Quickstart: Run SQL Server Linux container images with Docker

Article • 02/24/2024

Applies to: ✓ SQL Server - Linux

In this quickstart, you use Docker to pull and run the SQL Server 2022 (16.x) Linux container image, mssql-server-linux. Then you can connect with **sqlcmd** to create your first database and run queries.

For more information on supported platforms, see Release notes for SQL Server 2022 (16.x) on Linux.

Tip

This quickstart creates SQL Server 2022 (16.x) containers. If you prefer to create Linux containers for different versions of SQL Server, see the SQL Server 2017 (14.x) or SQL Server 2019 (15.x) versions of this article.

This image consists of SQL Server running on Linux based on Ubuntu. It can be used with the Docker Engine 1.8+ on Linux.

The examples in this article use the `docker` command. However, most of these commands also work with Podman. Podman provides a command-line interface similar to the Docker Engine. You can find out more about Podman.

Important

sqlcmd doesn't currently support the `MSSQL_PID` parameter when creating containers. If you use the **sqlcmd** instructions in this quickstart, you create a container with the Developer edition of SQL Server. Use the command line interface (CLI) instructions to create a container using the license of your choice. For more information, see Deploy and connect to SQL Server Linux containers.

Prerequisites

- Docker Engine 1.8+ on any supported Linux distribution. For more information, see Install Docker.

- For more information on hardware requirements and processor support, see SQL Server 2022: Hardware and software requirements
- Docker `overlay2` storage driver. This driver is the default for most users. If you aren't using this storage provider and need to change, see the instructions and warnings in the Docker documentation for configuring overlay2 .
- Install the latest **sqlcmd** on your Docker host.
- At least 2 GB of disk space.
- At least 2 GB of RAM.
- System requirements for SQL Server on Linux.

Pull and run the SQL Server Linux container image

Before starting the following steps, make sure that you select your preferred shell (**bash**, **PowerShell**, or **cmd**) at the top of this article.

Note

For the bash commands in this article, `sudo` is used. If you don't want to use `sudo` to run Docker, you can configure a `docker` group and add users to that group. For more information, see Post-installation steps for Linux .

CLI

Pull the container from the registry

Pull the SQL Server 2022 (16.x) Linux container image from the Microsoft Container Registry.

Bash

```
sudo docker pull mcr.microsoft.com/mssql/server:2022-latest
```

Tip

This quickstart creates SQL Server 2022 (16.x) containers. If you prefer to create Linux containers for different versions of SQL Server, see the SQL Server 2017 (14.x) or SQL Server 2019 (15.x) versions of this article.

The previous command pulls the latest SQL Server 2022 (16.x) Linux container image. If you want to pull a specific image, you add a colon and the tag name, such as `mcr.microsoft.com/mssql/server:2022-GA-ubuntu`. To see all available images, see the Microsoft Artifact Registry.

Run the container

To run the Linux container image with Docker, you can use the following command from a bash shell or elevated PowerShell command prompt.

Important

The `SA_PASSWORD` environment variable is deprecated. Use `MSSQL_SA_PASSWORD` instead.

Bash

```bash
sudo docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<Your-Strong@Passw0rd>" \
   -p 1433:1433 --name sql1 --hostname sql1 \
   -d \
   mcr.microsoft.com/mssql/server:2022-latest
```

Your password should follow the SQL Server default password policy, otherwise the container can't set up SQL Server, and stops working. By default, the password must be at least eight characters long and contain characters from three of the following four sets: uppercase letters, lowercase letters, base-10 digits, and symbols. You can examine the error log by using the docker logs command.

By default, this quickstart creates a container with the Developer edition of SQL Server. The process for running production editions in containers is slightly different. For more information, see Run production container images.

The following table provides a description of the parameters in the previous `docker run` example:

Expand table

Parameter	Description
`-e "ACCEPT_EULA=Y"`	Set the `ACCEPT_EULA` variable to any value to confirm your acceptance of the End-User Licensing Agreement. Required setting for the SQL Server image.
`-e "MSSQL_SA_PASSWORD=<YourStrong@Passw0rd>"`	Specify your own strong password that is at least eight characters and meets the Password Policy. Required setting for the SQL Server image.
`-e "MSSQL_COLLATION=<SQL_Server_collation>"`	Specify a custom SQL Server collation, instead of the default `SQL_Latin1_General_CP1_CI_AS`.
`-p 1433:1433`	Map a TCP port on the host environment (first value) with a TCP port in the container (second value). In this example, SQL Server is listening on TCP 1433 in the container and this container port is then exposed to TCP port 1433 on the host.
`--name sql1`	Specify a custom name for the container rather than a randomly generated one. If you run more than one container, you can't reuse this same name.
`--hostname sql1`	Used to explicitly set the container hostname. If you don't specify the hostname, it defaults to the container ID, which is a randomly generated system GUID.
`-d`	Run the container in the background (daemon).
`mcr.microsoft.com/mssql/server:2022-latest`	The SQL Server Linux container image.

Change the system administrator password

The **SA** account is a system administrator on the SQL Server instance that gets created during setup. After you create your SQL Server container, the `MSSQL_SA_PASSWORD` environment variable you specified is discoverable by running `echo $MSSQL_SA_PASSWORD` in the container. For security purposes, you should change your SA password in a production environment.

1. Choose a strong password to use for the SA user.

2. Use `docker exec` to run **sqlcmd** to change the password using Transact-SQL. In the following example, the old and new passwords are read from user input.

   Bash

   ```bash
   sudo docker exec -it sql1 /opt/mssql-tools/bin/sqlcmd \
   -S localhost -U SA \
    -P "$(read -sp "Enter current SA password: "; echo
   "${REPLY}")" \
    -Q "ALTER LOGIN SA WITH PASSWORD=\"$(read -sp "Enter new SA
   password: "; echo "${REPLY}")\""
   ```

   ⓘ **Note**

   Newer versions of **sqlcmd** are secure by default. For more information about connection encryption, see sqlcmd utility for Windows, and Connecting with sqlcmd for Linux and macOS. If the connection doesn't succeed, you can add the `-No` option to **sqlcmd** to specify that encryption is optional, not mandatory.

Disable the `sa` account as a best practice

ⓘ **Important**

You'll need these credentials for later steps. Be sure to write down the user ID and password that you enter here.

When you connect to your SQL Server instance using the `sa` account for the first time after installation, it's important for you to follow these steps, and then immediately disable the `sa` login as a security best practice.

1. Create a new login, and make it a member of the **sysadmin** server role.

   - Depending on whether you have a container or non-container deployment, enable Windows authentication, and create a new Windows-based login and add it to the **sysadmin** server role.

     - Tutorial: Use adutil to configure Active Directory authentication with SQL Server on Linux

    - Tutorial: Configure Active Directory authentication with SQL Server on Linux containers

  - Otherwise, create a login using SQL Server authentication, and add it to the **sysadmin** server role.

2. Connect to the SQL Server instance using the new login you created.

3. Disable the `sa` account, as recommended for security best practice.

View list of containers

1. To view your Docker containers, use the `docker ps` command.

   Bash

   ```
   sudo docker ps -a
   ```

   You should see output similar to the following example:

   Output

   ```
   CONTAINER ID   IMAGE
   COMMAND                     CREATED          STATUS          PORTS
   NAMES
   d4a1999ef83e   mcr.microsoft.com/mssql/server:2022-latest
   "/opt/mssql/bin/perm..."    2 minutes ago    Up 2 minutes
   0.0.0.0:1433->1433/tcp, :::1433->1433/tcp    sql1
   ```

2. If the `STATUS` column shows a status of `Up`, then SQL Server is running in the container and listening on the port specified in the `PORTS` column. If the `STATUS` column for your SQL Server container shows `Exited`, see Troubleshoot SQL Server Docker containers. The server is ready for connections once the SQL Server error logs display the message: `SQL Server is now ready for client connections. This is an informational message; no user action is required`. You can review the SQL Server error log inside the container using the command:

   Bash

   ```
   sudo docker exec -t sql1 cat /var/opt/mssql/log/errorlog | grep
   connection
   ```

The `--hostname` parameter, as discussed previously, changes the internal name of the container to a custom value. This value is the name you see returned in the following Transact-SQL query:

SQL

```sql
SELECT @@SERVERNAME,
    SERVERPROPERTY('ComputerNamePhysicalNetBIOS'),
    SERVERPROPERTY('MachineName'),
    SERVERPROPERTY('ServerName');
```

Setting `--hostname` and `--name` to the same value is a good way to easily identify the target container.

Connect to SQL Server

The following steps use the SQL Server command-line tool, sqlcmd utility, inside the container to connect to SQL Server.

1. Use the `docker exec -it` command to start an interactive bash shell inside your running container. In the following example, `sql1` is name specified by the `--name` parameter when you created the container.

   Bash

   ```bash
   sudo docker exec -it sql1 "bash"
   ```

2. Once inside the container, connect locally with **sqlcmd**, using its full path.

   Bash

   ```bash
   sudo /opt/mssql-tools/bin/sqlcmd -S localhost -U <userid> -P "
   <YourNewStrong@Passw0rd>"
   ```

   ⓘ **Note**

   Newer versions of **sqlcmd** are secure by default. For more information about connection encryption, see sqlcmd utility for Windows, and Connecting with sqlcmd for Linux and macOS. If the connection doesn't succeed, you can add the `-No` option to **sqlcmd** to specify that encryption is optional, not mandatory.

You can omit the password on the command-line to be prompted to enter it. For example:

```bash
sudo /opt/mssql-tools/bin/sqlcmd -S localhost -U <userid>
```

3. If successful, you should get to a **sqlcmd** command prompt: `1>`.

Create and query data

The following sections walk you through using **sqlcmd** and Transact-SQL to create a new database, add data, and run a query.

Create a new database

The following steps create a new database named `TestDB`.

1. From the **sqlcmd** command prompt, paste the following Transact-SQL command to create a test database:

```sql
CREATE DATABASE TestDB;
```

2. On the next line, write a query to return the name of all of the databases on your server:

```sql
SELECT Name from sys.databases;
```

3. The previous two commands weren't run immediately. Type `GO` on a new line to run the previous commands:

```sql
GO
```

Insert data

Next create a new table, `Inventory`, and insert two new rows.

1. From the *sqlcmd* command prompt, switch context to the new `TestDB` database:

   SQL

   ```sql
   USE TestDB;
   ```

2. Create new table named `Inventory`:

   SQL

   ```sql
   CREATE TABLE Inventory (id INT, name NVARCHAR(50), quantity INT);
   ```

3. Insert data into the new table:

   SQL

   ```sql
   INSERT INTO Inventory VALUES (1, 'banana', 150); INSERT INTO
   Inventory VALUES (2, 'orange', 154);
   ```

4. Type `GO` to run the previous commands:

   SQL

   ```sql
   GO
   ```

Select data

Now, run a query to return data from the `Inventory` table.

1. From the **sqlcmd** command prompt, enter a query that returns rows from the `Inventory` table where the quantity is greater than 152:

   SQL

   ```sql
   SELECT * FROM Inventory WHERE quantity > 152;
   ```

2. Run the command:

   SQL

```
GO
```

Exit the sqlcmd command prompt

1. To end your **sqlcmd** session, type `QUIT`:

   SQL
   ```
   QUIT
   ```

2. To exit the interactive command-prompt in your container, type `exit`. Your container continues to run after you exit the interactive bash shell.

Connect from outside the container

CLI

You can also connect to the SQL Server instance on your Docker machine from any external Linux, Windows, or macOS tool that supports SQL connections. The external tool uses the IP address for the host machine.

The following steps use **sqlcmd** outside of your container to connect to SQL Server running in the container. These steps assume that you already have the SQL Server command-line tools installed outside of your container. The same principles apply when using other tools, but the process of connecting is unique to each tool.

1. Find the IP address for your container's host machine, using `ifconfig` or `ip addr`.

2. For this example, install the **sqlcmd** tool on your client machine. For more information, see sqlcmd utility or Install the SQL Server command-line tools sqlcmd and bcp on Linux.

3. Run **sqlcmd** specifying the IP address and the port mapped to port 1433 in your container. In this example, the port is the same as port 1433 on the host machine. If you specified a different mapped port on the host machine, you would use it here. You also need to open the appropriate inbound port on your firewall to allow the connection.

   ⓘ Note

Newer versions of **sqlcmd** are secure by default. If the connection doesn't succeed, and you're using version 18 or higher, you can add the `-No` option to **sqlcmd** to specify that encryption is optional, not mandatory.

Bash

```bash
sudo sqlcmd -S <ip_address>,1433 -U <userid> -P "
<YourNewStrong@Passw0rd>"
```

4. Run Transact-SQL commands. When finished, type `QUIT`.

Other common tools to connect to SQL Server include:

- SQL Server extension for Visual Studio Code
- Use SQL Server Management Studio on Windows to manage SQL Server on Linux
- What is Azure Data Studio?
- mssql-cli (Preview)
- Manage SQL Server on Linux with PowerShell Core

Remove your container

CLI

If you want to remove the SQL Server container used in this tutorial, run the following commands:

Bash

```bash
sudo docker stop sql1
sudo docker rm sql1
```

⚠ **Warning**

Stopping and removing a container permanently deletes any SQL Server data in the container. If you need to preserve your data, create and copy a backup file out of the container or use a container data persistence technique.

Docker demo

After you finish using the SQL Server Linux container image for Docker, you might want to know how Docker is used to improve development and testing. The following video shows how Docker can be used in a continuous integration and deployment scenario. https://channel9.msdn.com/Events/Connect/2017/T152/player?nocookie=true&locale=en-us&embedUrl=%2Fsql%2Flinux%2Fquickstart-install-connect-docker

Related tasks

- Run multiple SQL Server containers
- Persist your data

Related content

- Restore a SQL Server database in a Linux container
- Troubleshoot SQL Server Docker containers
- mssql-docker GitHub repository

Contribute to SQL documentation

Did you know that you can edit SQL content yourself? If you do so, not only do you help improve our documentation, but you also get credited as a contributor to the page.

For more information, see How to contribute to SQL Server documentation