vSphere Kubernetes Service with NetApp Storage

NetApp container solutions

NetApp
August 25, 2026

Table of Contents

vSphere Kubernetes Service with NetApp Storage

Learn about vSphere Kubernetes Service with NetApp ONTAP

vSphere Kubernetes Service (VKS) is a managed Kubernetes offering from VMware that enables organizations to deploy, manage, and scale containerized applications using Kubernetes. Paired with NetApp ONTAP storage, VKS delivers enterprise-grade persistence, performance, and data management for cloud-native workloads.

vSphere Kubernetes Service is typically integrated into VMware's broader ecosystem, such as VMware Tanzu, which provides a comprehensive suite of tools for Kubernetes management, application modernization, and DevOps practices.

Key features of vSphere Kubernetes Service

1. **Managed Kubernetes Clusters:** vSphere Kubernetes Service simplifies the deployment and management of Kubernetes clusters. It automates tasks like cluster creation, upgrades, scaling, and monitoring.
2. **Integration with VMware Infrastructure:** VKS integrates seamlessly with VMware vSphere, NSX-T, and other VMware solutions, enabling organizations to leverage their existing VMware infrastructure for Kubernetes workloads.
3. **Multi-Cloud and Hybrid Cloud Support:** vSphere Kubernetes Service supports deployment on private clouds, public clouds (e.g., AWS, Azure, Google Cloud), and hybrid cloud environments, giving organizations flexibility in managing their applications.
4. **Built-in Security:** VKS includes security features such as role-based access control (RBAC), policy enforcement, and integration with VMware NSX for network security.
5. **Developer-Friendly Tools:** vSphere Kubernetes Service integrates with developer tools like Tanzu Application Platform to simplify the development and deployment of containerized applications.
6. **Scalability and High Availability:** VKS offers features like auto-scaling and fault tolerance to ensure applications remain highly available and performant.
7. **Observability and Monitoring:** vSphere Kubernetes Service integrates with VMware Tanzu Observability (formerly Wavefront) and other monitoring tools to provide real-time insights into cluster and application performance.
8. **Support for Kubernetes Ecosystem:** VKS is fully compliant with upstream Kubernetes, meaning it supports the Kubernetes API and works with Kubernetes-native tools like Helm, Istio, and Prometheus.

Use cases for vSphere Kubernetes Service

1. **Application Modernization:** Organizations can modernize legacy applications by containerizing them and deploying them on Kubernetes clusters managed by VKS.
2. **DevOps Enablement:** VKS supports DevOps workflows by providing automation, CI/CD integration, and developer-friendly tools.
3. **Hybrid Cloud Deployments:** Enterprises can use VKS to deploy Kubernetes clusters across on-premises and cloud environments, enabling consistent operations.

4. **Microservices Architecture:** VKS supports microservices-based application development, allowing teams to build and scale distributed systems.

Getting Started with Storage and vSphere Kubernetes Service

Install a vSphere Kubernetes Service cluster

Install a vSphere Kubernetes Service (VKS) cluster in your VCF 9.1 environment and configure worker nodes with the appropriate storage utilities for your workloads.

About this task

NFS utilities are automatically installed in the worker nodes of the cluster that you create. If you want to create worker nodes with iSCSI or NVMe utilities installed, you must create a custom OVA image and use that image for the worker nodes. The custom image can be created using Docker, and the `vcf` command line tool can then be used to create an OVA file from this image. This OVA file can be uploaded to the Content Library in vSphere. When creating the VKS cluster, this image from the Content Library can be selected for the node pools.

Steps

1. Create the VKS cluster:

   You can create a VKS cluster using either the default worker node image or a custom image that you create. The default worker node image includes NFS utilities pre-installed, while the custom image can include iSCSI and NVMe utilities. The following options are available:

   - **NAS only (default):** Create a VKS cluster using the default worker node image, which includes NFS utilities pre-installed.
   - **SAN enabled (custom image):** Create a custom Docker image that includes iSCSI and NVMe utilities, generate an OVA file using the `vcf` command line tool, upload the OVA to the vSphere Content Library, and then create the VKS cluster selecting that custom image for the node pools.

   ▼ Show instructions for creating a NAS-only vSphere Kubernetes Service cluster with the default worker node image

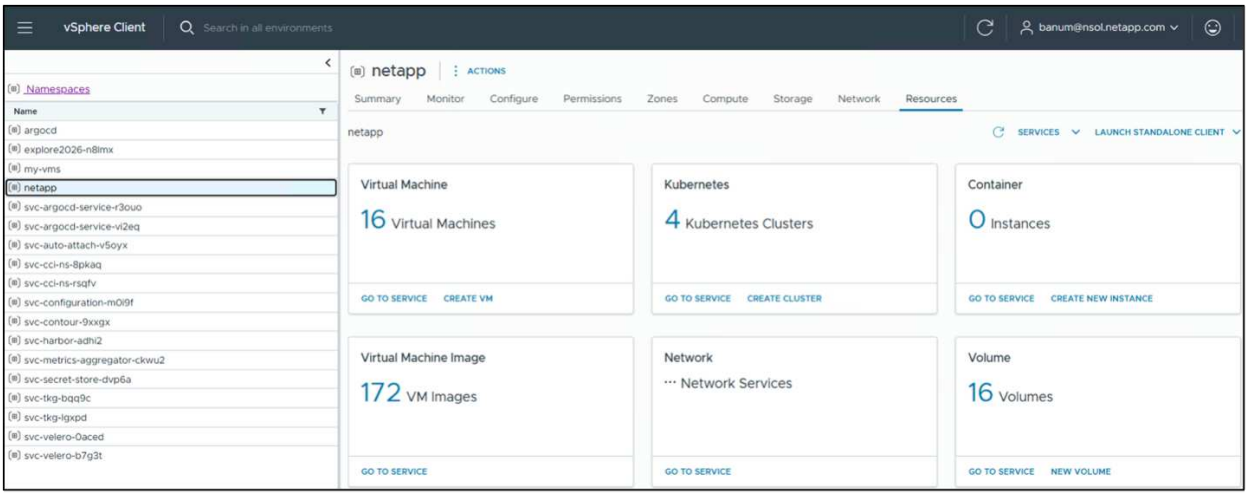
   Create the VKS cluster using the default worker node image. Follow the prompts to specify the cluster name, node pool configuration, and other settings.

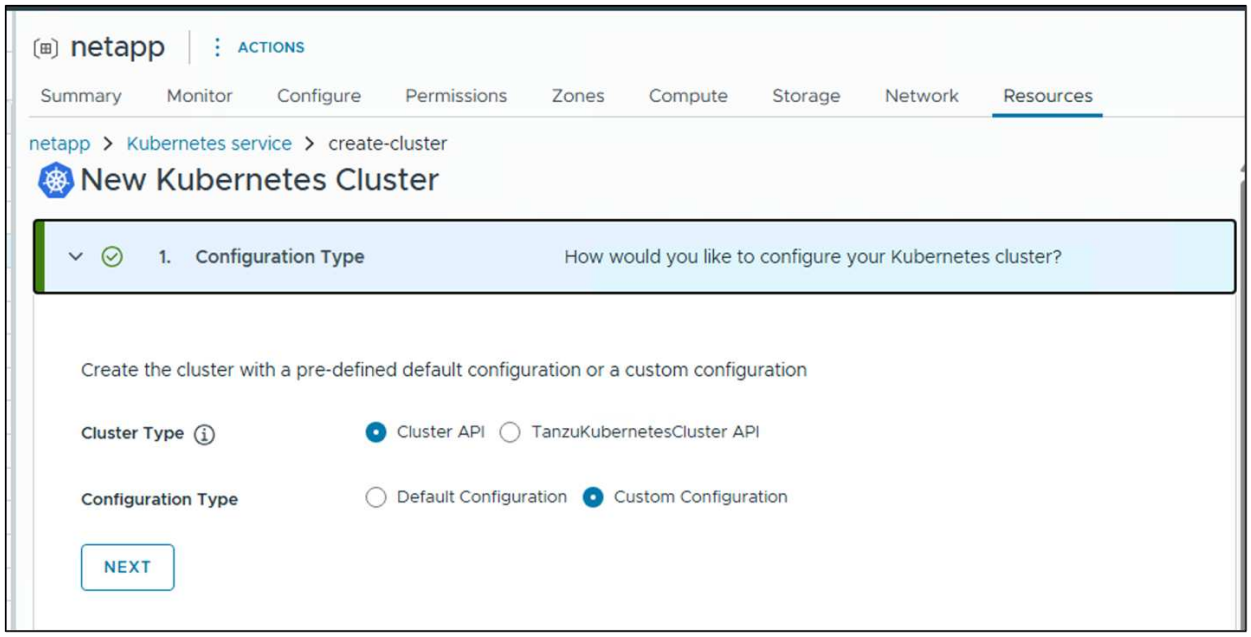
   From the vSphere Client, go to the namespace where you want to create the vSphere Kubernetes cluster. On the **Resources** tab of that namespace, click **Create Cluster** in the Kubernetes section, or click **Go to Service** and then click **Create Cluster**.

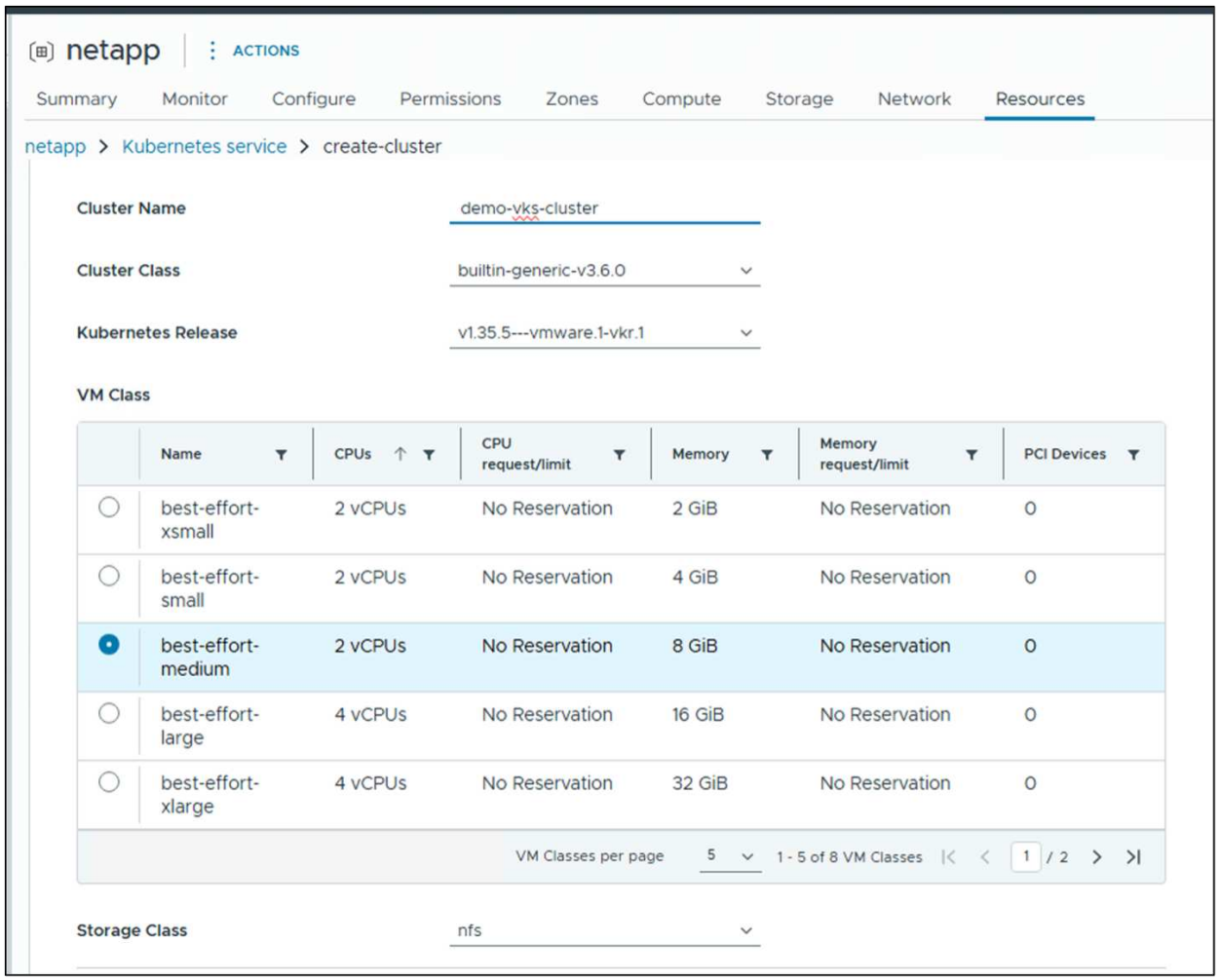
   Complete the form with the cluster details:

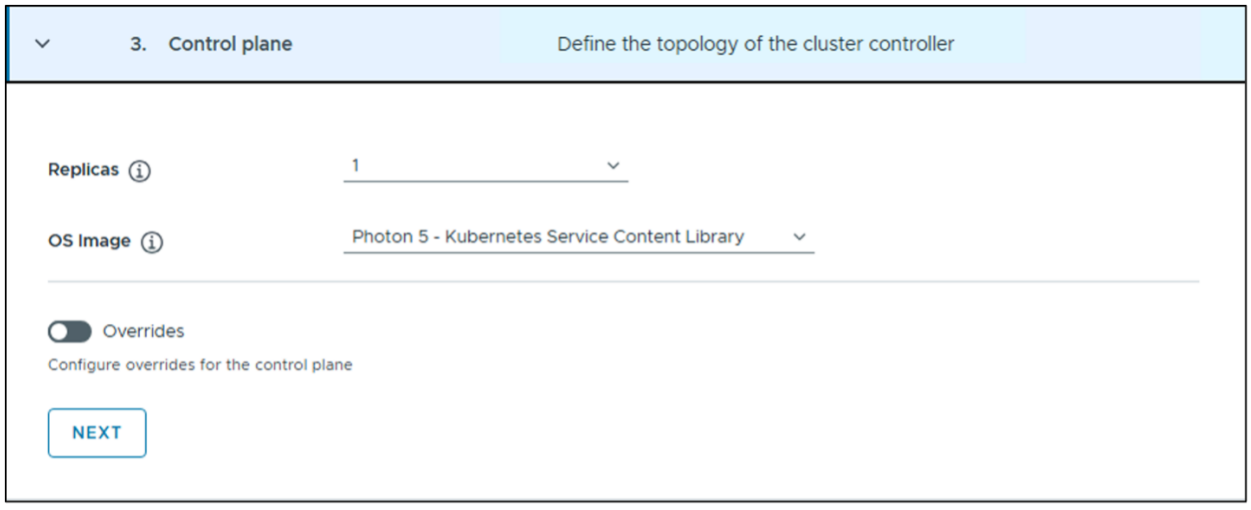
   - In section 1, **Configuration Type**, select **Custom**.
   - In section 2, **General Settings**, provide a name for the cluster and leave all other settings at their defaults.
   - Accept all defaults for section 3.
   - In section 4, **Node Pools**, click the ellipsis (…) next to the node pool and click **Edit**. Increase the replicas to 3 and select the latest Ubuntu version for the OS image. Click **Next**, then **Review and Confirm**.

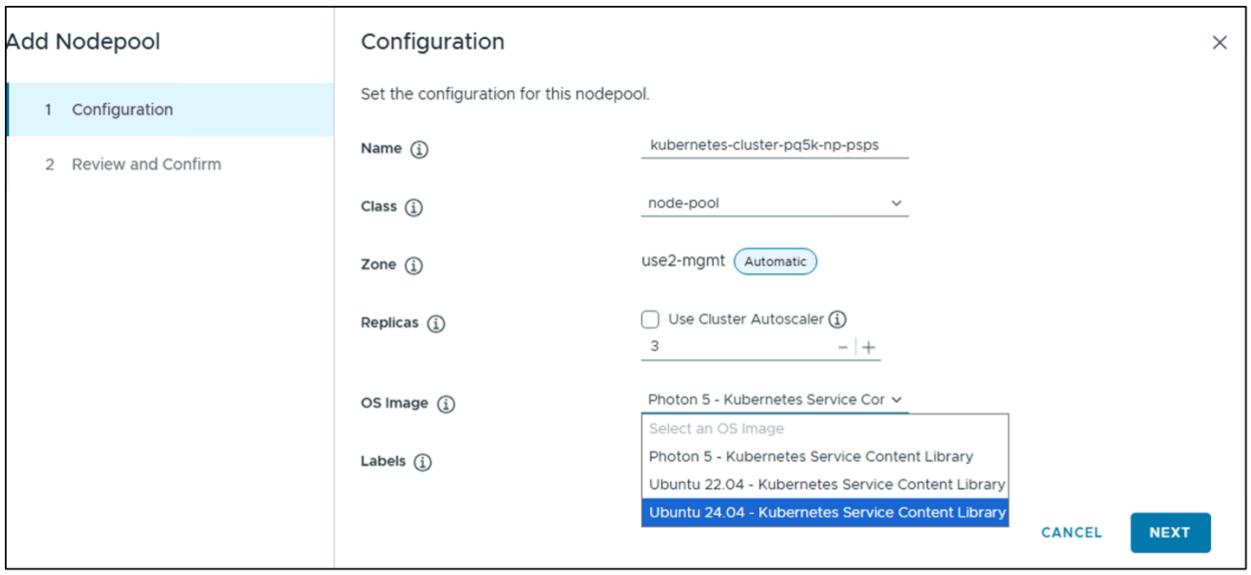
After reviewing the cluster details, click **Finish**. The vSphere Kubernetes cluster will be created in a few minutes.

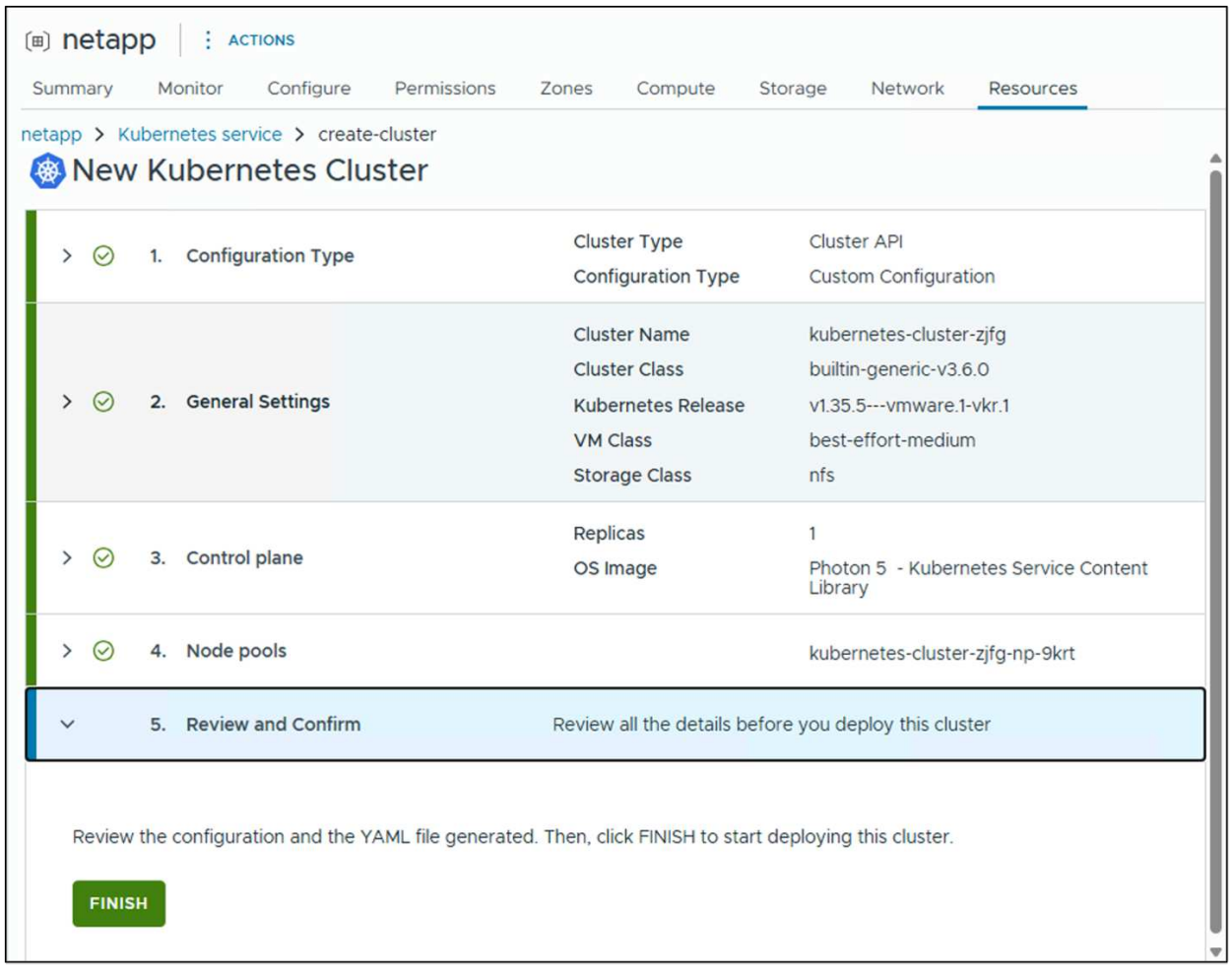


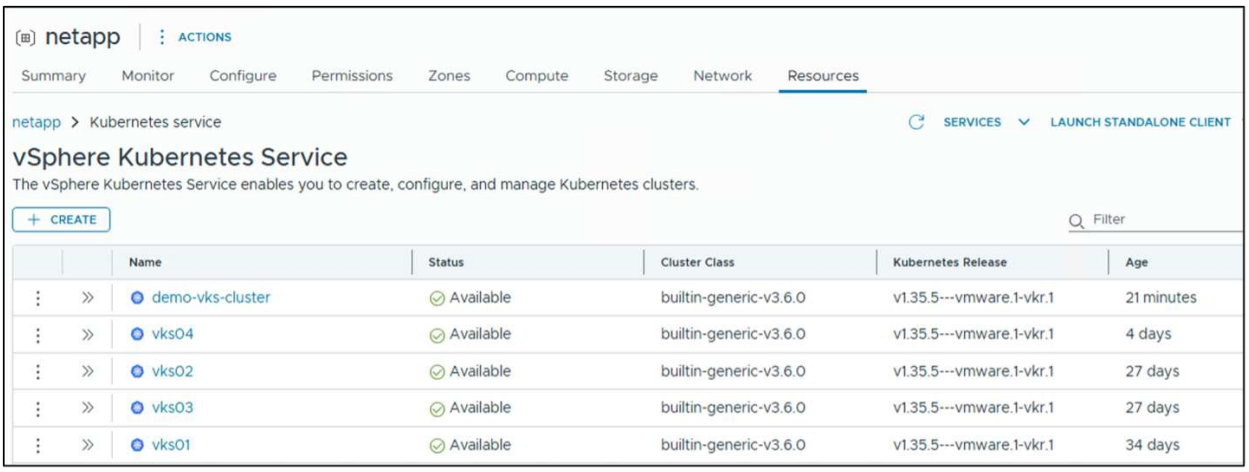



netapp
ACTIONS
Summary
Monitor
Configure
Permissions
Zones
Compute
Storage
Network
Resources
netapp > Kubernetes service > create-cluster
Cluster Name
demo-vks-cluster
Cluster Class
builtin-generic-v3.6.0
Kubernetes Release
v1.35.5---vmware.1-vkr.1
VM Class
Name
CPUs
CPU request/limit
Memory
Memory request/limit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



▼ **Show instructions for creating a custom OVA image for SAN-enabled worker nodes**

Create a Docker image with the required utilities installed. The following example demonstrates creating a Docker image based on Ubuntu 24.04 with iSCSI and Multipathing utilities installed.

```
#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"]
```

Build the Docker image using:

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

If Docker registry is not available, run this command to start a local registry and push the image to it:

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

Tag the image and push it.

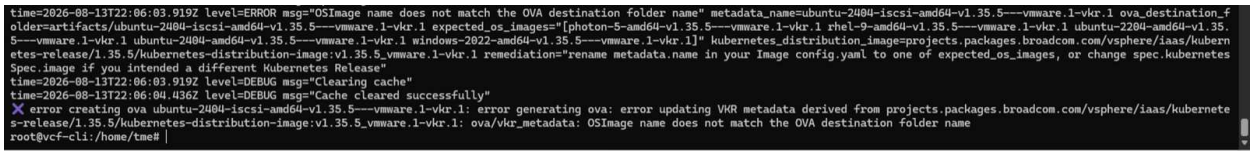
```
docker tag ubuntu-2404:san localhost:5000/ubuntu-2404:san
docker push localhost:5000/ubuntu-2404:san
```

Create the image configuration file (save as `ubuntu2404vkr135-san.yml`) and reference the image:

```
#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
```

Metadata name should be one from the pre-approved list. If not, you will get an error as shown below when you try to create the OVA file. The pre-approved list can be found in the VCF CLI help for the `kr bake` command.

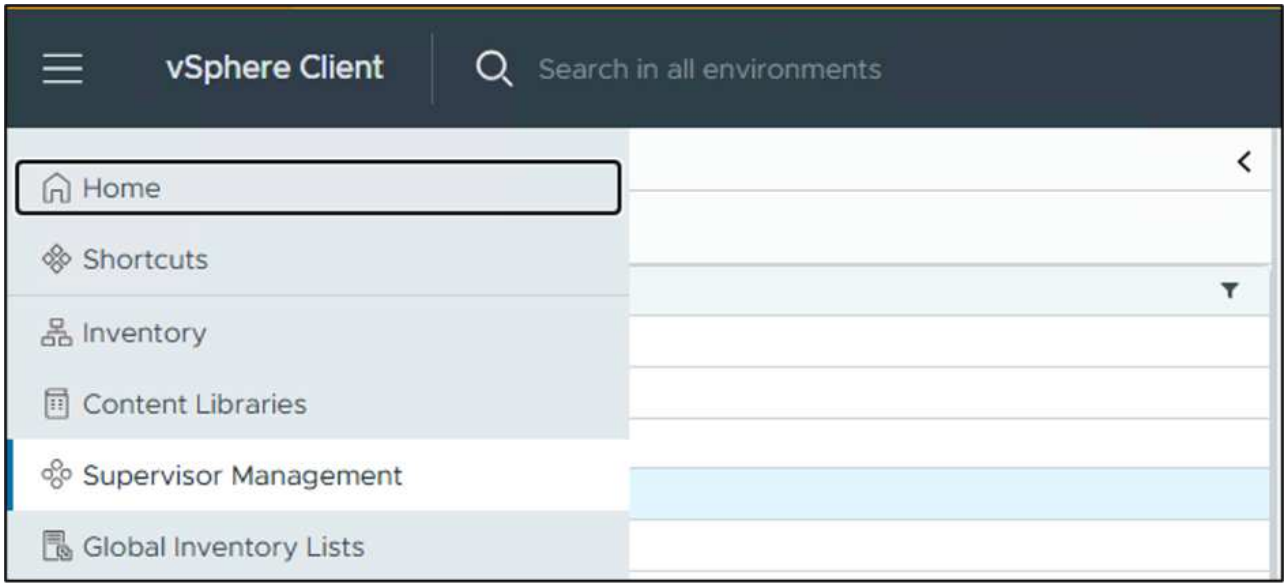


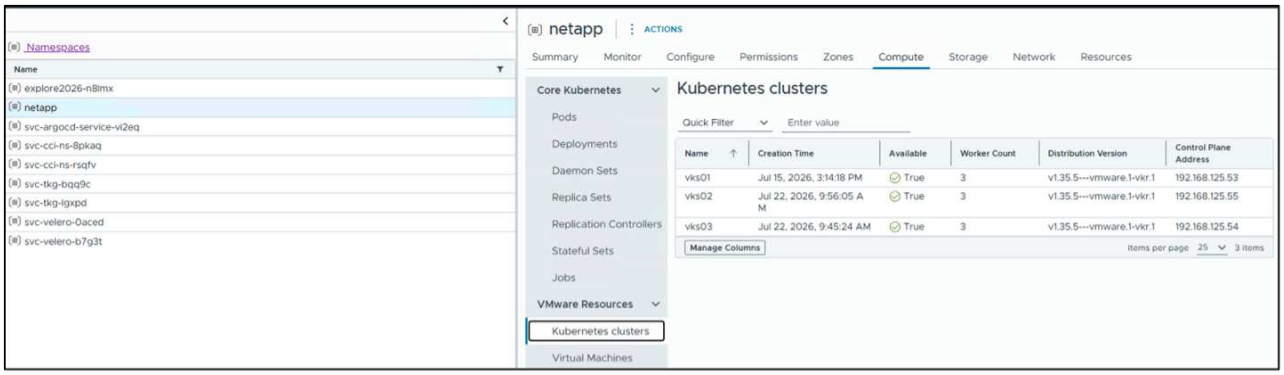
Create the OVA file using the VCF CLI:

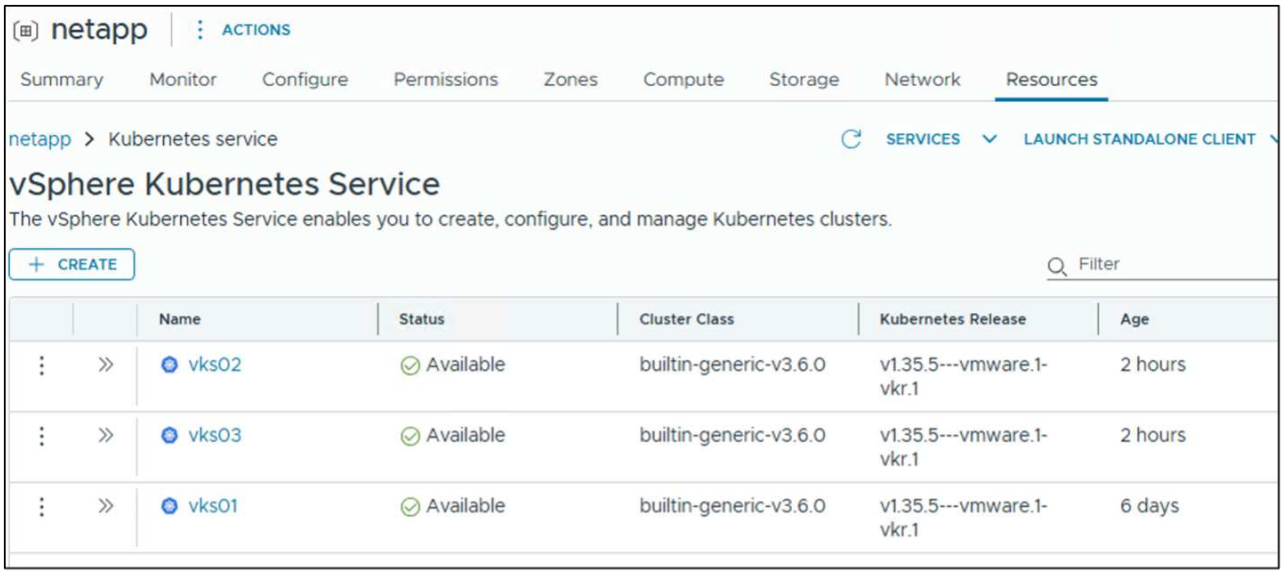
```
vcf kr bake -f <path/to/ubuntu2404vkr135-san.yml> --log-level DEBUG
--log-format plain
```

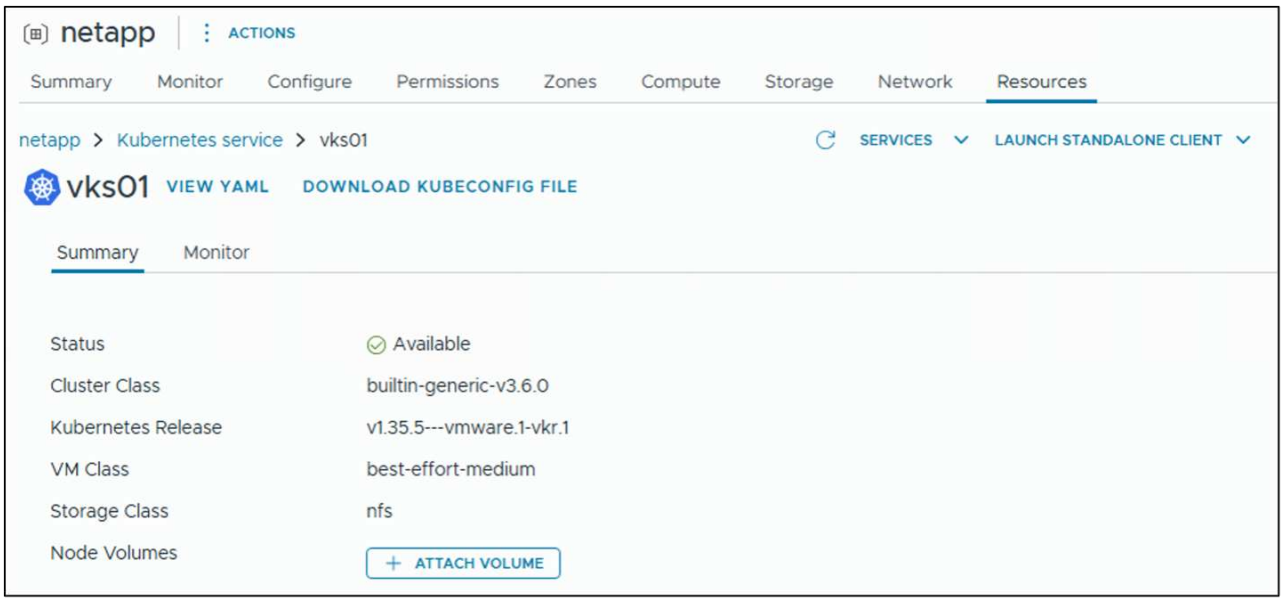
Transfer the OVA file using SCP to a machine from which you can upload it to the vSphere Content Library.

2. After the cluster is installed, locate it in vCenter and download the kubeconfig file.
   a. Click **Supervisor Management** from the main menu and select the namespace where your cluster was created.
   b. Select the **Compute** tab and then **Kubernetes Clusters**. All clusters in the namespace are displayed.
   c. Go to the **Resources** tab, select the Kubernetes cluster you need, and download its kubeconfig file.


vSphere Client
Search in all environments
Home
Shortcuts
Inventory
Content Libraries
Supervisor Management
Global Inventory Lists


Namespaces
Name
explore2026-n8lmx
netapp
svc-argocd-service-vi2eq
svc-cci-ns-8pkaq
svc-cci-ns-rsqfv
svc-tkg-bqq9c
svc-tkg-lgxpd
svc-velero-0aced
svc-velero-b7g3t
netapp
ACTIONS
Summary
Monitor
Configure
Permissions
Zones
Compute
Storage
Network
Resources
Core Kubernetes
Pods
Deployments
Daemon Sets
Replica Sets
Replication Controllers
Stateful Sets
Jobs
VMware Resources
Kubernetes clusters
Virtual Machines
Kubernetes clusters
Quick Filter
Enter value
Name
Creation Time
Available
Worker Count
Distribution Version
Control Plane Address
vks01
Jul 15, 2026, 3:14:18 PM
True
3
v1.35.5---vmware.1-vkr.1
192.168.125.53
vks02
Jul 22, 2026, 9:56:05 AM
True
3
v1.35.5---vmware.1-vkr.1
192.168.125.55
vks03
Jul 22, 2026, 9:45:24 AM
True
3
v1.35.5---vmware.1-vkr.1
192.168.125.54
Manage Columns
Items per page
25
3 items


netapp
ACTIONS
Summary
Monitor
Configure
Permissions
Zones
Compute
Storage
Network
Resources
netapp > Kubernetes service
SERVICES
LAUNCH STANDALONE CLIENT
vSphere Kubernetes Service
The vSphere Kubernetes Service enables you to create, configure, and manage Kubernetes clusters.
CREATE
Filter
Name
Status
Cluster Class
Kubernetes Release
Age
vks02
Available
builtin-generic-v3.6.0
v1.35.5---vmware.1-vkr.1
2 hours
vks03
Available
builtin-generic-v3.6.0
v1.35.5---vmware.1-vkr.1
2 hours
vks01
Available
builtin-generic-v3.6.0
v1.35.5---vmware.1-vkr.1
6 days



3. From a bastion host, connect to the cluster using the kubeconfig file to manage it from the CLI.

```
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
```

Video demonstration

The following video demonstrates creating a vSphere Kubernetes cluster.

Create a vSphere Kubernetes Cluster on the Supervisor Cluster

Learn about NetApp storage for vSphere Kubernetes Service

Using NetApp as storage for vSphere Kubernetes Service (VKS) is a powerful combination that leverages NetApp's robust storage solutions to enhance the performance, scalability, and reliability of Kubernetes workloads. NetApp Trident, an open-source dynamic storage provisioner for Kubernetes, is used for integrating storage with workloads on vSphere Kubernetes Service.

Key benefits of using NetApp storage with VKS

1. **Dynamic Storage Provisioning:** NetApp Trident allows for dynamic provisioning of storage volumes, enabling Kubernetes pods to request storage on-the-fly based on their requirements.
2. **Persistent Storage:** NetApp solutions provide persistent storage capabilities, ensuring data durability and availability across pod restarts and failures.
3. **High Performance:** NetApp's storage solutions, such as ONTAP, offer high-performance storage with low latency, which is ideal for I/O-intensive applications running on Kubernetes.
4. **Scalability:** NetApp storage systems can scale to meet the demands of growing Kubernetes workloads, supporting large-scale deployments.

5. **Data Management:** NetApp provides advanced data management features, including snapshots, cloning, and data replication, which can be beneficial for backup, disaster recovery, and development workflows.
6. **Multi-Cloud and Hybrid Cloud Support:** NetApp's storage solutions can be deployed across on-premises, public cloud, and hybrid cloud environments, providing flexibility and consistency in storage management.

NetApp ONTAP integration overview

NetApp ONTAP provides enterprise-grade storage with features like data deduplication, compression, and encryption. You use NetApp Trident to integrate NetApp storage with Kubernetes. NetApp Trident supports various NetApp storage platforms, including ONTAP on-premises, FSx for NetApp ONTAP in AWS, Azure NetApp Files in Azure, and Google Cloud NetApp Volumes in Google Cloud.

You use Trident Protect to handle data protection and disaster recovery for your Kubernetes workloads. Trident Protect allows you to create snapshots, clones, and replicate data across different storage backends, ensuring that your data is safe and recoverable in case of failures.

Key features of Trident

CSI Compliance
Ensuring compatibility with the latest Kubernetes storage features and standards.
Declarative Configuration
Allowing users to define storage requirements in YAML files, which are then automatically managed by Trident.
Drivers
Trident supports drivers for NFS, CIFS/SMB, iSCSI, FC and NVMe/TCP protocols supported by ONTAP.
Hybrid and multi-cloud support
Trident supports drivers for ONTAP storage on-premises and managed storage service in the hyperscalers namely FSx for NetApp ONTAP in AWS, Azure NetApp Files in Azure and Google Cloud NetApp Volumes in Google Cloud.
Advanced Data Management
Leveraging ONTAP's snapshot and cloning capabilities for efficient data protection and rapid provisioning of test and development environments.

For complete details on Trident, refer to the Trident documentation.

Trident Protect

Trident Protect is installed separately from Trident. Before deployment, verify that your Kubernetes platform, storage backend, snapshot components, and object storage meet the Trident Protect requirements.

Key features of Trident Protect

Automated backups
Trident Protect enables automated, policy-driven backups of Kubernetes applications, ensuring that they are regularly backed up without manual intervention.

Snapshot Management
It leverages ONTAP's snapshot capabilities to create frequent, point-in-time copies of container applications and VMs, providing a reliable mechanism for recovery.

Backup repository encryption
Trident Protect supports password-based encryption for Restic and Kopia backup repositories. Configure persistent-volume and in-transit encryption separately in the storage and network layers.

Compliance and Auditing
Provides tools and features that help organizations meet compliance requirements and conduct regular audits of their data protection practices.

Disaster Recovery
Integrates with ONTAP's replication features to provide robust disaster recovery solutions, ensuring business continuity even in the face of catastrophic events.

For complete details on Trident Protect, refer to the Trident Protect documentation.

NetApp ONTAP hardware models and protocols supported

NetApp ONTAP software runs on a variety of hardware models, each designed to cater to different performance and capacity requirements. Trident extends its robust capabilities to support various NetApp ONTAP systems, including All Flash FAS (AFF), All SAN Array (ASA), and ASA R2 systems. This support ensures that organizations can leverage the full potential of their high-performance storage solutions in containerized environments.

All Flash FAS (AFF) Systems
AFF systems deliver exceptional performance and low latency, making them ideal for high-demand applications.

All SAN Array (ASA) Systems
ASA systems are designed for dedicated SAN environments, offering robust performance and reliability.

ASA R2 Systems
ASA R2 systems provide enhanced features and capabilities for SAN environments.

AFX
NetApp AFX is a purpose-built, disaggregated, all-flash storage architecture designed for enterprise AI workloads. It offers high-performance NAS, allowing independent scaling of compute and capacity. NetApp AFX storage systems differ from other ONTAP-based systems (ASA, AFF, and FAS) in the implementation of their storage layer. AFX uses a distributed file system that allows for high performance and scalability, while other ONTAP-based systems use a traditional storage architecture.

1. Protocols supported for AFF: NFS, CIFS/SMB, iSCSI, FC, NVMe/TCP
2. Protocols supported for ASA: iSCSI, FC, NVMe/TCP
3. Protocols supported for ASA R2: iSCSI, FC, NVMe/TCP
4. Protocols supported for AFX: Starting with Trident 25.10, only NFS protocol is supported; SMB protocol is not supported.

Trident drivers for ONTAP storage

Trident includes the following CSI drivers, each capable of provisioning a volume in the backend storage.

For NAS protocols on supported hardware models

1. ontap-nas: One PVC maps to one PV, which is a dedicated FlexVol volume.
2. ontap-nas-economy: Each PV provisioned is a qtree, with a configurable number of qtrees per FlexVol volume (default is 200, configurable between 50 and 300).

   One PVC maps to one PV, which is a qtree on a shared FlexVol volume.

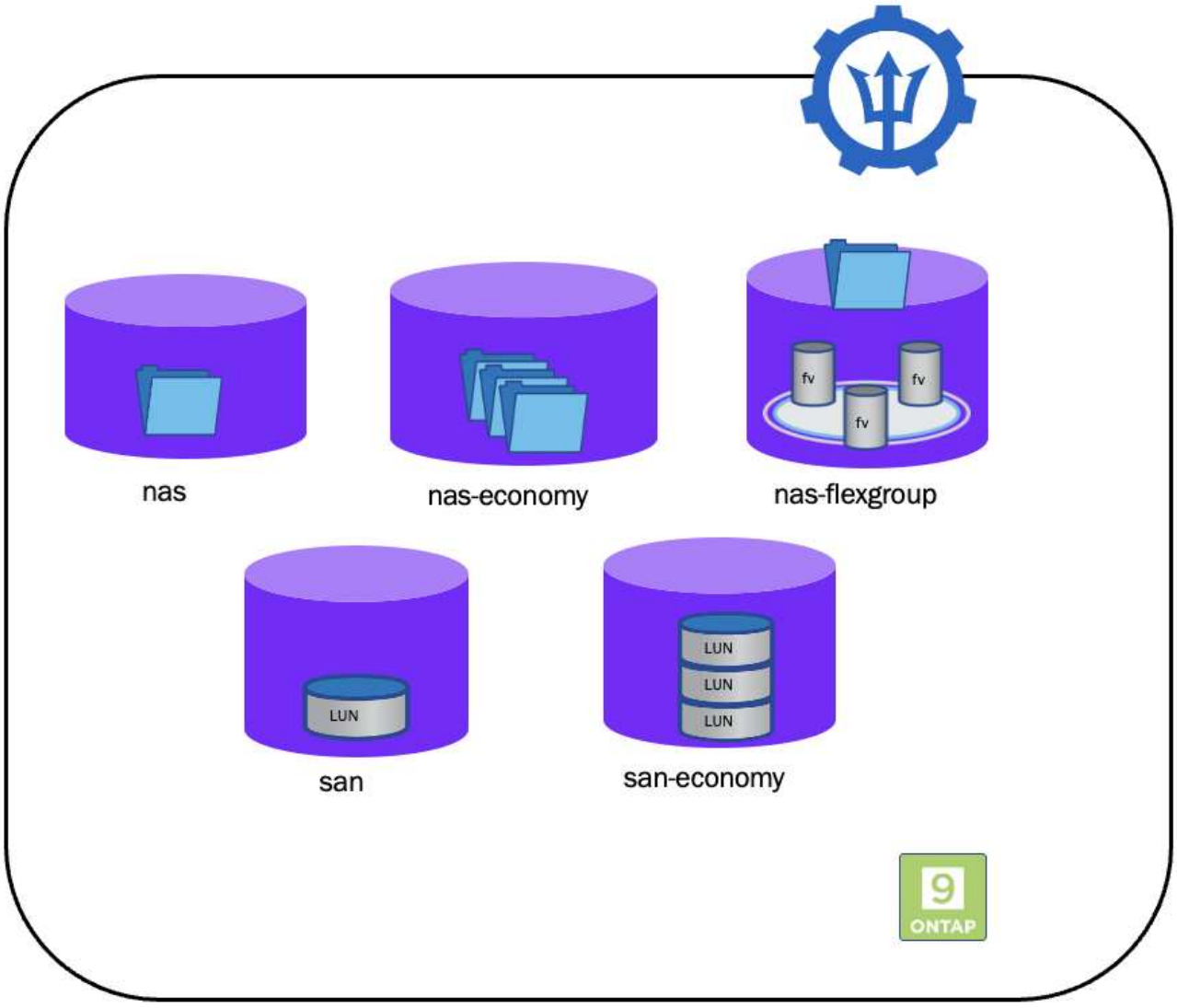
You can place all the disks of the VMs as qtrees in a single volume. However, please be aware that Snapshots, Clones and Replication features are not supported by Trident. When those features are managed by the storage administrator, they are not PV granular.

1. ontap-nas-flexgroup: Each PV provisioned is a full ONTAP FlexGroup, and all aggregates assigned to a SVM are used.

For SAN protocols on supported hardware models

1. ontap-san: One PVC maps to one PV, which is a LUN on a dedicated FlexVol volume.
2. ontap-san-economy: One PVC maps to one PV, which is a LUN on a shared FlexVol volume. You can have multiple LUNs in the shared FlexVol volume (default is 100, configurable between 50 and 200 LUNs/PVs per FlexVol volume).

You can place all the disks of the VMs as LUNs in a single volume. However, this driver supports Snapshots and Clones but does not support Replication. When Replication is managed by storage administrator, it is not PV granular.



For the complete list of capabilities exposed through Trident drivers and those that must be applied by the

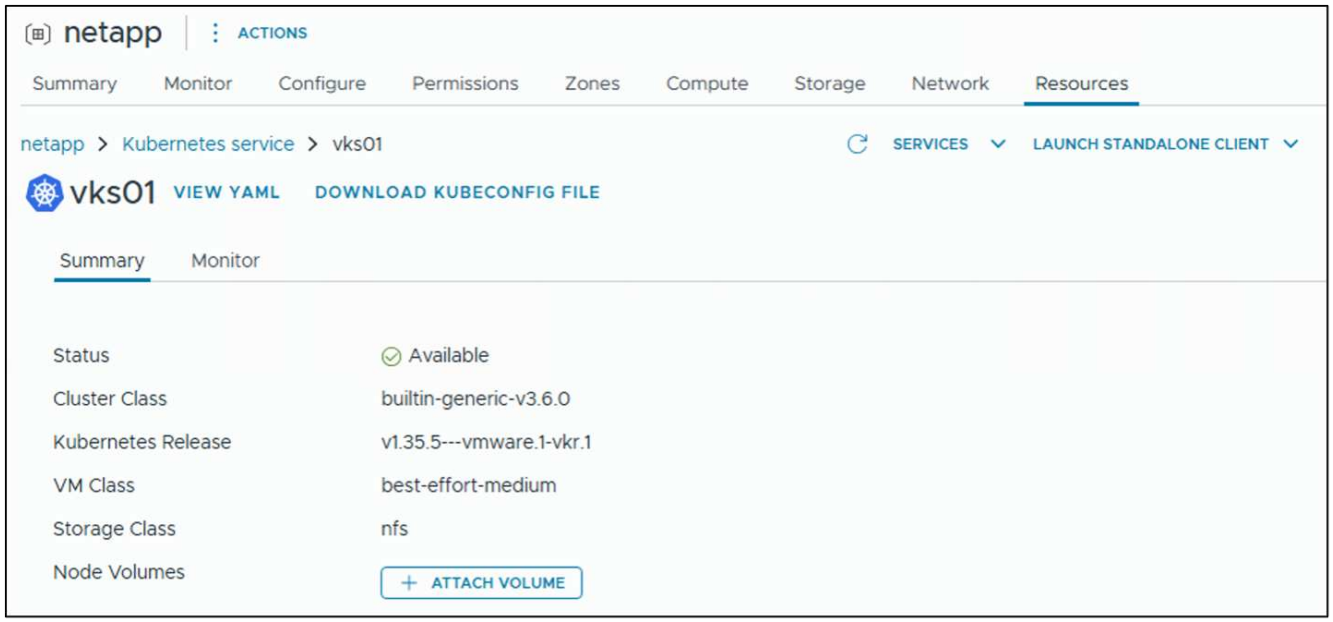
storage administrator, see ONTAP backend drivers section in the Trident documentation.

Install NetApp Trident in a VKS cluster

Install NetApp Trident as the dynamic storage provisioner in your vSphere Kubernetes Service cluster to integrate NetApp ONTAP storage with your Kubernetes workloads.

Before you begin

- Ensure your ONTAP cluster is configured and connected to your VMware Kubernetes environment.
- Ensure your VKS cluster has the iSCSI or NVMe tools installed if your workloads will be using those protocols for storage.
- Ensure you have `kubectl` installed on a machine from which you can connect to the VKS cluster using the kubeconfig file.



Steps

1. Install the Trident Operator using a Helm chart by following the instructions in the Trident documentation:

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

   ▼ Show example

   ```
   helm install trident netapp-trident/trident-operator --version
   100.2606.0 --create-namespace --namespace trident
   --kubeconfig=/path/to/your-kubeconfig-file
   ```

   To enable debug logging, add `--set tridentDebug=true`:

   ```
   helm install trident netapp-trident/trident-operator --version
   100.2606.0 --create-namespace --namespace trident --set
   tridentDebug=true --kubeconfig=/path/to/your-kubeconfig-file
   ```

You can also install Trident manually using the Trident Operator. Review the procedures in the Trident documentation: Manually deploy the Trident Operator

2. Confirm that all Trident pods are running in the cluster.

▼ **Show example**

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

The Trident pods may not get created at this point. If you describe the ReplicaSet object, you might see an error about violating PodSecurity as shown below. This error occurs because the `trident` namespace created during installation enforces the Kubernetes restricted Pod Security Standard (PSS). The Trident Operator requires elevated system privileges to manage host storage and cannot run under the rigid constraints of a restricted profile. To fix this, grant the `trident` namespace a privileged profile so the ReplicaSet controller can successfully spin up the pods.

```
Warning  FailedCreate  25s (x5 over 64s)  replicaset-controller
(combined from similar events): Error creating: pods "trident-
operator-5cbf7f857-z7ztd" is forbidden: violates PodSecurity
"restricted:latest": allowPrivilegeEscalation != false (container
"trident-operator" must set
securityContext.allowPrivilegeEscalation=false), unrestricted
capabilities (container "trident-operator" must set
securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod
or container "trident-operator" must set
securityContext.runAsNonRoot=true), seccompProfile (pod or container
"trident-operator" must set securityContext.seccompProfile.type to
"RuntimeDefault" or "Localhost")
```

Apply the built-in Kubernetes Pod Security Admission labels to the `trident` namespace to exempt it from the restricted profile constraints:

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

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

```
vksbastion@vks:~/vks02$  kubectl get pods -n trident --kubeconfig=vks02-kubeconfig.yaml
NAME                                  READY   STATUS    RESTARTS   AGE
trident-controller-84576f9bcf-vz622   6/6     Running   0          4d
trident-node-linux-24d47              2/2     Running   0          4d
trident-node-linux-52ccn              2/2     Running   0          4d
trident-node-linux-5bnv9              2/2     Running   0          4d
trident-node-linux-8hgnb              2/2     Running   0          4d
trident-operator-5cbf7f857-kskcx      1/1     Running   0          4d
vksbastion@vks:~/vks02$
```

Prepare storage backend and StorageClass configuration files for your environment

1. Configure a Trident backend for ONTAP by defining the necessary connection details and storage parameters.
2. Define storage classes in Kubernetes that map to NetApp storage backends. These classes specify parameters like performance characteristics and replication policies.

   Define Trident backends and storage classes for the following protocols as required by your workloads:

   - NAS (ontap-nas driver)
   - NAS Economy (ontap-nas-economy driver)
   - SAN (ontap-san driver) for iSCSI, FC, or NVMe protocols
   - SAN Economy (ontap-san-economy driver) for iSCSI

   NAS

   Create a TridentBackendConfig and StorageClass for ONTAP NAS to enable NFS-based persistent storage provisioning. The backend configuration includes credentials stored in a Kubernetes Secret and references your ONTAP SVM and management LIF.

   Sample backend secret and backend configuration file using username and password authentication (save as `tbc-nas.yaml`):

```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
```

Sample backend secret and backend configuration file using client certificate authentication (save as `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 (save as `sc-nas.yaml`):

`mountOptions` is an optional parameter that specifies additional mount options for NFS volumes. The `nconnect` option allows multiple parallel TCP connections for a single NFS mount, which can improve performance for high-throughput workloads. The default value is 1, but it can be increased up to the maximum allowed by the ONTAP system.

ONTAP supports up to 16 connections for a single NFS mount on an nconnect-capable client. Trident passes the mount options directly to the Kubernetes worker nodes, so choose a value supported by both the worker-node NFS client and ONTAP; the following example uses four connections.

```
# 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
```

NAS Economy

Create a TridentBackendConfig and StorageClass for the ONTAP NAS Economy driver to enable NFS-based persistent storage provisioning using qtrees. The backend configuration includes credentials stored in a Kubernetes Secret and references your ONTAP SVM and management LIF.

Sample backend secret and backend configuration file using username and password authentication (save as `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 (save as `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
```

iSCSI SAN

Create a TridentBackendConfig and StorageClass for ONTAP SAN with iSCSI to enable iSCSI-based block storage provisioning. The backend configuration uses the ontap-san driver and includes credentials stored in a Kubernetes Secret. The `sanType` parameter defaults to the iSCSI protocol if omitted.

Backend secret and backend configuration file using username and password authentication (save as `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 secret and backend configuration file using client certificate authentication (save as `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 (save as `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
```

NVMe SAN

Create a TridentBackendConfig and StorageClass for ONTAP SAN with NVMe to enable NVMe-based block storage provisioning. The backend configuration uses the ontap-san driver and includes credentials stored in a Kubernetes Secret. The `sanType` parameter must be set to `nvme`.

Backend secret and backend configuration file using username and password authentication (save as `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 secret and backend configuration file using client certificate authentication (save as `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 (save as `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
```

Review the Trident documentation for additional samples of YAML files and information on access modes and volume modes supported by SAN drivers and NAS drivers.

- Samples using NAS drivers
- Samples using SAN drivers

Create a VolumeSnapshotClass configuration file

Create a VolumeSnapshotClass definition. This configuration enables snapshot-based operations for persistent volumes.

VolumeSnapshotClass definition (save as `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
```

Apply the configuration files to your cluster

Apply the configuration files you created in the previous steps to the vSphere Kubernetes Service cluster using `kubectl`. This creates the necessary secrets, backend configurations, storage classes, and snapshot class in your cluster.

Steps

1. Apply the TridentBackendConfig and StorageClass files for each protocol you configured.

```
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. Verify that the resources were created successfully.

   Check TridentBackendConfig objects:

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

   Check StorageClass objects:

   ```
   kubectl get storageclass
   ```

   Check VolumeSnapshotClass:

   ```
   kubectl get volumesnapshotclass
   ```

Set the default Trident storage and snapshot classes

Set the Trident StorageClass and VolumeSnapshotClass as the defaults in the vSphere Kubernetes Service cluster.

Steps

1. Set the default Trident StorageClass.

   Set a Trident-backed StorageClass as the cluster default so that PersistentVolumeClaims automatically use it when no storage class is specified.

   Ensure that only one StorageClass is set as default. If another StorageClass is already set to default, set its annotation to `false`.

   From the CLI:

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

2. Set the default Trident VolumeSnapshotClass.

   Set a Trident-backed VolumeSnapshotClass as the cluster default to enable snapshot-based operations for persistent volumes. This ensures that VolumeSnapshots automatically use the Trident CSI driver when no snapshot class is specified.

   Ensure that only one VolumeSnapshotClass is set as default. If another VolumeSnapshotClass is already set to default, set its annotation to `false`.

   From the CLI:

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

Use Kubernetes Persistent Volume Claims to request storage for workloads

Trident automatically provisions the necessary volumes from the backend NetApp storage. Refer to Deploy workloads with persistent storage for examples of deploying a workload with PVCs in a VKS cluster.

Deploy a container application with NetApp persistent storage

Deploy a containerized application in your vSphere Kubernetes Service cluster using NetApp ONTAP persistent storage. The following examples demonstrate deploying a PostgreSQL database using either NAS (ontap-nas) or SAN iSCSI (ontap-san) storage.

The following YAML configuration sets POSTGRES_USER / POSTGRES_PASSWORD directly in the manifest. In production, it is recommended to use Kubernetes Secrets to manage sensitive information like database credentials.

Deploy PostgreSQL with NAS storage (ontap-nas driver)

You can deploy a PostgreSQL container application backed by an NFS persistent volume provisioned by the ontap-nas driver. The following example demonstrates how to create a PersistentVolumeClaim (PVC) and a Deployment for PostgreSQL.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pvc
spec:
  storageClassName: sc-nas # Specify the StorageClass for dynamic
provisioning
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

```yaml
---
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
```

Deploy PostgreSQL with SAN storage (ontap-san iSCSI driver)

Use the following YAML to deploy a PostgreSQL container application backed by an iSCSI persistent volume provisioned by the ontap-san driver. The `Recreate` deployment strategy ensures the pod is fully terminated before a new one starts, which is required for ReadWriteOnce iSCSI volumes and prevents data corruption across pod restarts. This configuration supports data persistence across pod deletions and recreations, and is compatible with Trident volume snapshots and backup and restore operations.

```
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
```

Validate the database deployment

After deploying the application, verify that the pods are in the running state and that the database is operational. Use the following commands to check the status of the pods, PVCs, and services. Ensure that the pods are running and the PVCs are bound to the appropriate volumes.

```
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:~$
```

Depending on the protocol used, the backend storage is a volume, qtree, or LUN. You can verify the storage provisioned in the ONTAP system by describing the PersistentVolume (PV) to retrieve the internal volume name and then using it in ONTAP CLI commands to get the storage details. Use the following command to

describe the PV and retrieve the internal volume name:

```
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>
```

Figure 1. Show Example

```
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>
```

Copy the internal volume name from the output and run the following command to get the storage details from the ONTAP CLI.

For 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
```

For SAN LUNs, run the following command to get the LUN details from the ONTAP CLI:

```
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::>
```

If you used `nconnect` mount options in the NAS or NAS Economy storage class, you can verify the number of active TCP connections from the worker node to the storage server.

First, list the Trident backend configurations to find the backend name, then describe it to retrieve the vserver name and data LIF:

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

Then log into the ONTAP cluster and run the following command, substituting the data LIF from the output above:

```
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.
```

Figure 2. Show Example

After the pods are in the running state, connect to the database and verify data operations.

Steps

1. Forward the PostgreSQL service port to your local machine:

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

2. From a different terminal, connect to the database using psql:

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

3. Create a database and a table, then populate the table with data:

```
CREATE DATABASE employee;
```

Switch to the new database (psql meta-command):

```
\c employee
```

Create the table, insert a row, and query the results:

```
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;
```

Video demonstration

The following video demonstrates installing Trident on a vSphere Kubernetes cluster and deploying a PostgreSQL database with persistent storage using Trident.

Deploying workloads with ONTAP persistent storage using Trident

Copyright information

Trademark information