

How to Deploy Milvus Vector Database on Vultr Kubernetes Engine

Learn how to deploy Milvus vector database on Vultr Kubernetes Engine with our step-by-step guide. Perfect for AI applications and vector similarity search.

Contents

01	Introduction	3
02	Prerequisites	3
03	Deploy a Milvus Operator	4
04	Install a Milvus Cluster	6
05	Enable External Access to MilvusDB	10
06	Scale the Milvus Cluster	13
07	Verify MilvusDB Block Storage Resources	18
08	Troubleshooting	20
09	Conclusion	21
010	Next Steps	21

Introduction

Milvus is an open-source, distributed database designed to efficiently manage vector data. Its actual real-world use cases include integration with applications such as recommendation systems, Natural Language Processing (NLP), image and video retrieval. By leveraging vector data, Milvus DB is a crucial component for systems that rely on similarity-based search and retrieval of high-dimensional data points.

To scale a Milvus database system on your Vultr Kubernetes Engine (VKE) cluster, install a Milvus operator that enables the deployment and management of the Milvus cluster.

This article explains how to install Milvus DB on a Vultr Kubernetes Engine (VKE) cluster. You are to deploy a Milvus operator and install a Milvus cluster. Then, you will apply custom resource definitions to set up external access to the database service and test the external service by creating a new database.

Prerequisites

Before you start:

- [Deploy a Vultr Kubernetes Engine \(VKE\) Optimized Cloud Compute cluster](#) with at least:
 - 5 nodes
 - 4GB RAM
 - 2 vCPUs
- [Deploy a Vultr Docker Server](#) to use as the management machine
- Using [SSH](#), access the server as a [non-root user](#)
- [Install and configure Kubectl](#) to access the VKE Cluster

Deploy a Milvus Operator

You can install a Milvus Operator to a VKE cluster in two ways. You can either use a helm chart or Kubectl. In this section, deploy a Milvus Operator to your cluster using Kubectl. Then, change the PVC storage class and install Cert-Manager to handle the process of renewing and obtaining cluster SSL certificates as described in the steps below.

1. Milvus DB requires Vultr high performance PVCs to work well. Disable the default cluster Vultr Block Storage HDD annotation class

```
$ kubectl patch storageclass vultr-block-storage-hdd -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

2. Enable the Vultr Block Storage high-performance class

```
$ kubectl patch storageclass vultr-block-storage -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

When successful, your output should look like the one below:

```
storageclass.storage.k8s.io/vultr-block-storage-hdd patched
storageclass.storage.k8s.io/vultr-block-storage patched
```

3. Install the latest Cert-Manager version

```
$ kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.5.3/cert-manager.yaml
```

The above command installs version `1.5.3`, visit the Cert-Manager releases page to verify the latest version

Output:

```
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.io
```

```
created
  customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manager.io
created
  customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manager.io
created
  customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io created
  customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.io
created
  namespace/cert-manager created
.....
```

4. View the Cert-Manager pods to verify that the installation is successful

```
$ kubectl get pods -n cert-manager
```

Output:

NAME	READY	STATUS	RESTARTS	AGE
cert-manager-679895f5-8wxg9	1/1	Running	0	47s
cert-manager-cainjector-b78544fd4-l86s8	1/1	Running	0	47s
cert-manager-webhook-cd8bbbd67-ktjg2	1/1	Running	0	47s

5. Install the Milvus operator

```
$ kubectl apply -f https://raw.githubusercontent.com/zilliztech/milvus-operator/main/deploy/manifests/deployment.yaml
```

Output:

```
namespace/milvus-operator created
serviceaccount/milvus-operator-checker created
serviceaccount/milvus-operator created
customresourcedefinition.apiextensions.k8s.io/milvusclusters.milvus.io created
customresourcedefinition.apiextensions.k8s.io/milvuses.milvus.io created
customresourcedefinition.apiextensions.k8s.io/milvusupgrades.milvus.io created
clusterrole.rbac.authorization.k8s.io/milvus-operator-checker-role created
clusterrole.rbac.authorization.k8s.io/milvus-operator-manager-role created
```

6. Verify that the Milvus operator is ready and running in your cluster

```
$ kubectl get pods -n milvus-operator
```

Output:

NAME	READY	STATUS	RESTARTS	AGE
milvus-operator-7f48446758-csq5f	1/1	Running	0	84s
milvus-operator-checker-nh54m	0/1	Completed	0	84s

Install a Milvus Cluster

Milvus Custom Resource Definitions (CRDs) allow you to create custom resources and schemas. When installed, CRDs can be managed using operators. In this section, install the Milvus cluster by defining Milvus CRDs, apply the cluster configuration, and verify that the cluster pods are running as described below.

1. Using a text editor such as `nano`, edit the `milvus_cluster.yaml` file

```
$ nano milvus_cluster.yaml
```

2. Add the following configurations to the file

```
apiVersion: milvus.io/v1beta1
kind: Milvus
metadata:
  name: milvus-cluster
  labels:
    app: milvus
spec:
  mode: cluster
  dependencies: {}
  components:
    mixCoord:
      replicas: 1
    dataNode:
      replicas: 1
    indexNode:
      replicas: 1
```

```
queryNode:
  replicas: 1
config:
  common:
    security:
      authorizationEnabled: true
```

Save and close the file.

In the above configuration, no dependencies are defined as Milvus relies on default dependencies such as **etcd**, **minio**, and **pulsar**. Below is what each dependency does:

- **etcd**: Supports Milvus's metadata storage and access
- **minio**: MinIO and S3 are supported as storage engines for data persistence
- **pulsar**: Supports the Milvus cluster's reliable storage and the publication/subscription of message streams

Below is the function of each defined Milvus component:

- **mixCoord**: Includes coordinators such as query coordinator, index coordinator, root coordinator, and data coordinator. These are responsible for load-balancing, maintaining index metadata, and maintaining background data operations
- **dataNode**: Retrieves incremental log data, converts it to log snapshots, and stores it to a Milvus block storage volume
- **indexNode**: Builds Vector indexes
- **queryNode**: Performs a hybrid search of vector and scalar data
- **authorizationEnabled**: When set to **true**, it allows you to apply and control user access to the database to increase security and reliability

3. Apply the Milvus cluster configuration

```
$ kubectl apply -f milvus_cluster.yaml
```

Output:

```
milvus.milvus.io/milvus-cluster created
```

4. Wait for at least 10 minutes and verify the Milvus cluster status

```
$ kubectl get milvus milvus-cluster -o yaml
```

When successful, your output should look like the one below:

```
- lastTransitionTime: "2023-10-12T19:52:42Z"
  message: Milvus components are all updated
  reason: MilvusComponentsUpdated
  status: "True"
  type: MilvusUpdated
endpoint: milvus-cluster-milvus.default:19530
ingress:
  loadBalancer: {}
observedGeneration: 1
replicas:
  dataNode: 1
  indexNode: 1
  mixCoord: 1
  proxy: 1
  queryNode: 1
status: Healthy
```

As displayed in the above output, `status: Healthy` shows that the deployment is successfully created and running. Also, the number of replicas for each component should match your Milvus CRD configuration.

5. Verify the Milvus cluster pods status

```
$ kubectl get pods
```

Output:

NAME	READY	STATUS
milvus-cluster-etcd-0	1/1	Running
milvus-cluster-etcd-1	1/1	Running
milvus-cluster-etcd-2	1/1	Running

0	5m45s	milvus-cluster-milvus-datanode-667456bf77-9wsnr	1/1	Running
0	75s	milvus-cluster-milvus-indexnode-57789b66cf-bg92m	1/1	Running
0	75s	milvus-cluster-milvus-mixcoord-6c559b6b4f-p5bh9	1/1	Running
0	75s	milvus-cluster-milvus-proxy-7c64b67f85-bxrdt	1/1	Running
0	75s	milvus-cluster-milvus-querynode-78c487875-w6rm4	1/1	Running
0	75s	milvus-cluster-minio-0	1/1	Running
0	5m45s	milvus-cluster-minio-1	1/1	Running
0	5m45s	milvus-cluster-minio-2	1/1	Running
0	5m45s	milvus-cluster-minio-3	1/1	Running
0	5m45s	milvus-cluster-pulsar-bookie-0	1/1	Running
0	5m44s	milvus-cluster-pulsar-bookie-1	1/1	Running
0	5m43s	milvus-cluster-pulsar-bookie-2	1/1	Running
0	5m43s	milvus-cluster-pulsar-bookie-init-44kc6	0/1	Completed
0	5m45s	milvus-cluster-pulsar-broker-0	1/1	Running
0	5m45s	milvus-cluster-pulsar-proxy-0	1/1	Running
0	5m45s	milvus-cluster-pulsar-pulsar-init-dbg4z	0/1	Completed
0	5m45s	milvus-cluster-pulsar-recovery-0	1/1	Running
0	5m45s	milvus-cluster-pulsar-zookeeper-0	1/1	Running
0	5m45s	milvus-cluster-pulsar-zookeeper-1	1/1	Running
0	4m50s	milvus-cluster-pulsar-zookeeper-2	1/1	Running
0	3m59s			

As displayed in the above output, all required pods are running and execution jobs are complete. By default Milvus creates pods the dependencies before all components.

6. When all Pods are running, the Milvus Cluster creates the necessary Vultr high performance storage volumes in your account. Verify the PVC status to verify the available block storage volumes

```
$ kubectl get pvc
```

Access the Vultr customer portal and verify that the listed object storage volumes in your output are available in your account.

Enable External Access to MilvusDB

In this section, set up an external access service using the Vultr Cloud Control Manager (CCM) and LoadBalancer service to expose the database using the Milvus default port `19530` as described below.

1. Create a new file `external_access.yaml`

```
$ nano external_access.yaml
```

2. Add the following configurations to the file

```
apiVersion: v1
kind: Service
metadata:
  name: milvus-external
  annotations:
    service.beta.kubernetes.io/vultr-loadbalancer-protocol: "tcp"
spec:
  type: LoadBalancer
  ports:
    - name: milvus
      port: 19530
      protocol: TCP
      targetPort: milvus
```

```
selector:
  app.kubernetes.io/instance: milvus-cluster
  app.kubernetes.io/managed-by: milvus-operator
  app.kubernetes.io/name: milvus
  milvus.io/service: "true"
```

Save and close the file.

The above configuration exposes the Milvus service through the default TCP port `19530`. The `targetPort` is set to `milvus` to encounter any MilvusDB default port changes.

3. Apply the external deployment to your cluster

```
$ kubectl apply -f external_access.yaml
```

Output:

```
service/milvus-external created
```

4. Wait for at least `6` minutes to deploy the Load balancer resource. Then, view the cluster service

```
$ kubectl get svc milvus-external
```

Verify the **External-IP** value in your output like the one below:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	
milvus-external	LoadBalancer	10.10.4.1	192.0.2.100	27017:32096/
PORT(S)	AGE			
TCP 106s				

In the above output, `192.0.2.100` is the load balancer external IP Address you can use to access the Milvus DB.

Test Access to MilvusDB

1. Using Pip, install the `pymilvus` module on your server

```
$ pip3 install pymilvus
```

The above command installs `pymilvus`: An open-source Python SDK designed to interact with MilvusDB.

2. Create a new Python application file

```
$ nano milvusDb.py
```

3. Add the following contents to the file. Replace `192.0.2.100` with your actual Milvus load balancer IP Address

```
from pymilvus import connections, db, utility

conn = connections.connect(host="192.0.2.100", port="19530", user="root",
password="Milvus")

database_name = "book1"

database = db.create_database(database_name)

print(f"Database '{database_name}' created successfully.")

list = db.list_database()

print(f"All available databases are {list}")
```

Save and exit the file.

The above application file connects to your Milvus service from your local terminal as the Milvus user `root` user and the password `Milvus` since the cluster `authorizationEnabled` field is set to `true`.

In addition, the application creates a `book` database and prints all the available databases. You can create a maximum of up to `64` Milvus databases on top of the `default` database to manage user privileges within the database.

4. Run the application

```
$ python3 milvusDb.py
```

When successful, your output should look like the one below:

```
Database 'book1' created successfully.  
All available databases are ['default', 'book1']
```

As displayed in the above output, a new database is created that confirms a connection to the Milvus cluster is working correctly. You can perform more functions such as creating collections and vector searches.

Scale the Milvus Cluster

In this section, scale your Milvus Cluster by increasing and decreasing the number of replica pods running in your cluster. Scaling is important when managing resource allocation depending on your user traffic requirements.

Scale the Cluster Up

1. To increase the number of pods, Edit the `milvus_cluster.yaml` file

```
$ nano milvus_cluster.yaml
```

2. Change the `queryNode:` replicas value to `4` to set a new number of replica pods

```
apiVersion: milvus.io/v1beta1  
kind: Milvus  
metadata:  
  name: milvus-cluster  
  labels:  
    app: milvus  
spec:  
  mode: cluster  
  dependencies: {}  
  components:  
    mixCoord:
```

```
    replicas: 1
  dataNode:
    replicas: 1
  indexNode:
    replicas: 1
  queryNode:
    replicas: 4
  config:
    common:
      security:
        authorizationEnabled: true
```

Save and close the file.

3. Apply the change to your cluster

```
$ kubectl apply -f milvus_cluster.yaml
```

Output:

```
milvus.milvus.io/milvus-cluster configured
```

4. View the list of running Milvus Cluster pods

```
$ kubectl get pods
```

Verify that the `milvus-querynode` pods are at least `4` similar to the output below:

NAME	READY	STATUS
milvus-cluster-etcd-0	1/1	Running
0	18m	
milvus-cluster-etcd-1	1/1	Running
0	18m	
milvus-cluster-etcd-2	1/1	Running
0	18m	
milvus-cluster-milvus-datanode-667456bf77-6tthb	1/1	Running
0	13m	
milvus-cluster-milvus-indexnode-57789b66cf-fpvnd	1/1	Running
0	13m	

milvus-cluster-milvus-mixcoord-6c559b6b4f-x5jtg	1/1	Running
0 13m		
milvus-cluster-milvus-proxy-7c64b67f85-vtc82	1/1	Running
0 13m		
milvus-cluster-milvus-querynode-78c487875-2727q	1/1	Running
0 45s		
milvus-cluster-milvus-querynode-78c487875-872qs	1/1	Running
0 45s		
milvus-cluster-milvus-querynode-78c487875-h2zq4	1/1	Running
0 13m		
milvus-cluster-milvus-querynode-78c487875-xz2jc	1/1	Running
0 45s		
milvus-cluster-minio-0	1/1	Running
0 18m		
milvus-cluster-minio-1	1/1	Running
0 18m		
milvus-cluster-minio-2	1/1	Running
0 18m		
milvus-cluster-minio-3	1/1	Running
0 18m		
milvus-cluster-pulsar-bookie-0	1/1	Running
0 17m		
milvus-cluster-pulsar-bookie-1	1/1	Running
0 17m		
milvus-cluster-pulsar-bookie-2	1/1	Running
0 17m		
milvus-cluster-pulsar-bookie-init-rfqsg	0/1	Completed
0 18m		
milvus-cluster-pulsar-broker-0	1/1	Running
0 18m		
milvus-cluster-pulsar-proxy-0	1/1	Running
0 18m		
milvus-cluster-pulsar-pulsar-init-czptk	0/1	Completed
0 18m		
milvus-cluster-pulsar-recovery-0	1/1	Running
0 18m		
milvus-cluster-pulsar-zookeeper-0	1/1	Running
0 18m		
milvus-cluster-pulsar-zookeeper-1	1/1	Running
0 16m		
milvus-cluster-pulsar-zookeeper-2	1/1	Running
0 16m		

Scale the Cluster Down

1. Modify the `milvus_cluster.yaml` file to decrease the number of pods

```
$ nano milvus_cluster.yaml
```

2. Reduce the number of `queryNode` replicas to `2`

```
apiVersion: milvus.io/v1beta1
kind: Milvus
metadata:
  name: milvus-cluster
  labels:
    app: milvus
spec:
  mode: cluster
  dependencies: {}
  components:
    mixCoord:
      replicas: 1
    dataNode:
      replicas: 1
    indexNode:
      replicas: 1
    queryNode:
      replicas: 2
  config:
    common:
      security:
        authorizationEnabled: true
```

Save and close the file.

3. Apply the change to your cluster

```
$ kubectl apply -f milvus_cluster.yaml
```

Output:

```
milvus.milvus.io/milvus-cluster configured
```

4. View the list of running pods

```
$ kubectl get pods
```

Verify that the `milvus-querynode` pods are `2` instead of `4` similar to the output below:

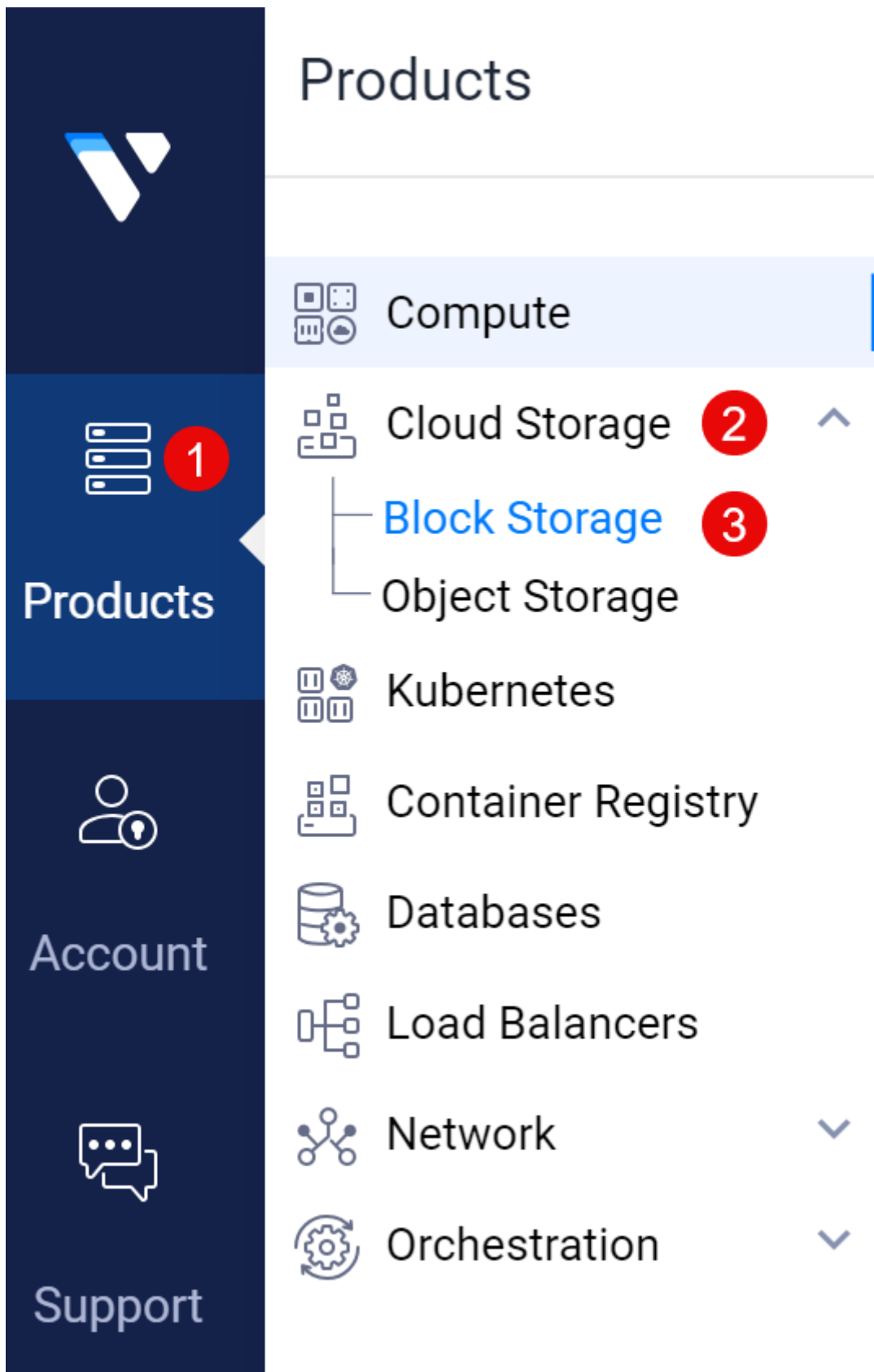
NAME	READY	STATUS
milvus-cluster-etcd-0	1/1	Running
0		
22m		
milvus-cluster-etcd-1	1/1	Running
0		
22m		
milvus-cluster-etcd-2	1/1	Running
0		
22m		
milvus-cluster-milvus-datanode-667456bf77-6tthb	1/1	Running
0		
17m		
milvus-cluster-milvus-indexnode-57789b66cf-fpvnd	1/1	Running
0		
17m		
milvus-cluster-milvus-mixcoord-6c559b6b4f-x5jtg	1/1	Running
0		
17m		
milvus-cluster-milvus-proxy-7c64b67f85-vtc82	1/1	Running
0		
17m		
milvus-cluster-milvus-querynode-78c487875-872qs	1/1	Running
0		
5m29s		
milvus-cluster-milvus-querynode-78c487875-h2zq4	1/1	Running
0		
17m		
milvus-cluster-minio-0	1/1	Running
0		
22m		
milvus-cluster-minio-1	1/1	Running
0		
22m		
milvus-cluster-minio-2	1/1	Running
0		
22m		
milvus-cluster-minio-3	1/1	Running
0		
22m		
milvus-cluster-pulsar-bookie-0	1/1	Running
0		
22m		
milvus-cluster-pulsar-bookie-1	1/1	Running
0		
22m		

milvus-cluster-pulsar-bookie-2	1/1	Running
0 22m		
milvus-cluster-pulsar-bookie-init-rfqsg	0/1	Completed
0 22m		
milvus-cluster-pulsar-broker-0	1/1	Running
0 22m		
milvus-cluster-pulsar-proxy-0	1/1	Running
0 22m		
milvus-cluster-pulsar-pulsar-init-czptk	0/1	Completed
0 22m		
milvus-cluster-pulsar-recovery-0	1/1	Running
0 22m		
milvus-cluster-pulsar-zookeeper-0	1/1	Running
0 22m		
milvus-cluster-pulsar-zookeeper-1	1/1	Running
0 21m		
milvus-cluster-pulsar-zookeeper-2	1/1	Running
0 20m		

Verify MilvusDB Block Storage Resources























MilvusDB creates at least 20 high performance Vultr block storage volumes on your account. If multiple volumes fail or get deleted, your Milvus Cluster may run into errors. To keep track of your account billing and MilvusDB resources, verify the available storage volumes on your Vultr account as described in the steps below.

1. Visit the [Vultr Customer Portal](#)
2. Click **Products**



3. Expand the **Cloud Storage** group and click **Block Storage** from the list of options

4. Verify the available MilvusDB storage volumes with the `pvc` naming scheme

<code>pvc-3a98145e6c34453b</code> 500 GB	NVMe	 New Jersey	--	\$8.93	● Active	
<code>pvc-38a6d223774f4c77</code> 10 GB	NVMe	 New Jersey	--	\$0.18	● Active	
<code>pvc-5301b2be778a453d</code> 500 GB	NVMe	 New Jersey	--	\$8.93	● Active	
<code>pvc-47f72a842eb54794</code> 500 GB	NVMe	 New Jersey	--	\$8.93	● Active	
<code>pvc-14442d426e994938</code> 10 GB	NVMe	 New Jersey	--	\$0.18	● Active	
<code>pvc-e50f6648020a47ed</code> 500 GB	NVMe	 New Jersey	--	\$8.93	● Active	
<code>pvc-7d437efed66e4bec</code> 10 GB	NVMe	 New Jersey	--	\$0.18	● Active	
<code>pvc-a4c48cd224a242f4</code> 200 GB	NVMe	 New Jersey	--	\$3.58	● Active	
<code>pvc-4a6b9aaa9302426c</code> 20 GB	NVMe	 New Jersey	--	\$0.36	● Active	
<code>pvc-20a871ea412143df</code> 100 GB	NVMe	 New Jersey	--	\$1.79	● Active	
<code>pvc-03b1ced82d4b4e10</code> 20 GB	NVMe	 New Jersey	--	\$1.79	● Active	

Troubleshooting

Depending on your Milvus Cluster deployment, you may encounter errors based on the following Milvus DB issues:

- Not enough Vultr Block Storage volumes
- Not enough Memory
- Not enough CPU
- Not enough Cluster Nodes

A Milvus Cluster requires at least 20 Vultr Block Storage volumes, verify that enough volumes are attached to your account to avoid any resource limitation errors.

1. In case your Milvus cluster pods are not running. Describe the target pod to view any error logs using the command below

```
$ kubectl describe pod_name
```

The above command displays detailed information about a specific pod in your Milvus cluster. The information may include the `status`, `events`, `namespace`, and `IP address` information.

2. Verify that the Milvus Cluster storage volumes are healthy and running

```
$ kubectl describe pvc
```

The above command displays a summary of the cluster Persistent Volume Claims (PVCs). Verify that all available PVCs are running correctly

3. If you receive an authentication error while creating a new database, verify the `user`, `Milvus Address`, and `password` details in your Python application file.

Conclusion

You have deployed a Milvus Cluster using the Milvus operator on a Vultr Kubernetes Engine (VKE) cluster. Then, you created an external service by exposing the database server outside the cluster and tested access to the service by creating a new database. For more information, visit the [Milvus DB documentation](#).

Next Steps

To install other database systems to your VKE cluster, visit the following resources:

- [Install MS SQL on Vultr Kubernetes Engine \(VKE\)](#)
- [Install MySQL on Vultr Kubernetes Engine \(VKE\)](#)
- [Install PostgreSQL on Vultr Kubernetes Engine \(VKE\)](#)



VULTR

