

How to Install MySQL on Vultr Kubernetes Engine (VKE)

Learn how to install MySQL on Vultr Kubernetes Engine (VKE) with our step-by-step guide. Deploy, configure, and manage your database efficiently on VKE.

Contents

01	Introduction	3
02	Prerequisites	3
03	Install the MySQL Operator for Kubernetes	4
04	Deploy the MySQL Cluster	5
05	Connect to the MySQL Cluster	8
06	Set Up the External Access	10
07	Scale the MySQL Cluster	12
08	Delete the MySQL Cluster	13
09	Conclusion	14
010	More Information	14

Introduction

MySQL is an open-source database server. Classified as a Relational Database Management System (RDBMS), the data stored in a MySQL database is in tabular format. It uses Structured Query Language (SQL) to store, manipulate, and retrieve data. The relational data model of the MySQL databases allows developers to link the data saved in the rows to other rows for managing complex data structures.

You can scale the MySQL database server on the Vultr Kubernetes Engine (VKE) using the MySQL Operator for Kubernetes, an extension for Kubernetes. It allows deployment and management of MySQL InnoDB clusters in the Kubernetes environment. The MySQL InnoDB cluster is a group of MySQL servers that store the same data providing redundancy and high availability for production usage.

This guide walks you through the deployment of the MySQL InnoDB cluster on the Vultr Kubernetes Engine (VKE). It covers the steps to install the MySQL Operator for Kubernetes, configure the MySQL InnoDB cluster, and set up external access to the MySQL router using Vultr Load Balancer.

Prerequisites

- [Deploy a Vultr Ubuntu 20.04 instance](#) to use as a management workstation.
- [Deploy a Kubernetes cluster at Vultr](#) with at least three nodes, each with 2GB RAM and 2 CPUs.

On the management workstation:

- Install [Kubectl](#).
- Download your VKE configuration and [configure Kubectl](#).
- Install [Vultr Container Storage Interface](#).
- Install [Vultr Cloud Controller Manager](#).

You must perform the rest of the steps in this guide from your management workstation.

Install the MySQL Operator for Kubernetes

The MySQL Operator is an extension for Kubernetes that simplifies the deployment and management of MySQL clusters. The extension consists of custom resource definitions that define the `InnoDBCluster` and the `MySQLBackup` resource types, the role bindings, and the operator. The operator is a container that manages the full lifecycle with setup and maintenance, including automation of upgrades and backup. This section explains the steps to install the custom resource definitions, the role bindings, and the operator.

Clone the MySQL Operator repository.

```
# git clone https://github.com/mysql/mysql-operator.git
```

Switch to the repository directory.

```
# cd mysql-operator
```

Install the custom resource definitions.

```
# kubectl apply -f deploy/deploy-crds.yaml
```

The above command installs the `InnoDBCluster` and the `MySQLBackup` resource types in the cluster. You use the `InnoDBCluster` resource type to define a MySQL cluster and the `MySQLBackup` resource type to create a backup.

Install the MySQL Operator.

```
# kubectl apply -f deploy/deploy-operator.yaml
```

The above command installs the necessary role bindings and an operator in the cluster that uses the [mysql/mysql-operator](https://github.com/mysql/mysql-operator) image. The operator is a container that monitors and looks for state changes in the cluster to initialize and manage the MySQL cluster's management.

Verify the installation.

```
# kubectl get pods -n mysql-operator
```

Output.

NAME	READY	STATUS	RESTARTS	AGE
mysql-operator-b689b9f69-np6sf	1/1	Running	0	10s

Deploy the MySQL Cluster

The MySQL InnoDB cluster is a group of MySQL servers that store the same data providing redundancy and high availability for production usage. The `InnoDBCluster` resource type allows declarative configuration of a MySQL cluster, you enter the number of desired MySQL router and server instances, and the operator performs the deployment. The MySQL router is a container that routes the incoming traffic to the MySQL server instances which store the data. This section explains the configuration and deployment of the MySQL cluster.

Create and enter a new configuration directory.

```
# mkdir ~/mysql-innodb-cluster
# cd ~/mysql-innodb-cluster
```

The above commands create a new directory named **mysql-innodb-cluster**. You use this directory to store configuration files related to the deployment and management of the MySQL cluster.

Using a text editor, create a new file named **secret.yaml**.

```
# nano secret.yaml
```

Add the following contents to the file.

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
type: Opaque
```

```
stringData:
  rootUser: 'root'
  rootHost: '%'
  rootPassword: 'password_here'
```

The above configuration defines a new `Secret` resource. It contains the credentials such as the username, the password, and the allowed hosts of the user that can perform administrative tasks in the MySQL cluster.

Apply the configuration.

```
# kubectl apply -f secret.yaml
```

Create a new file named **cluster.yaml**.

```
# nano cluster.yaml
```

Add the following contents to the file.

```
apiVersion: mysql.oracle.com/v2
kind: InnoDBCluster
metadata:
  name: mysql-cluster
spec:
  secretName: mysql-secret
  instances: 3
  router:
    instances: 1
  tlsUseSelfSigned: true
  datadirVolumeClaimTemplate:
    accessModes: [ "ReadWriteOnce" ]
    storageClassName: vultr-block-storage
  resources:
    requests:
      storage: 50Gi
```

The above configuration defines a new `InnoDBCluster` resource. This resource declares a MySQL cluster with the specified values. When you apply the configuration, the operator provisions a `StatefulSet` for MySQL server instances with replicas matching the number of instances specified and a `ReplicaSet` for

MySQL router instances with the replicas matching the number of routers specified.

The following are the highlights of the configuration:

- **metadata.name:** The name of the `InnoDBCluster` resource, used as a prefix for sub-resources.
- **spec.secretName:** The secret resource for creating a user to perform administrative operations. It must contain the **rootUser**, the **rootHost**, and the **rootPassword** values.
- **spec.instances:** The number of MySQL server instances to spawn.
- **spec.router.instances:** The number of MySQL router instances to spawn.
- **datadirVolumeClaimTemplate:** The storage class and the storage size for the volumes created for each MySQL server instance. The above configuration uses the storage class defined by Vultr Container Storage Interface (CSI) to provision Vultr Block Storage to store MySQL data. Refer to the [Vultr Container Storage Interface documentation](#) for more information.

Refer to the [MySQL Operator Custom Resource Properties](#) to explore all available properties for advanced configuration.

Apply the configuration.

```
# kubectl apply -f cluster.yaml
```

Follow the deployment process.

```
# watch -n 0.1 kubectl get pods
```

It takes around 5 to 10 minutes to finish the deployment.

```
Every 0.1s: kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
mysql-cluster-0	2/2	Running	0	3m6s
mysql-cluster-1	2/2	Running	0	102s

mysql-cluster-2	2/2	Running	0	55s
mysql-cluster-router-568557d9c6-ndgvc	1/1	Running	0	3m6s

If the deployment stops making any progress, you can use the following commands to troubleshoot and find the issue causing the delay.

```
# kubectl describe innodbcluster mysql-cluster
# kubectl describe pod mysql-cluster-0
# kubectl get pvc
# kubectl get pv
```

The following are the issues you might face during the deployment:

- Not having enough CPU.
- Not having enough Memory.
- Vultr Block Storage limit exceeded.

Ensure that your Vultr Kubernetes Engine (VKE) cluster has enough resources available and you do not exceed the total number of Vultr Block Storage volumes limit.

Connect to the MySQL Cluster

The MySQL Operator creates a `ClusterIP` service in the Kubernetes cluster during the initialization. It exposes the connection to MySQL router instances on a cluster-wide IP which routes incoming connections to primary/secondary MySQL server instances. You can use the service hostname or the cluster IP to establish a connection with the MySQL cluster from any container running in the Kubernetes cluster. This section demonstrates two methods to connect to the MySQL cluster.

The following is the syntax of the service hostname.

```
{{ InnoDBCluster.metadata.name }}.{{ namespace }}.svc.cluster.local
```

Fetch the IP address of the `ClusterIP` service.

```
# kubectl get svc mysql-cluster -o jsonpath='{.spec.clusterIP}'
```

Connect via MySQL Shell

You can create a temporary container in interactive mode to access the MySQL cluster via the MySQL shell for debugging or administrative tasks.

Create a temporary container.

```
# kubectl run --rm -it mysql-shell --image=mysql/mysql-operator -- mysqlsh
```

The above command runs a new container named **mysql-shell** in the interactive mode with the [mysql/mysql-operator](#) image as it includes the MySQL shell binary.

Connect to the MySQL cluster.

```
MySQL JS> \connect root@mysql-cluster.default.svc.cluster.local
```

Output.

```
Creating a session to 'root@mysql-cluster.default.svc.cluster.local'

Please provide the password for 'root@mysql-cluster.default.svc.cluster.local':
*****

Save password for 'root@mysql-cluster.default.svc.cluster.local'? [Y]es/[N]o/[e]ver
(default No): Y

Fetching schema names for autocompletion... Press ^C to stop.
Your MySQL connection id is 968335 (X protocol)
Server version: 8.0.30 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.

MySQL mysql-cluster.default.svc.cluster.local:33060+ ssl JS>
```

You can exit the MySQL shell using the `\exit` command, which also deletes the running container.

Connect via Port Forwarding

You can create a temporary network tunnel that routes the traffic from a specified port at your management workstation to the specified port on the MySQL service. This allows establishing connections from the management workstation to the MySQL cluster running in the Kubernetes cluster.

Create a temporary network tunnel.

```
# kubectl port-forward service/mysql-cluster 3306
```

The above command maps the **3306** port at your management workstation to the MySQL service on the Kubernetes cluster. This network tunnel allows you to connect with the MySQL cluster in a new terminal window from your management workstation to perform administrative tasks, test an application, and so on.

Output.

```
Forwarding from 127.0.0.1:3306 -> 6446
Forwarding from [::1]:3306 -> 6446
```

You can exit the network tunnel using the Ctrl + C key combination.

Set Up the External Access

The Vultr Cloud Controller Manager (CCM) binds the `LoadBalancer` and the `Ingress` service types with the Vultr Load Balancer service. You use the `LoadBalancer` service type to configure a new service like the `ClusterIP` service created by the MySQL operator to set up external access. You can specify the desired ports to expose. This section explains the configuration and deployment of the Vultr Load Balancer that exposes the MySQL cluster.

Create a new file named **external.yaml**.

```
# nano external.yaml
```

Add the following contents to the file.

```
apiVersion: v1
kind: Service
metadata:
  name: mysql-external
  annotations:
    service.beta.kubernetes.io/vultr-loadbalancer-protocol: "tcp"
spec:
  type: LoadBalancer
  ports:
    - name: mysql
      port: 3306
      protocol: TCP
      targetPort: 6446
  selector:
    component: mysqlrouter
    mysql.oracle.com/cluster: mysql-cluster
    tier: mysql
```

Apply the configuration.

```
# kubectl apply -f external.yaml
```

Follow the deployment process.

```
# watch -n 0.1 kubectl get svc mysql-external
```

It takes around 5 to 10 minutes to finish the deployment.

```
Every 0.1s: kubectl get svc mysql-external
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
mysql-external	LoadBalancer	10.96.248.162	45.63.58.47	3306:31975/TCP	5m

The external IP of the `LoadBalancer` service allows establishing a connection with the MySQL cluster from other networks. You can verify the connection using the [MySQL shell](#) on your local system.

Scale the MySQL Cluster

The MySQL Operator for Kubernetes provides automated scaling and zero downtime. You can reconfigure the MySQL cluster to reduce or increase the number of server instances by changing the `spec.instances` value and the number of router instances by the `spec.router.instances` value in the `InnoDBCluster` resource configuration file. This section explains the steps to reconfigure the MySQL cluster with the increased number of MySQL server and router instances.

Edit the **cluster.yaml** configuration file.

```
# nano cluster.yaml
```

Change the `spec.instances` value to 5 and the `spec.router.instances` value to 2.

```
apiVersion: mysql.oracle.com/v2
kind: InnoDBCluster
metadata:
  name: mysql-cluster
spec:
  secretName: mysql-secret
  instances: 5
  router:
    instances: 2
  tlsUseSelfSigned: true
  datadirVolumeClaimTemplate:
    accessModes: [ "ReadWriteOnce" ]
    storageClassName: vultr-block-storage
  resources:
    requests:
      storage: 50Gi
```

Apply the configuration.

```
# kubectl apply -f cluster.yaml
```

Follow the deployment process.

```
# watch -n 0.1 kubectl get pods
```

It takes around 2 to 3 minutes to finish the deployment.

```
Every 0.1s: kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
mysql-cluster-0	2/2	Running	0	4d22h
mysql-cluster-1	2/2	Running	0	4d22h
mysql-cluster-2	2/2	Running	0	4d22h
mysql-cluster-3	2/2	Running	0	73s
mysql-cluster-4	1/2	Running	0	73s
mysql-cluster-router-568557d9c6-45sbp	1/1	Running	0	73s
mysql-cluster-router-568557d9c6-ndgvc	1/1	Running	0	4d22h

The reconfiguration options of the `InnoDBCluster` resource are not limited to changing the number of instances spawn. It also enables you to change the version of the MySQL package used in the cluster via the `spec.version` value. Kubernetes deletes and recreates each pod of the MySQL cluster from last to first when you change the version.

The `InnoDBCluster` resource properties remain unchanged in case of manual changes to the `StatefulSet` or the `ReplicaSet` to change the number of instances spawned or any other property.

Delete the MySQL Cluster

The MySQL cluster deployment demonstrated in this guide uses add-ons such as the Vultr Block Storage for storing MySQL data and the Vultr Load Balancer

for setting up external access. The add-ons to the Vultr Kubernetes Engine (VKE) cluster incur additional charges, and you must delete them to avoid further charges. This section explains the deletion of the `InnoDBCluster` resource, the associated `PVC` resources, and the `LoadBalancer` resource.

Delete the `InnoDBCluster` resource.

```
# kubectl delete innodbcluster mysql-cluster
```

Delete the associated `PVC` resources.

```
# kubectl delete pvc --selector mysql.oracle.com/cluster=mysql-cluster
```

Delete the `LoadBalancer` resource.

```
# kubectl delete -f external.yaml
```

Conclusion

You deployed a MySQL cluster on the Vultr Kubernetes Engine (VKE) using the MySQL Operator for Kubernetes. The cluster stores the database contents on the Vultr Block Storage for persistent data storage and uses the Vultr Load Balancer to expose the MySQL router outside the Kubernetes cluster. The guide walks you through the basic deployment lifecycle of the MySQL InnoDB cluster. To ensure data safety in production use cases, you can implement backup profiles in the `InnoDBCluster` resource and set up automatic periodic backups of the cluster. For more information about MySQL backups, refer to the [Handling MySQL Backups](#) section in the MySQL Operator for Kubernetes documentation.

More Information

- [Vultr Kubernetes Engine](#)
- [MySQL Operator for Kubernetes](#)
- [Vultr Container Storage Interface](#)
- [Vultr Cloud Controller Manager](#)



VULTR

