

How to Deploy MS SQL Server 2022 on Vultr Kubernetes Engine

Learn how to deploy Microsoft SQL Server 2022 on Vultr Kubernetes Engine with our step-by-step guide covering configuration, installation, and optimization.

Contents

01	Introduction	3
02	Prerequisites	3
03	Prepare the Kubernetes Cluster	4
04	Deploy the SQL Server	5
05	Configure the External Access	8
06	Connect to the SQL Server using sqlcmd	10
07	Connect to the SQL Server using SQL Server Management Studio	10
08	Conclusion	11
09	More Information	11

Introduction

Microsoft SQL (MS SQL) Server is a relational database management system (RDBMS). It enables you to manage and store data in a structured format, offering ease of access and manipulation of the data. The SQL Server offers a range of features and tools for data management, analysis, and reporting and is often used in enterprise-level applications. They offer various editions to suit different needs and budgets. It is available as a container image, which allows you to run SQL Server in a containerized environment such as Docker or Kubernetes. The containerized version of SQL Server provides additional isolation and security benefits because it allows the database to run in a separate container with its resources and access controls.

Deploying the SQL Server on Kubernetes can offer greater flexibility, cost savings, performance, and security compared to running SQL Server on a Windows VM. This article demonstrates the steps to deploy the Microsoft SQL Server 2022 on the Vultr Kubernetes Engine (VKE). It also walks you through the steps to deploy a Vultr Load Balancer to expose the database server outside the Kubernetes cluster and use the Vultr Block Storage service for persistent storage.

Prerequisites

Before you begin, you should:

- Deploy a [Vultr Kubernetes Engine](#) cluster.
- Deploy a [Ubuntu 22.04](#) server to use as a management workstation. On the management workstation:
 - Install [Kubectl](#).
 - Download your VKE configuration and [configure Kubectl](#).

Provision the cluster in the region closest to the client to avoid slow operations due to network latency.

Prepare the Kubernetes Cluster

You must prepare the Kubernetes cluster for the SQL Server deployment by creating the required resources. This section explains the steps to create a Secret resource for storing the credentials used for protecting the SQL server and a PVC resource to provision a block storage service for persistent storage.

Create a Secret resource named `mssql-login`.

```
# kubectl create secret generic mssql-login --from-literal=MSSQL_SA_PASSWORD="YOUR_PASS"
```

The above command creates a new Secret resource named `mssql-login` containing a single key-value pair defining the credentials for the `sa` user. You must use a secure password with a minimum length of 8 characters. The `sa` user is the system administrator user with full access to all database objects and settings. This user has the highest permissions and can perform any operation, including creating and modifying tables, users, and other objects, configuring database settings, and security.

Create a new manifest file named `mssql-data.yaml`.

```
# nano mssql-data.yaml
```

Add the following contents to the file and save the file using `Ctrl + X` then `Enter`.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mssql-data
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
```

```
storage: 100Gi
storageClassName: vultr-block-storage
```

The above manifest declares a PVC resource named `mssql-data` that provisions a Vultr Block Storage linked resource. It uses the `vultr-block-storage` storage class created by the Container Storage Interface (CSI) Driver for Vultr. You can change the storage size by editing the `spec.resources.requests.storage` value. This resource will serve as a persistent storage block for the SQL Server container. Refer to the [VKE Reference Guide](#) for more information.

Apply the manifest file.

```
# kubectl apply -f mssql-data.yaml
```

Verify the deployment.

```
# kubectl get pvc mssql-data
```

It may take a few minutes before the resource is ready for use. If the deployment takes more than a few minutes, you can use the `kubectl describe pvc mssql-data` command to view the event log for troubleshooting the errors.

Deploy the SQL Server

The SQL Server container images are available on the Microsoft Container Registry. You can deploy the SQL Server container images in a containerized environment like Kubernetes. Deploying the SQL Server on a Kubernetes cluster offers improved scalability, security, flexibility, and cost savings compared to the other options. If the SQL Server pod fails, Kubernetes automatically re-creates a new pod, starting where it left off with the help of persistent storage. This section explains the steps to deploy the SQL Server on the Kubernetes cluster using a Deployment resource.

Create a new manifest file named `mssql-deployment.yaml`.

```
# nano mssql-deployment.yaml
```

Add the following contents to the file and save the file using `Ctrl + X` then `Enter`.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mssql-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mssql
  template:
    metadata:
      labels:
        app: mssql
    spec:
      terminationGracePeriodSeconds: 30
      hostname: mssqlinst
      securityContext:
        fsGroup: 10001
      containers:
        - name: mssql
          image: mcr.microsoft.com/mssql/server:2022-latest
          ports:
            - containerPort: 1433
          env:
            - name: MSSQL_PID
              value: "Developer"
            - name: ACCEPT_EULA
              value: "Y"
            - name: MSSQL_SA_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mssql-login
                  key: MSSQL_SA_PASSWORD
          volumeMounts:
            - name: mssqldb
              mountPath: /var/opt/mssql
      volumes:
        - name: mssqldb
```

```
persistentVolumeClaim:  
  claimName: mssql-data
```

The above manifest declares a Deployment resource named `mssql` that deploys a ReplicaSet with a single replica of the `mssql` container that uses the official SQL Server container image. The container image expects 3 environment variables to configure the SQL Server: `MSSQL_PID`, `ACCEPT_EULA`, and `MSSQL_SA_PASSWORD`. The `MSSQL_PID` variable defines the edition of the SQL Server. The `ACCEPT_EULA` variable defines that you agree with the mandatory [End-User License Agreement \(EULA\)](#). The `MSSQL_SA_PASSWORD` variable defines the credentials for the `sa` user using the Secret resource created in the previous steps. It also uses the `mssql-data` PVC resource for storing the `/var/opt/mssql` directory.

Apply the manifest file.

```
# kubectl apply -f mssql-deployment.yaml
```

Verify the deployment.

```
# kubectl get deployment mssql-deployment
```

It may take a few minutes before the resource is ready for use. If the deployment takes more than a few minutes, you can use the `kubectl describe deployment mssql-deployment` command to view the event log for troubleshooting the errors.

Server Editions and License

As mentioned above, the SQL Server container image uses the `MSSQL_PID` environment variable to determine the edition you want. The following are the possible values for the `MSSQL_PID` environment variable.

- Developer
- Express
- Web
- Standard
- Enterprise

- A product key

The Developer and Express editions are free, the Developer edition offers the same features as the Enterprise edition. Still, you can only use it for development purposes, as it does not allow production usage. The other options, such as Web, Standard, and Enterprise, requires you to purchase a license for additional features. Refer to the [SQL Server Editions](#) page for more information.

The SQL Server container image has no technical checks to validate the license. However, accepting the [End-User License Agreement \(EULA\)](#) is mandatory to use the SQL Server. This means you need to purchase a license before using the paid editions of the SQL Server. By accepting the EULA, you agree to the terms and conditions, which include the condition to purchase a license for the paid editions of SQL Server. This is important to keep in mind when using SQL Server in a container, as using the paid editions without a valid license could result in legal consequences.

As mentioned in the above list, you can also use a product key to define the SQL Server edition, which automatically determines the edition according to the specified product key. While specifying a product key, it must be in the form of `#####-#####-#####-#####-#####`, where `#` is a number or a letter.

Configure the External Access

The Vultr Kubernetes Engine allows you to provision a linked load balancer resource to expose the connections to the SQL Server container outside the network using Vultr Cloud Controller Manager. This section demonstrates the steps to create a Service resource for exposing the connections to SQL Server using a Vultr Load Balancer.

Create a new manifest file named `mssql-loadbalancer.yaml`.

```
# nano mssql-loadbalancer.yaml
```

Add the following contents to the file and save the file using `Ctrl + X` then Enter.

```
apiVersion: v1
kind: Service
metadata:
  name: mssql-loadbalancer
  annotations:
    service.beta.kubernetes.io/vultr-loadbalancer-protocol: "tcp"
spec:
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 1433
      targetPort: 1433
  selector:
    app: mssql
```

The above manifest declares a Service resource named `mssql-loadbalancer` that provisions a Vultr Load Balancer linked resource. It uses the `vultr-loadbalancer-protocol` annotation type that comes with the Cloud Controller Manager for Vultr to set the connection protocol. This resource will expose connections to the SQL Server outside the Kubernetes cluster. Refer to the [VKE Reference Guide](#) for more information.

Apply the manifest file.

```
# kubectl apply -f mssql-loadbalancer.yaml
```

Verify the deployment.

```
# kubectl get service mssql-loadbalancer
```

It may take 5 to 10 minutes before the resource is ready for use. If the deployment takes more than that, you can use the `kubectl describe service mssql-loadbalancer` command to view the event log for troubleshooting the errors.

Fetch the Load Balancer IP Address.

```
# kubectl get service mssql-loadbalancer -o
jsonpath={.status.loadBalancer.ingress[0].ip}
```

The output contains the IP address by which you can connect to the SQL Server outside the cluster on the default port `1433`. The load balancer may take additional few minutes after it enters the **Ready** state to start handling the requests.

Connect to the SQL Server using `sqlcmd`

`sqlcmd` is a command-line utility for the SQL Server that allows users to interact with the database using SQL commands and scripts. It provides a flexible and efficient way to run queries, manage database objects, and perform other tasks on the database. Refer to the [sqlcmd Utility](#) page for more information.

Connect to the SQL Server using `sqlcmd`.

```
sqlcmd -S LOADBALANCER_IP -U sa -P PASSWORD
```

Replace `LOADBALANCER_IP` with the IP address of the load balancer and `PASSWORD` with the password you used in the previous steps to create the Secret resource.

Connect to the SQL Server using SQL Server Management Studio

SQL Server Management Studio (SSMS) is a graphical user interface tool for managing and administering SQL Server databases. It provides a comprehensive set of tools and features for managing database objects, running queries, performing backups and restores, and monitoring the performance and health of the database. Refer to the [SQL Server Management Studio](#) page for more information.

Connect to the SQL Server using SQL Server Management Studio.

1. Open the SQL Server Management Studio and click the "File" button in the toolbar.
2. From the dropdown menu, select the "Connect Object Explorer.." option.
3. Set the server type to "Database Engine" from the dropdown menu.

4. Select "SQL Server Authentication" as the authentication mode, and enter the login credentials in the "Login" and "Password" fields.
5. Click the "Connect" button to establish a connection.

Tested on the SQL Server Management Studio v18.21.1

Conclusion

Deploying the SQL Server on Kubernetes can offer greater flexibility, cost savings, performance, and security compared to running SQL Server on a Windows VM. This article demonstrated the steps to deploy the Microsoft SQL (MS SQL) Server 2022 on the Vultr Kubernetes Engine (VKE). It also walked you through the steps to deploy a Vultr Load Balancer to expose the database server outside the Kubernetes cluster and use the Vultr Block Storage service for persistent storage.

Note that the additional services used with the Vultr Kubernetes Engine, such as the Vultr Block Storage and the Vultr Load Balancer, are not included with the cluster and are individually chargeable cloud resources.

More Information

- [Microsoft SQL Server Licensing](#)
- [Microsoft SQL Server Container Images](#)



VULTR

