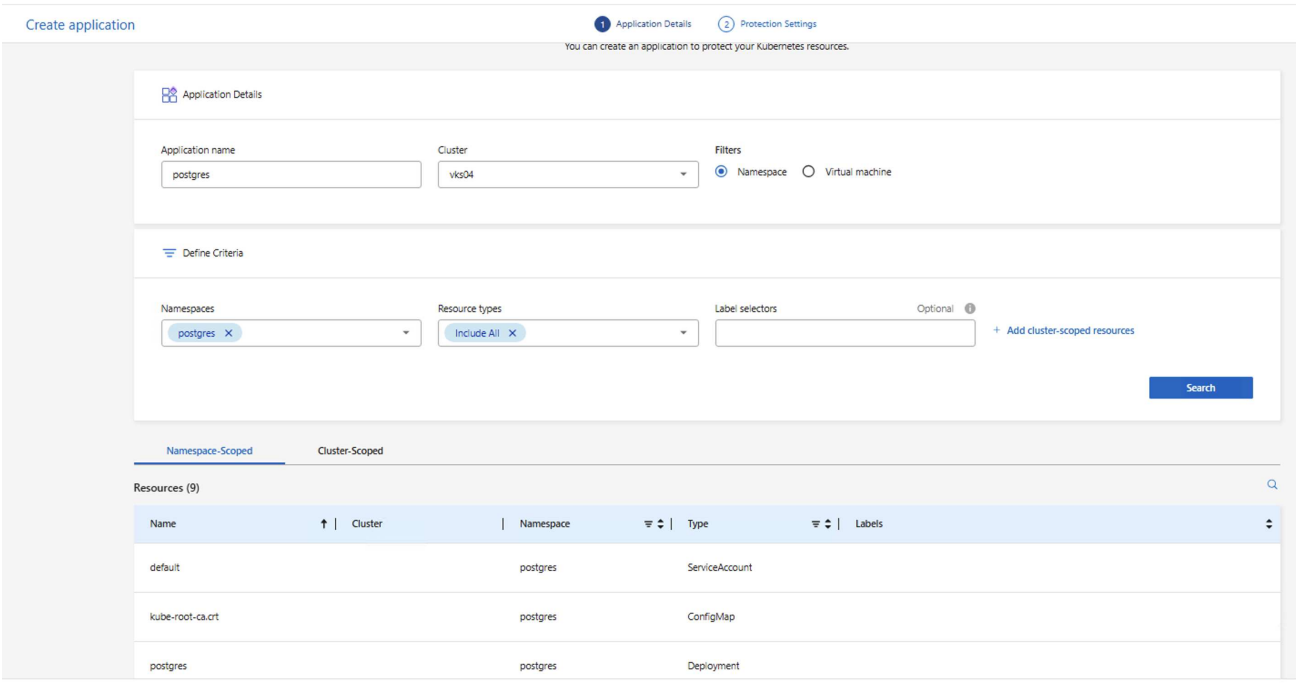
▼ 例を表示

この手順では、保護が必要な vSphere Kubernetes クラスター内のコンテナアプリケーション用に、NetApp Console でアプリケーションを作成します。ネームスペースを使用すると、vSphere Kubernetes クラスター内のネームスペースにあるすべてのコンテナアプリケーションを保護するためのアプリケーションを作成できます。

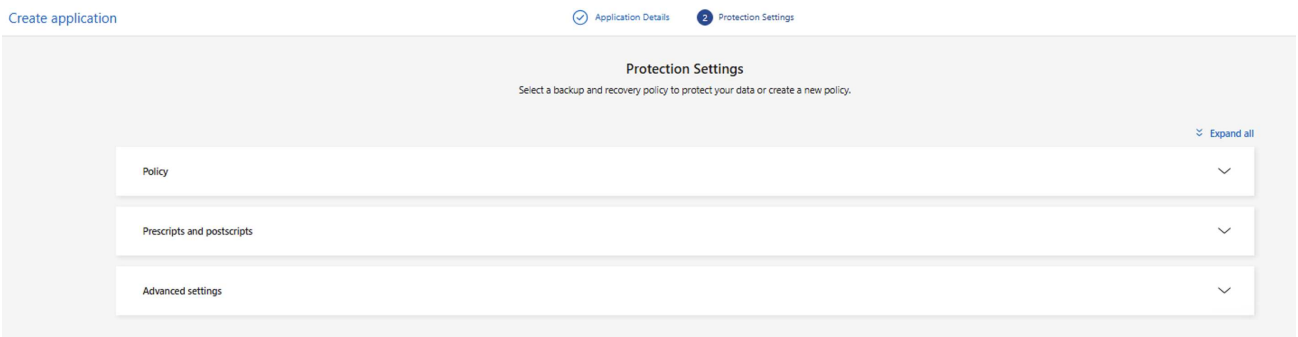
また、アプリケーションに関連付ける保護ポリシーも必要です。新しい保護ポリシーを作成するか、既存のものを使用することができます。



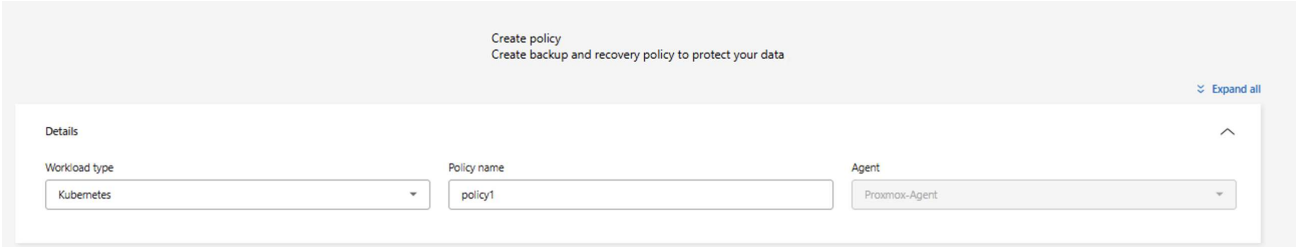
アプリケーションの詳細を入力し、アプリケーションの条件を定義します。この例では、vSphere Kubernetesクラスターの名前空間内のすべてのコンテナアプリケーションに対してアプリケーションが作成されます。「検索」をクリックすると、アプリケーションに含まれる名前空間スコープのリソースの一覧が表示されます。「次へ」をクリックして続行します。

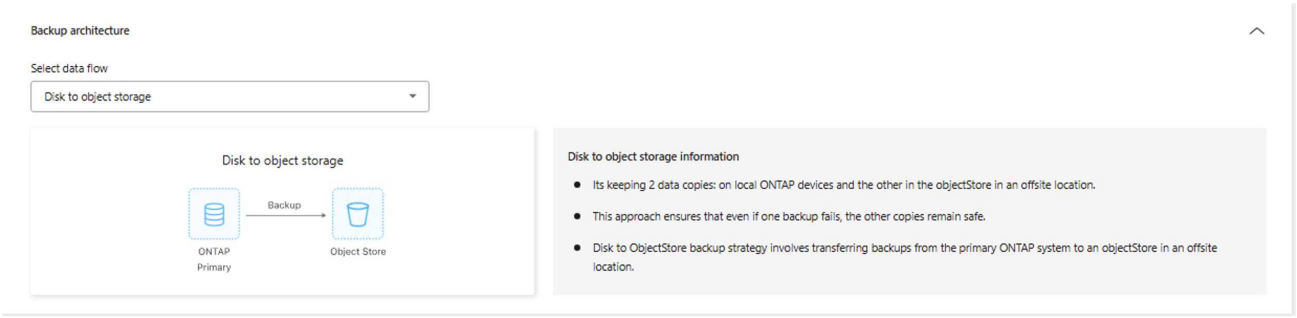


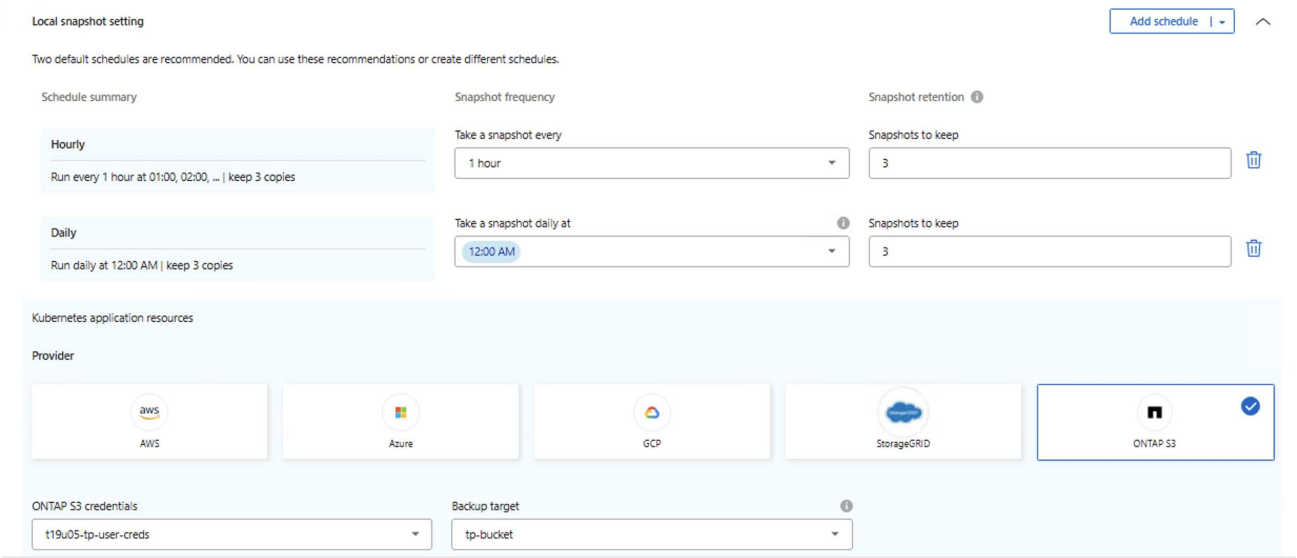
次に、アプリケーションの保護設定を行う必要があります。新しい保護ポリシーを作成するか、既存のものを使用することができます。この例では、アプリケーション用に新しい保護ポリシーが作成され、アプリケーションに関連付けられます。「ポリシー」を展開し、「新しいポリシーを作成」をクリックします。

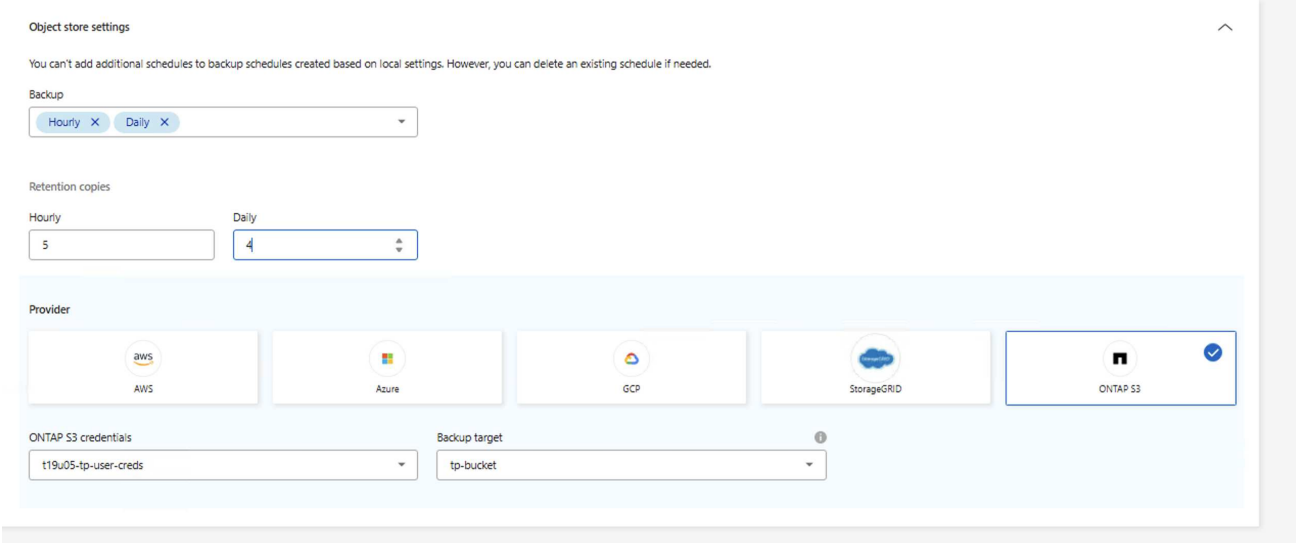


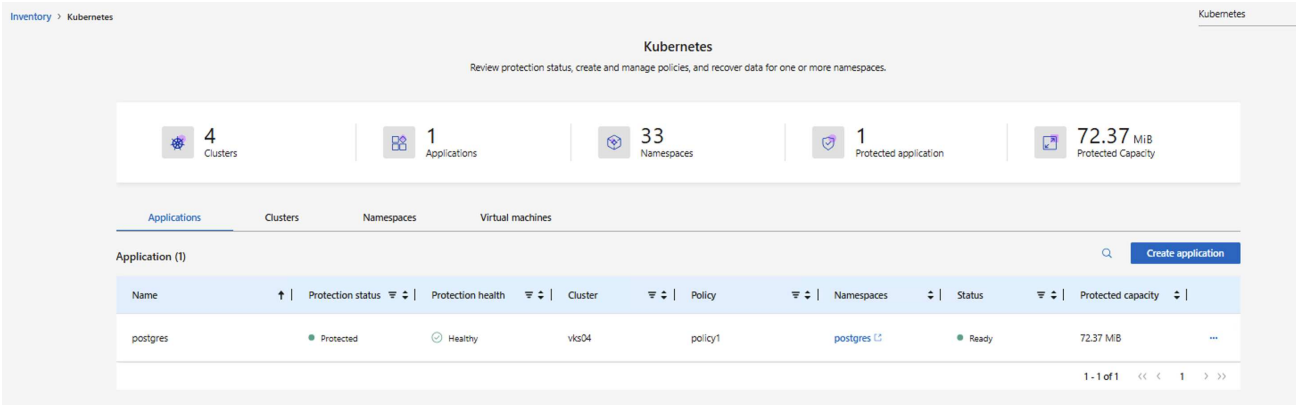
ポリシーに名前を付け、バックアップアーキテクチャを選択してください。この例では、ディスクからオブジェクトストレージへの構成が選択されています。プライマリストレージでスナップショットが取得される頻度を制御するローカルスナップショットスケジュールを定義します。次に、オブジェクトストレージのバックアップ設定を個別に構成します。バックアップターゲットを選択し（この例では ONTAP S3）、スナップショットデータがオブジェクトストレージにコピーされるタイミングを制御する転送スケジュールを設定して、保持コピーの数を指定します。転送スケジュールはスナップショットスケジュールとは独立しているため、リソースは各スナップショットが取得された直後ではなく、転送スケジュールに従ってオブジェクトストレージにコピーされます。必要に応じて、最大再試行回数と再試行間隔を変更することもできます。続行するには「作成」をクリックしてください。アプリケーションが検出され、保護ポリシーがそれに関連付けられます。









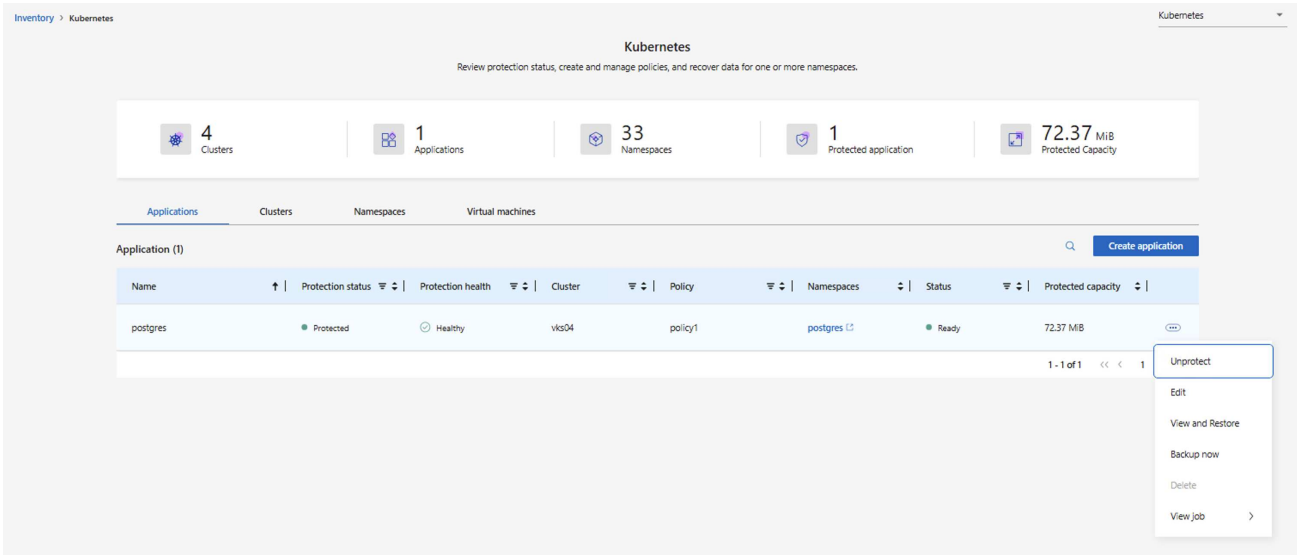


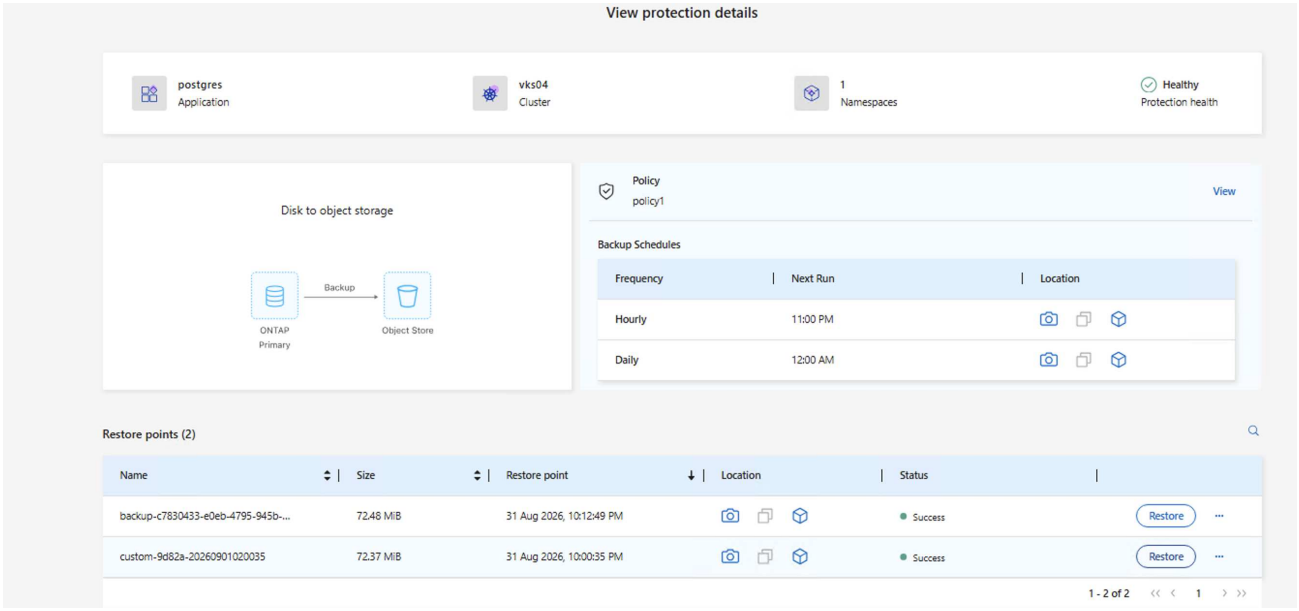
6. バックアップからアプリケーションを復元する

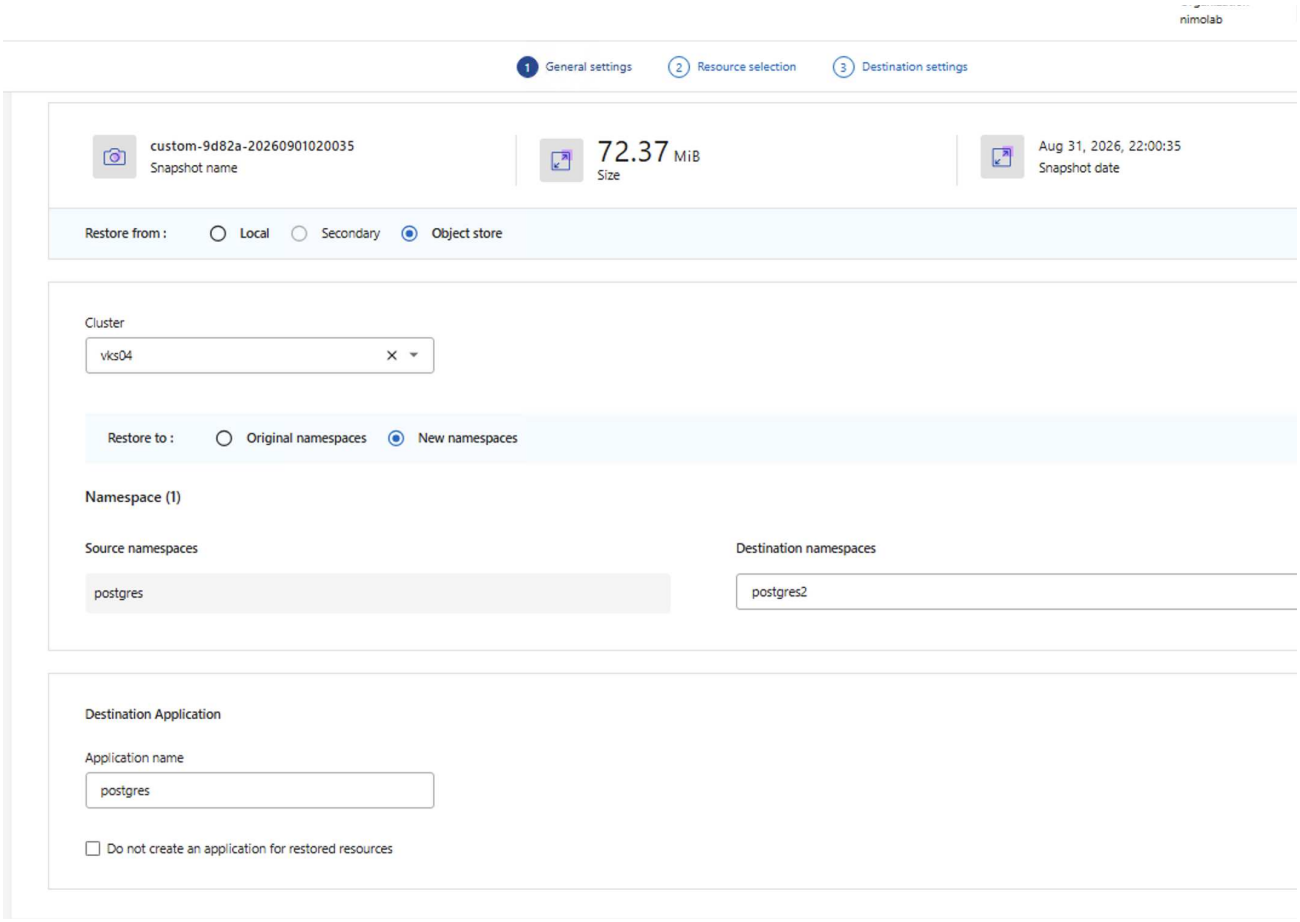
> Kubernetesアプリケーションを復元するには、Backup and Recoveryのスーパー管理者ロール、またはBackup and Recoveryの復元管理者ロールが必要です。

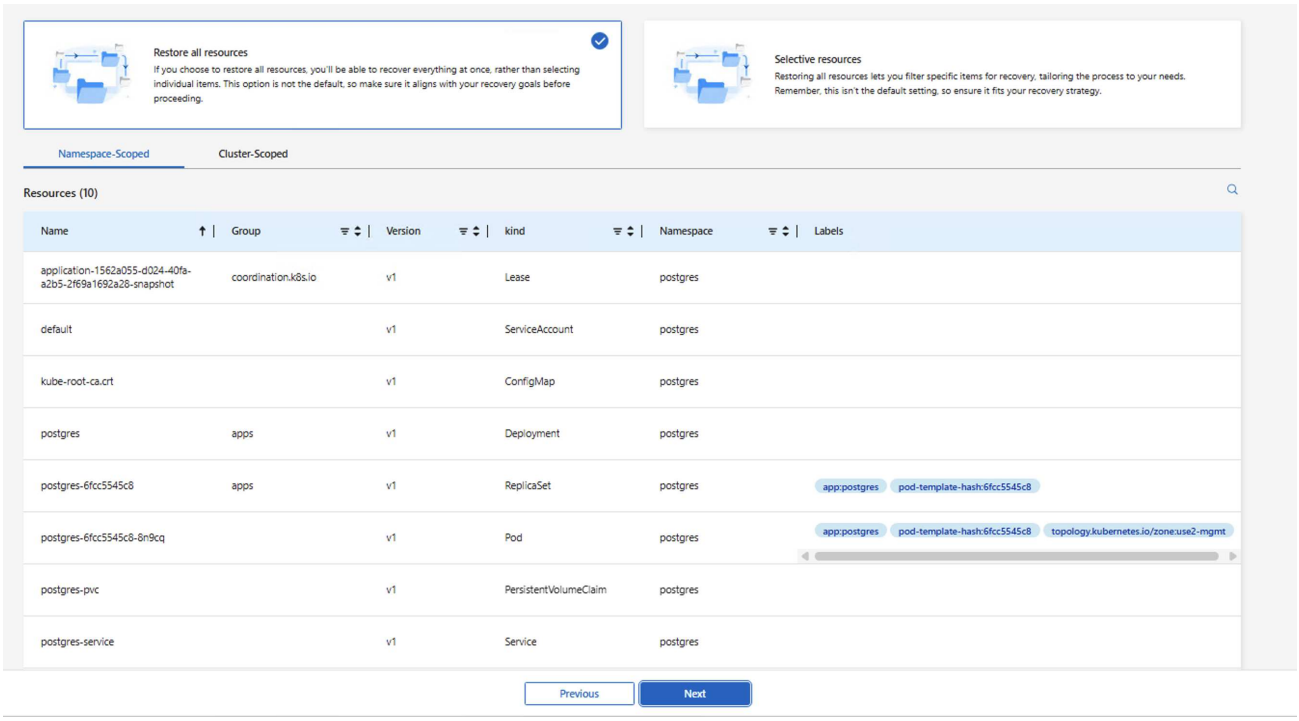
▼ 例を表示

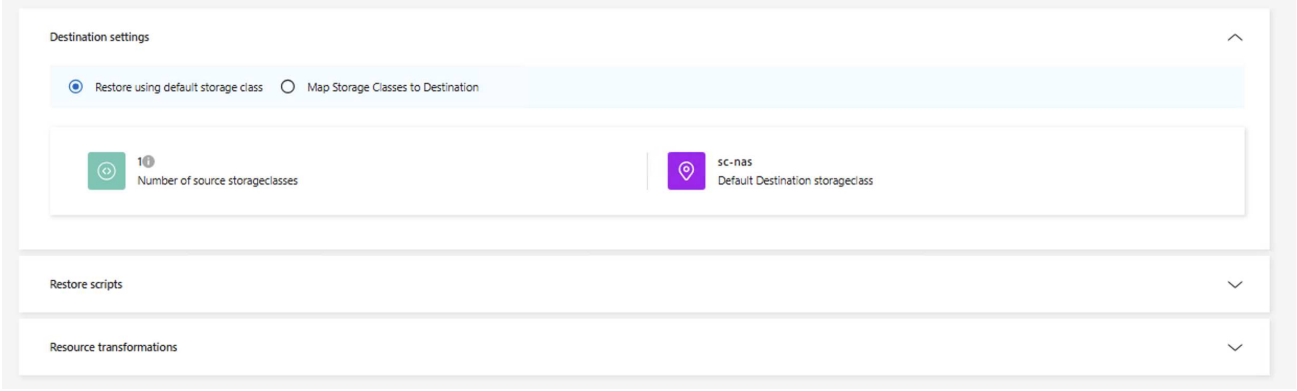
- アプリケーションを復元するには、ローカルスナップショット、オブジェクトストレージバックアップ、またはセカンダリターゲットからのバックアップなど、利用可能な復元ポイントのいずれかを選択できます。
- アプリケーションを元の名前空間に復元することも、別の名前空間に復元することもできます。
- 復元に含めるリソースを選択できます。
- 復元したアプリケーションリソースに使用するストレージクラスを選択できます。


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
Unprotect
Edit
View and Restore
Backup now
Delete
View job

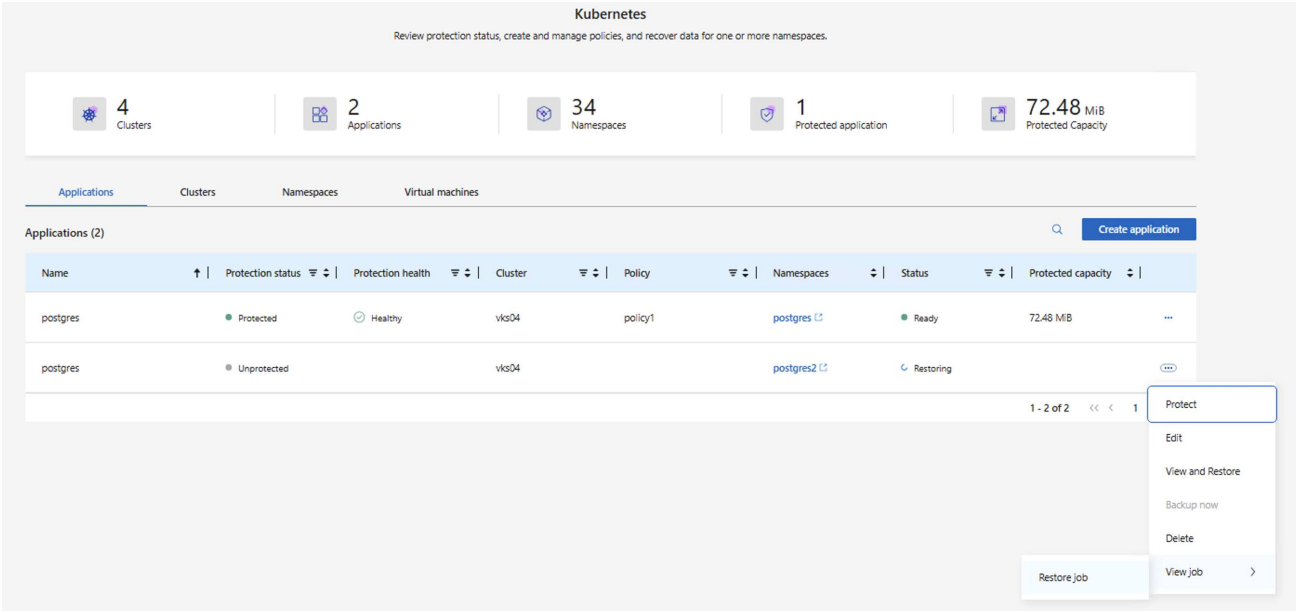

View protection details
postgres
Application
vks04
Cluster
1
Namespaces
Healthy
Protection health
Disk to object storage
Backup
ONTAP Primary
Object Store
Policy
policy1
View
Backup Schedules
Frequency
Next Run
Location
Hourly
11:00 PM
Daily
12:00 AM
Restore points (2)
Name
Size
Restore point
Location
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

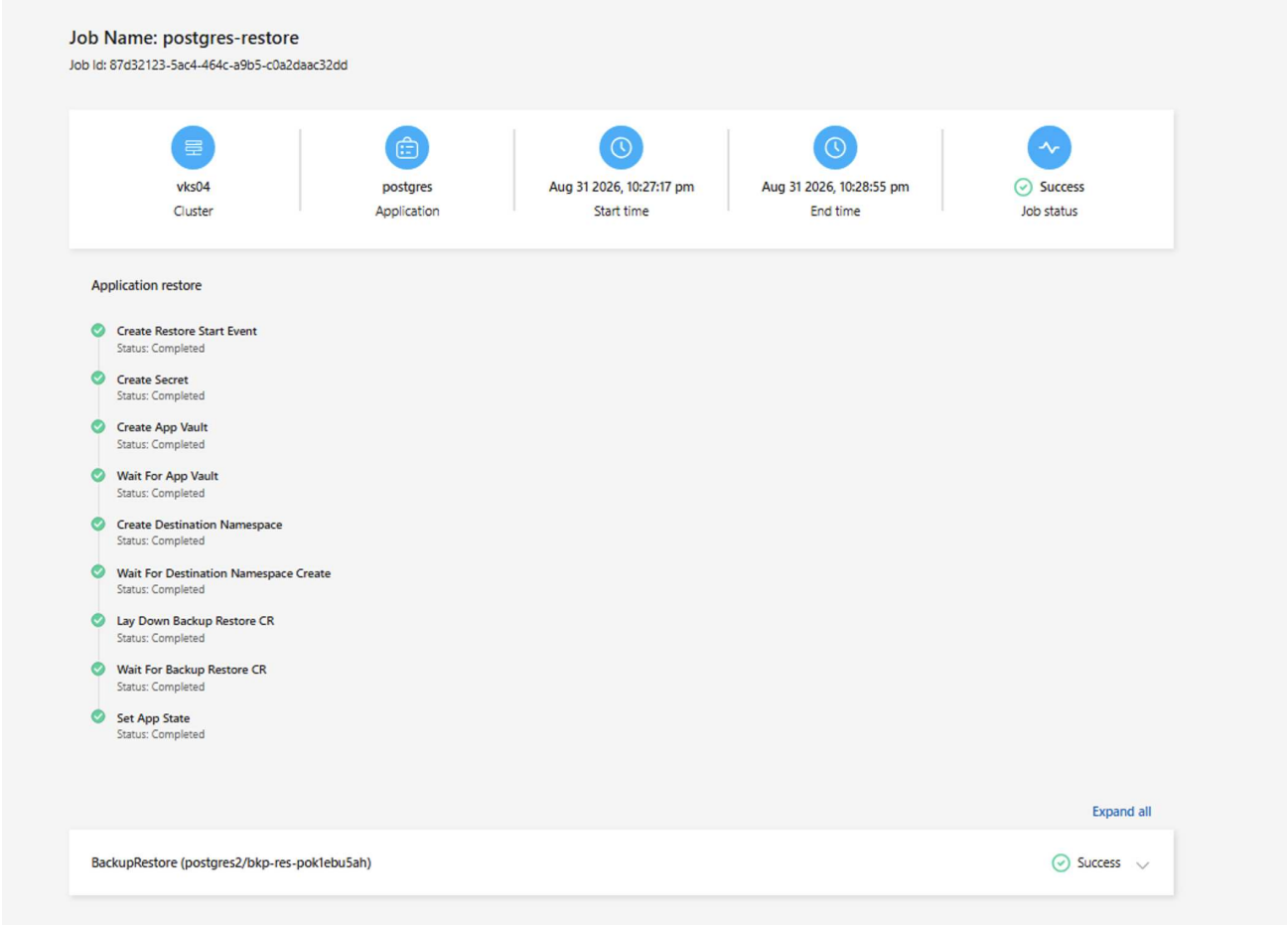

nimolab
1 General settings
2 Resource selection
3 Destination settings
custom-9d82a-20260901020035
Snapshot name
72.37 MiB
Size
Aug 31, 2026, 22:00:35
Snapshot date
Restore from :
Local
Secondary
Object store
Cluster
vks04
Restore to :
Original namespaces
New namespaces
Namespace (1)
Source namespaces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

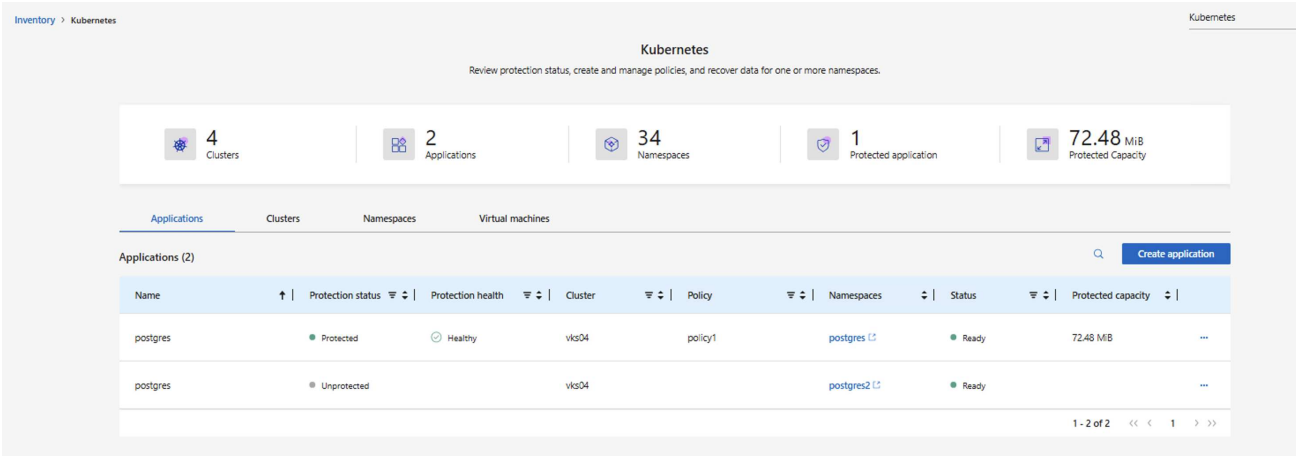

Restore all resources
If you choose to restore all resources, you'll be able to recover everything at once, rather than selecting individual items. This option is not the default, so make sure it aligns with your recovery goals before proceeding.
Selective resources
Restoring all resources lets you filter 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



復元ジョブの進行状況は、NetApp Console のモニターセクションで確認できます。復元ジョブが完了すると、vSphere Kubernetes クラスターで復元されたリソースを確認できます。







vSphere Kubernetes Service クラスタのコンテナアプリケーションを **Trident Protect** を使用してバックアップおよびリストアする

vSphere Kubernetes Service （VKS） クラスター内のコンテナアプリケーションを、Trident Protect のスナップショットとバックアップを使用してバックアップおよびリストアします。この手順には、ONTAP S3 オブジェクトストレージを使用した AppVault の作成、Kubernetes リソースオブジェクトと永続ボリュームを含むアプリケーションデータをキャプチャするための Trident Protect の設定、および必要に応じたデータのリストアが含まれます。

VKSクラスタ内で実行されるコンテナアプリケーションは、ワーカーノードの名前空間においてKubernetesワークロードとして管理されます。アプリケーションのメタデータと永続ボリュームの両方を保護することが重要です。そうすることで、それらが失われたり破損したりした場合でも、復旧できるようになります。

VKSアプリケーションの永続ボリュームは、"Trident CSI"を使用してVKSクラスターと統合されたONTAPストレージによってバックアップできます。この手順では、"Trident Protect"を使用してアプリケーションスナップショットを作成します。スナップショットのメタデータはONTAPオブジェクトストレージに保存され、ボリュームスナップショットデータはストレージバックエンドに残ります。また、バックアップのデータはONTAPオブジェクトストレージにコピーされます。必要に応じて、スナップショットまたはバックアップから復元できます。

Trident Protectは、Kubernetesクラスタ上のアプリケーションのスナップショット、バックアップ、復元、およびディザスタリカバリを可能にします。Trident Protectで保護できるデータには、アプリケーションに関連付けられたKubernetesリソースオブジェクトと、それらの永続ボリュームが含まれます。

以下は、このセクションの例で使用されている各種コンポーネントのバージョンです

- VMware Cloud Foundation (VCF) 9.1 と vSphere Kubernetes サービス
- "Trident 26.06"
- "Trident Protect 26.06"
- "ONTAP 9.16"

vSphere Kubernetes クラスターに Trident Protect をインストールする

▼ **Trident Protect**のインストール

Helm を使用して、vSphere Kubernetes Service クラスタに Trident Protect をインストールします。このセクションの例では、Trident Protect CLI である `tridentctl-protect`を使用します。"Trident Protect CLI ドキュメント"の手順に従ってインストールしてください。Trident Protect のその他のインストール方法については、"Trident Protect ドキュメント"を参照してください。

まず、 `trident-protect`名前空間を作成してラベルを付けます。Trident Protect は特権ワークロードを実行するため、 `enforce=privileged`ラベルが必要です。これがないと、Kubernetes Pod セキュリティアドミッションコントローラーが Trident Protect Pod の起動をブロックします。

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

次に、Helmリポジトリを追加し、Trident Protectを名前空間にインストールします。

```
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
```

Trident Protect ポッドが `PodSecurity`アドミッションエラーで起動しない場合、インストー

ル前にポッドのセキュリティラベルが `trident-protect`ネームスペースに適用されていない可能性があります。 `--overwrite`を使用して再適用し、ポッドが起動することを確認してください。

```
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$
```

オブジェクトストレージ用の**App Vault**を作成する

▼ **AppVault**を作成する

アプリケーションのスナップショットとバックアップを作成する前に、Trident Protect でオブジェクトストレージを構成する必要があります。これは、AppVault CR を作成することで行います。管理者のみが AppVault CR を作成および構成できます。

AppVaultオブジェクトは、ストレージバケットのKubernetesカスタムリソース表現です。AppVault CRには、バックアップ、スナップショット、リストア操作、SnapMirrorレプリケーションなどの保護操作でバケットを使用するために必要な設定が含まれています。

以下の手順では、ONTAP S3 用に設定された AppVault CR を作成します。ONTAP クラスタの SVM に S3 オブジェクトストアサーバーを作成します。.オブジェクトストアサーバーにバケットを作成します。.SVM に S3 ユーザーを作成します。アクセスキーとシークレットキーは安全な場所に保管してください。

+ 注記：Trident Protect では、S3 ユーザーが少なくとも `PutObject`、 `GetObject`、 `ListBucket`、および `DeleteObject`の権限を持つ必要があります。これらの権限がない場合、AppVault のバックアップおよびリストア操作はアクセス拒否エラーで失敗します。詳細については、"Trident Protect AppVault ドキュメント"を参照してください。.VKS クラスターで、ONTAP S3 クレデンシャルを格納するシークレットを作成します。.ONTAP S3 の AppVault オブジェクトを作成します。

ONTAP S3 向けの **Trident Protect AppVault** の設定

以下のマニフェストは、本番環境で必須となる証明書検証を有効にしたHTTPSを使用しています。ONTAP S3エンドポイントがクラスターによってすでに信頼されている公開CAによって署名された証明書を使用している場合、追加の証明書設定は不要です。自己署名証明書または内部CA証明書を使用する場合は、マニフェストに示されているように、 `rootCA`フィールドを使用してカスタムルートCA証明書を指定してください。隔離された非本番環境テスト用のHTTP設定が必要な場合は、このセクションの最後にあるラボ専用バリアントを参照してください。

```
# 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
```

ラボ専用バリアント（**HTTP**、証明書検証なし）

機密データが存在する隔離されたラボ環境でのみ、以下の `s3`設定を使用してください。本番環境ではこれらの設定を使用しないでください。

```
    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$
```

Trident Protect アプリケーションの作成

▼ **Trident Protect** アプリケーションの作成

この例では、postgres名前空間にインストールされているサンプルpostgresアプリケーションのデータ保護について説明します。インストールの詳細については、"NetApp ストレージを使用して VKS ワークロードをデプロイする"を参照してください。

> ⓘ サンプルPostgreSQL PVCはRWOアクセスモードを要求します。アクセスモードはPVCマニフェスト内で明示的に指定する必要があります。Tridentはストレージ プロトコルに基づいてアクセスモードを自動的に選択しません。RWXはNASを基盤とするPVCでサポートされており、SANを基盤とするPVCも追加設定を行うことでRWXをサポートできます。詳細については、"Trident Protect ドキュメント"を参照してください。

この例では、postgres 名前空間には 1 つのアプリケーションがあり、Trident Protect Application の作成時に名前空間のすべてのリソースが含まれます。

```
# 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
```

バックアップを作成してアプリを保護する

▼ バックアップを作成する

オンデマンドバックアップを作成する

以前作成した postgres-app のバックアップを作成します。このバックアップには、postgres 名前空間内のすべてのリソースを含めます。バックアップを保存する appvault 名を指定してください。

```
# 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
```

スケジュールに基づいてバックアップを作成する

バックアップのスケジュールを作成し、保持するバックアップの粒度と数を指定します。

```
# 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
```

バックアップからの復元

▼ バックアップから復元する

アプリケーションを同じ名前空間に復元する

この例では、バックアップ postgres-backup-on-demand には postgres-app のバックアップが含まれています。

アプリケーションを削除する前に、復元に使用する予定のバックアップが `Completed`状態であることを確認してください。進行中または失敗したバックアップは、アプリケーションの復元には使用できません。

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

バックアップ状態が `Completed`であることを確認した後、postgresアプリケーションを削除し、PVCとpodオブジェクトが名前空間「postgres」から削除されていることを確認してください。

```
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.
```

次に、バックアップIn Placeリストアオブジェクトを作成します。

```
# 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
```

PostgreSQLアプリケーションのデプロイメント、サービス、Pod、およびPVCが復元されていることを確認してください。

```
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
```

アプリケーションを別の名前空間に復元する

まず、アプリを復元する新しい名前空間を作成します。この例では、postgres2 を使用します。postgres-app では、スケジュールによって作成された 1 時間ごとのバックアップが利用可能になりました。このバックアップを使用して、アプリケーションを新しい名前空間 postgres2 に復元してください。

```
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
```

バックアップのパスを取得するには、次のコマンドを使用します `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
```

新しい名前空間 postgres2 に postgres アプリケーション オブジェクトと PVC が作成されていることを確認してください。

```
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$
```

スナップショットを使用してアプリを保護する

▼ スナップショットを作成する

オンデマンドスナップショットの作成 アプリのスナップショットを作成し、スナップショットのメタデータが保存される AppVault を指定します。ボリュームスナップショットデータ自体はオブジェクトストレージにコピーされず、ストレージバックエンドに残ります。

```
# 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$
```

スナップショットのスケジュールを作成する スナップショットのスケジュールを作成します。保持するスナップショットの粒度と数を指定します。

```
# 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
```

スナップショットから復元する

▼ スナップショットから復元する

スナップショットから同じ名前空間にアプリケーションを復元する

アプリケーションを削除する前に、リストア元として使用する予定のスナップショットが `Completed`状態であることを確認してください。進行中または失敗したスナップショットは、アプリケーションのリストアには使用できません。

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

スナップショットの状態が `Completed`であることを確認した後、postgres名前空間からpostgres用のアプリケーションオブジェクトとPVCを削除します。

```
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$
```

スナップショットからスナップショットIn Placeリストアオブジェクトを作成します。

```
# 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
```

アプリケーションのオブジェクトとPVCがpostgres名前空間に作成されていることを確認してください。

```
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$
```

スナップショットから別の名前空間にアプリケーションを復元する

バックアップから復元したpostgres2名前空間内のアプリケーションを削除します。

```
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$
```

スナップショットからスナップショット復元オブジェクトを作成し、名前空間マッピングを指定します。

```
# 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
```

アプリケーションのオブジェクトとPVCがpostgres2名前空間に復元されていることを確認してください。

```
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$
```

著作権に関する情報

商標に関する情報