Red Hat Enterprise Linux 10

Configuring and using database servers

Installing, configuring, backing up and migrating data on database servers

Last Updated: 2026-01-16

Red Hat Enterprise Linux 10 Configuring and using database servers

Installing, configuring, backing up and migrating data on database servers

Legal Notice

Copyright © Red Hat.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Install the MariaDB, MySQL, or PostgreSQL database server on Red Hat Enterprise Linux 10. Configure the chosen database server, back up your data, and migrate your data to a later version of the database server.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION 4

CHAPTER 1. USING MARIADB 5
1.1. INSTALLING MARIADB 5
1.2. USING CONTAINERS TO RUN MULTIPLE MARIADB AND MYSQL INSTANCES ON A SINGLE HOST 5
1.3. CONFIGURING NETWORK ACCESS TO MARIADB 6
1.4. SETTING UP TLS ENCRYPTION ON A MARIADB SERVER 7
1.4.1. Placing the CA certificate, server certificate, and private key on the MariaDB server 7
1.4.2. Configuring TLS encryption on a MariaDB server 8
1.4.3. Requiring TLS encrypted connections for specific user accounts on a MariaDB server 10
1.5. CONFIGURING THE MARIADB CLIENT TO USE TLS ENCRYPTION BY DEFAULT 11
1.6. BACKING UP AND RESTORING MARIADB DATA WITH LOGICAL DUMPS 12
1.6.1. Performing a logical backup by using mariadb-dump 12
1.6.2. Restoring MariaDB data from a dump in SQL format 14
1.7. BACKING UP AND RESTORING MARIADB DATA WITH PHYSICAL COPIES 14
1.7.1. Performing a physical online backup by using mariabackup 14
1.7.2. Restoring data by using mariabackup 15
1.7.3. Performing a file system backup on a MariaDB server 16
1.7.4. Restoring a file system backup on a MariaDB server 17
1.8. REPLICATING MARIADB WITH GALERA 18
1.8.1. Introduction to MariaDB Galera Cluster 18
1.8.2. Components to build MariaDB Galera Cluster 19
1.8.3. Deploying MariaDB Galera Cluster 20
1.8.4. Checking the status of a MariaDB Galera cluster 21
1.8.5. Adding a new node to MariaDB Galera Cluster 24
1.8.6. Restarting MariaDB Galera Cluster 24
1.9. MIGRATING A MARIADB INSTANCE FROM A PREVIOUS RHEL VERSION TO MARIADB 10.11 ON RHEL 10 25

CHAPTER 2. USING MYSQL 27
2.1. INSTALLING MYSQL 27
2.2. USING CONTAINERS TO RUN MULTIPLE MARIADB AND MYSQL INSTANCES ON A SINGLE HOST 27
2.3. CONFIGURING NETWORK ACCESS TO MYSQL 29
2.4. SETTING UP TLS ENCRYPTION ON A MYSQL SERVER 29
2.4.1. Placing the CA certificate, server certificate, and private key on the MySQL server 29
2.4.2. Configuring TLS encryption on a MySQL server 30
2.4.3. Requiring TLS encrypted connections for specific user accounts on a MySQL server 32
2.5. CONFIGURING THE MYSQL CLIENT TO USE TLS ENCRYPTION BY DEFAULT 33
2.6. BACKING UP AND RESTORING MYSQL DATA WITH LOGICAL DUMPS 34
2.6.1. Performing a logical backup by using mysqldump 34
2.6.2. Restoring MySQL data from a dump in SQL format 35
2.7. BACKING UP AND RESTORING MYSQL DATA WITH PHYSICAL COPIES 36
2.7.1. Performing a file system backup on a MySQL server 36
2.7.2. Restoring a file system backup on a MySQL server 37
2.8. REPLICATING MYSQL WITH TLS ENCRYPTION 38
2.8.1. Configuring a MySQL source server 38
2.8.2. Configuring a MySQL replica server 39
2.8.3. Creating a replication user on the MySQL source server 40
2.8.4. Connecting the MySQL replica server to the source server 41
2.8.5. Verifying replication on a MySQL server 42
2.9. MIGRATING A MYSQL INSTANCE FROM A PREVIOUS RHEL VERSION TO MYSQL 8.4 ON RHEL 10 43

CHAPTER 3. USING POSTGRESQL **45**
3.1. INSTALLING POSTGRESQL 45
3.2. USING CONTAINERS TO RUN MULTIPLE POSTGRESQL INSTANCES ON A SINGLE HOST 45
3.3. CREATING POSTGRESQL USERS 46
3.4. CONFIGURING POSTGRESQL 49
3.5. CONFIGURING TLS ENCRYPTION ON A POSTGRESQL SERVER 49
3.6. BACKING UP AND RESTORING POSTGRESQL DATA WITH LOGICAL DUMPS 51
3.6.1. Advantages and disadvantages of an SQL dump 51
3.6.2. Backing up a single PostgreSQL database by using pg_dump 51
3.6.3. Restoring a single PostgreSQL database by using pg_dump 52
3.6.4. Backing up all databases on a PostgreSQL server by using pg_dumpall 53
3.6.5. Restoring all databases on a PostgreSQL server by using pg_dumpall 53
3.6.6. Handling SQL errors during restore 53
3.7. BACKING UP AND RESTORING POSTGRESQL DATA WITH PHYSICAL COPIES 54
3.7.1. Performing a file system backup on a PostgreSQL server 54
3.7.2. Restoring a file system backup on a PostgreSQL server 55
3.8. BACKING UP AND RESTORING POSTGRESQL DATA WITH CONTINUOUS ARCHIVING 56
3.8.1. Advantages and disadvantages of continuous archiving 57
3.8.2. Setting up WAL archiving 57
3.8.3. Making a base backup 58
3.8.4. Restoring the database by using a continuous archive backup 60
3.9. DIRECTLY TRANSFERRING A POSTGRESQL DATABASE FROM ONE SERVER TO ANOTHER 61
3.10. MIGRATING A POSTGRESQL INSTANCE FROM A PREVIOUS RHEL VERSION TO POSTGRESQL ON RHEL 10 61
3.10.1. Migrating to PostgreSQL on RHEL 10 by using the backup and restore method 62
3.10.2. Migrating PostgreSQL 13 from a previous RHEL version to PostgreSQL 16 on RHEL 10 by using pg_update 63

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the Jira website.
2. Click **Create** in the top navigation bar.
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

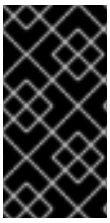
CHAPTER 1. USING MARIADB

MariaDB Server is a high-performance, open source relational database management system (RDBMS). Built on the MySQL technology, it offers a powerful SQL interface for data access and includes advanced features, such as support for multiple storage engines.

Learn how to install and configure MariaDB on a RHEL system, how to back up MariaDB data, how to migrate from an earlier MariaDB version, and how to replicate a database by using the MariaDB Galera Cluster.

1.1. INSTALLING MARIADB

RHEL 10 provides MariaDB 10.11 as the initial version of the Application Stream, which you can install as an RPM package. Additional MariaDB versions are provided as modules with a shorter life cycle in minor releases of RHEL 10.



IMPORTANT

By design, you can install only one version (stream) of the same module and, because of conflicting RPM packages, you cannot install MariaDB and MySQL on the same host. As an alternative, you can run the database server services in a container. See Using containers to run multiple MariaDB and MySQL instances on a single host.

Procedure

1. Install MariaDB server packages:

```
# dnf install mariadb-server
```

2. Enable and start the **mariadb** service:

```
# systemctl enable --now mariadb.service
```

1.2. USING CONTAINERS TO RUN MULTIPLE MARIADB AND MYSQL INSTANCES ON A SINGLE HOST

If you install MariaDB or MySQL from packages, you can only run one of these services and only a single version of it on the same host. As an alternative, you can run the services in a container.

You can configure the following scenarios:

- You want to run multiple instances of MariaDB or MySQL on the same host.
- You want to run both MariaDB and MySQL on the same host.

Prerequisites

- The **podman** package is installed.

Procedure

1. Authenticate to the **registry.redhat.io** registry by using your Red Hat Customer Portal account:

```
# podman login registry.redhat.io
```

Skip this step if you are already logged in to the container registry.

2. Start the containers you want to use:

   - MariaDB 10.11:

     ```
     $ podman run -d --name <container_name_1> -e
     MYSQL_ROOT_PASSWORD=<password> -p <host_port_1>:3306 rhel10/mariadb-
     1011
     ```

     For more information about the usage of this container image, see the Red Hat Ecosystem Catalog.

   - MySQL 8.4:

     ```
     $ podman run -d --name <container_name_2> -e
     MYSQL_ROOT_PASSWORD=<password> -p <host_port_2>:3306 rhel10/mysql-84
     ```

     For more information about the usage of this container image, see the Red Hat Ecosystem Catalog.

IMPORTANT

The container names and host ports of the two database servers must differ.

3. To ensure that clients can access the database server on the network, open the host ports in the firewall:

```
# firewall-cmd --permanent --add-port={<host_port_1>/tcp,<host_port_2>/tcp,...}
# firewall-cmd --reload
```

Verification

1. Connect to the database server and log in as root:

```
# mysql -u root -p -h localhost -P <host_port> --protocol tcp
```

2. Optional: Display information about the running containers:

```
$ podman ps
```

Additional resources

- Building, running, and managing containers
- Browse containers in the Red Hat Ecosystem Catalog

1.3. CONFIGURING NETWORK ACCESS TO MARIADB

You must configure MariaDB to listen on network interfaces and open firewall ports to allow remote client access to the database server.

Procedure

1. Edit the **[mysqld]** section of the **/etc/my.cnf.d/mariadb-server.cnf** file. You can set the following configuration directives:
   - **bind-address** - is the address on which the server listens. Possible options are:
     - a host name
     - an IPv4 address
     - an IPv6 address
   - **skip-networking** - controls whether the server listens for TCP/IP connections. Possible values are:
     - 0 - to listen for all clients
     - 1 - to listen for local clients only
   - **port** - the port on which **MariaDB** listens for TCP/IP connections.
2. To ensure that clients can access the database server on the network, open the port in the firewall:

```
# firewall-cmd --permanent --add-service=mysql
# firewall-cmd --reload
```

3. Restart the **mariadb** service:

```
# systemctl restart mariadb.service
```

1.4. SETTING UP TLS ENCRYPTION ON A MARIADB SERVER

By default, MariaDB uses unencrypted connections. For secure connections, enable TLS support on the MariaDB server and configure your clients to establish encrypted connections.

1.4.1. Placing the CA certificate, server certificate, and private key on the MariaDB server

Before you can enable TLS encryption in the MariaDB server, store the certificate authority (CA) certificate, the server certificate, and the private key on the MariaDB server.

Prerequisites

- The following files in Privacy Enhanced Mail (PEM) format have been copied to the server:
  - The private key of the server: **server.example.com.key.pem**
  - The server certificate: **server.example.com.crt.pem**
  - The Certificate Authority (CA) certificate: **ca.crt.pem**

For details about creating a private key and certificate signing request (CSR), as well as about requesting a certificate from a CA, see your CA's documentation.

Procedure

1. Store the CA and server certificates in the **/etc/pki/tls/certs/** directory:

```
# mv <path>/server.example.com.crt.pem /etc/pki/tls/certs/
# mv <path>/ca.crt.pem /etc/pki/tls/certs/
```

2. Set permissions on the CA and server certificate that enable the MariaDB server to read the files:

```
# chmod 644 /etc/pki/tls/certs/server.example.com.crt.pem /etc/pki/tls/certs/ca.crt.pem
```

Because certificates are part of the communication before a secure connection is established, any client can retrieve them without authentication. Therefore, you do not need to set strict permissions on the CA and server certificate files.

3. Store the server's private key in the **/etc/pki/tls/private/** directory:

```
# mv <path>/server.example.com.key.pem /etc/pki/tls/private/
```

4. Set secure permissions on the server's private key:

```
# chmod 640 /etc/pki/tls/private/server.example.com.key.pem
# chgrp mysql /etc/pki/tls/private/server.example.com.key.pem
```

If unauthorized users have access to the private key, connections to the MariaDB server are no longer secure.

5. Restore the SELinux context:

```
#  restorecon -Rv /etc/pki/tls/
```

1.4.2. Configuring TLS encryption on a MariaDB server

By default, MariaDB uses unencrypted connections. For more secure connections, you can enable Transport Layer Security (TLS) support on the MariaDB server and configure your clients to establish encrypted connections.

Prerequisites

- You installed the MariaDB server.
- The **mariadb** service is running.
- The following files in Privacy Enhanced Mail (PEM) format exist on the server and are readable by the **mysql** user:
  - The private key of the server: **/etc/pki/tls/private/server.example.com.key.pem**
  - The server certificate: **/etc/pki/tls/certs/server.example.com.crt.pem**

- The Certificate Authority (CA) certificate **/etc/pki/tls/certs/ca.crt.pem**

- The subject distinguished name (DN) or the subject alternative name (SAN) field in the server certificate matches the server's host name.

- If the FIPS mode is enabled, clients must either support the Extended Master Secret (EMS) extension or use TLS 1.3. TLS 1.2 connections without EMS fail. For more information, see the Red Hat Knowledgebase solution TLS extension "Extended Master Secret" enforced enforced on RHEL 9.2 and later.

Procedure

1. Create the **/etc/my.cnf.d/mariadb-server-tls.cnf** file:

   a. Add the following content to configure the paths to the private key, server and CA certificate:

   ```
   [mariadb]
   ssl_key = /etc/pki/tls/private/server.example.com.key.pem
   ssl_cert = /etc/pki/tls/certs/server.example.com.crt.pem
   ssl_ca = /etc/pki/tls/certs/ca.crt.pem
   ```

   b. If you have a Certificate Revocation List (CRL), configure the MariaDB server to use it:

   ```
   ssl_crl = /etc/pki/tls/certs/example.crl.pem
   ```

   c. Optional: Reject connection attempts without encryption. To enable this feature, append:

   ```
   require_secure_transport = on
   ```

   d. Optional: Set the TLS versions the server should support. For example, to support TLS 1.2 and TLS 1.3, append:

   ```
   tls_version = TLSv1.2,TLSv1.3
   ```

   By default, the server supports TLS 1.1, TLS 1.2, and TLS 1.3.

2. Restart the **mariadb** service:

   ```
   # systemctl restart mariadb
   ```

Verification

To simplify troubleshooting, perform the following steps on the **MariaDB** server before you configure the local client to use TLS encryption:

1. Verify that MariaDB now has TLS encryption enabled:

   ```
   # mysql -u root -p -e "SHOW GLOBAL VARIABLES LIKE 'have_ssl';"
   +---------------+----------------+
   | Variable_name | Value          |
   +---------------+----------------+
   | have_ssl      | YES            |
   +---------------+----------------+
   ```

If the **have_ssl** variable is set to **yes**, TLS encryption is enabled.

2. If you configured the **MariaDB** service to only support specific TLS versions, display the **tls_version** variable:

```
# mysql -u root -p -e "SHOW GLOBAL VARIABLES LIKE 'tls_version';"
+---------------+-----------------+
| Variable_name | Value           |
+---------------+-----------------+
| tls_version   | TLSv1.2,TLSv1.3 |
+---------------+-----------------+
```

Additional resources

- Placing the CA certificate, server certificate, and private key on the MariaDB server

1.4.3. Requiring TLS encrypted connections for specific user accounts on a MariaDB server

You can configure specific MariaDB user accounts to require TLS-encrypted connections to protect sensitive data transmission.

If you cannot configure on the server that a secure transport is required for all connections (**require_secure_transport = on**), configure individual user accounts to require TLS encryption.

Prerequisites

- The **MariaDB** server has TLS support enabled.
- The user you configure to require secure transport exists.

Procedure

1. Connect as an administrative user to the **MariaDB** server:

```
# mysql -u root -p -h server.example.com
```

If your administrative user has no permissions to access the server remotely, perform the command on the **MariaDB** server and connect to **localhost**.

2. Use the **REQUIRE SSL** clause to enforce that a user must connect by using a TLS-encrypted connection:

```
MariaDB [(none)]> ALTER USER 'example'@'%' REQUIRE SSL;
```

Verification

1. Connect to the server as the **example** user by using TLS encryption:

```
# mysql -u example -p -h server.example.com --ssl
...
MariaDB [(none)]>
```

If no error is shown and you have access to the interactive **MariaDB** console, the connection with TLS succeeds.

2. Attempt to connect as the **example** user with TLS disabled:

```
# mysql -u example -p -h server.example.com --skip-ssl
ERROR 1045 (28000): Access denied for user 'example'@'server.example.com' (using password: YES)
```

The server rejected the login attempt because TLS is required for this user but disabled (**--skip-ssl**).

1.5. CONFIGURING THE MARIADB CLIENT TO USE TLS ENCRYPTION BY DEFAULT

On RHEL, you can globally configure that the MariaDB client uses TLS encryption and verifies that the Common Name (CN) in the server certificate matches the hostname the user connects to. This prevents man-in-the-middle attacks.

Prerequisites

- The MariaDB server has TLS support enabled.
- If the certificate authority (CA) that issued the server's certificate is not trusted by RHEL, the CA certificate has been copied to the client.
- If the FIPS mode is enabled, this client supports the Extended Master Secret (EMS) extension or uses TLS 1.3. TLS 1.2 connections without EMS fail. For more information, see the Red Hat Knowledgebase solution TLS extension "Extended Master Secret" enforced on RHEL 9.2 and later.

Procedure

1. If RHEL does not trust the CA that issued the server's certificate:
   a. Copy the CA certificate to the **/etc/pki/ca-trust/source/anchors/** directory:

```
# cp <path>/ca.crt.pem /etc/pki/ca-trust/source/anchors/
```

   b. Set permissions that enable all users to read the CA certificate file:

```
# chmod 644 /etc/pki/ca-trust/source/anchors/ca.crt.pem
```

   c. Rebuild the CA trust database:

```
# update-ca-trust
```

2. Create the **/etc/my.cnf.d/mariadb-client-tls.cnf** file with the following content:

```
[client-mariadb]
ssl
ssl-verify-server-cert
```

These settings define that the MariaDB client uses TLS encryption (**ssl**) and that the client compares the hostname with the CN in the server certificate (**ssl-verify-server-cert**).

Verification

- Connect to the server by using the hostname, and display the server status:

```
# mysql -u root -p -h server.example.com -e status
...
SSL:        Cipher in use is TLS_AES_256_GCM_SHA384
```

If the **SSL** entry contains **Cipher in use is…**, the connection is encrypted.

Note that the user you use in this command has permissions to authenticate remotely.

If the hostname you connect to does not match the hostname in the TLS certificate of the server, the **ssl-verify-server-cert** parameter causes the connection to fail. For example, if you connect to **localhost**:

```
# mysql -u root -p -h localhost -e status
ERROR 2026 (HY000): SSL connection error: Validation of SSL server certificate failed
```

1.6. BACKING UP AND RESTORING MARIADB DATA WITH LOGICAL DUMPS

A logical backup of MariaDB data consists of the SQL statements necessary to restore the data. The advantage of logical backup over physical backup is that the data can be restored on other hardware configurations and MariaDB versions.

1.6.1. Performing a logical backup by using mariadb-dump

You can use the **mariadb-dump** utility to back up databases while the MariaDB server is running and store the exported data in an SQL file. Store the backup in a safe location to be able to recover it in data loss scenarios.

Frequent scenarios with **mariadb-dump** include:

- Backing up a single database
- Backing up multiple databases
- Backing up all databases

The **mariadb-dump** utility stores its output in a single file. If you want to back up multiple databases and require one file per database, back up each database individually.

NOTE

The **mariadb-dump** utility can back up only databases. This also includes server settings stored in the **mysql** database. However, the utility does not back up configuration files, such as **/etc/my.cnf**.

Prerequisites

- The **mariadb** service is running.
- You have credentials with permissions to back up a database, for example, the **root** account.

Procedure

- Create a consistent and comprehensive logical backup of MariaDB databases:

```
# mariadb-dump -u <username> -p --routines --events --triggers --single-transaction --result-file=backup.sql --databases <database_1> <database_2>
```

where:

-u *<username>*

Sets the username the utility uses to connect to the database server.

-p

Prompts for the password.

--routines

Includes stored procedures and functions in the backup.

--events

Includes scheduled events in the backup.

--triggers

Includes triggers in the backup.

--single-transaction

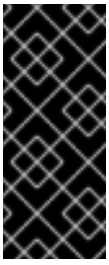
Initiates a consistent snapshot of databases with a transactional storage engine, such as InnoDB. By using a single transaction, all read operations reflect the database state at the moment the dump begins.
If you still use a non-transactional storage engine, such as MyISAM, use the **--lock-tables** option instead of **--single-transaction** to ensure a consistent backup.

--result-file=*<output_file>*

Defines in which file **mariadb-dump** stores the output.

--databases *<list_of_databases>*

Defines the databases to back up. Alternatively, to back up all databases at once, use the **--all-databases** option.



IMPORTANT

A backup of a database contains only the data from that database. It does not include MariaDB user accounts or other server settings. MariaDB stores this essential security and system information in a separate **mysql** system database. Therefore, you must back up **mysql** as well if you need to preserve these settings.

Verification

- Restore the backup in a sandbox environment and ensure that the data is correct.

1.6.2. Restoring MariaDB data from a dump in SQL format

If you backed up one or multiple databases to an SQL file, you can use this file to re-create the database structure and its data.

Prerequisites

- The **mariadb** service is running.
- You have credentials with permissions to restore data, for example, the **root** account.

Procedure

1. If the database you want to restore already exists and your SQL file does not contain **DROP TABLE IF EXISTS** statements, you must manually remove the tables or the entire database:
   - To remove a table, enter:

     ```
     # mariadb -u root -p -e "DROP TABLE <database>.<table>;"
     ```

     Repeat this command for every table that the SQL file will re-create.
   - To remove a database, enter:

     ```
     # mariadb -u root -p -e "DROP DATABASE <database>;"
     ```

     Repeat this command for every database that the SQL file will re-create.
2. Import the SQL file:

   ```
   # mariadb -u root -p < backup.sql"
   ```

Verification

- Connect to a MariaDB database and query data, for example:

  ```
  # mariadb -u root -p <database> -e "*SELECT * FROM <table>;"
  ```

1.7. BACKING UP AND RESTORING MARIADB DATA WITH PHYSICAL COPIES

A physical backup of MariaDB data contains files and directories that store the content. This method is typically faster and smaller in size.

1.7.1. Performing a physical online backup by using mariabackup

You can create physical online backups of your MariaDB server by using the **mariabackup** utility to backup InnoDB, Aria, and MyISAM tables while the server is running. The utility supports full backup capability for MariaDB server, which includes encrypted and compressed data.

Prerequisites

- The **mariadb-backup** package is installed on the system:
- You must provide **mariabackup** with credentials for the user under which the backup will be run. You can provide the credentials either on the command line or by a configuration file.
- Users of **mariabackup** must have the **RELOAD**, **LOCK TABLES**, and **REPLICATION CLIENT** privileges.

Procedure

- Use one of the following options to create a backup:
  - To create a backup while providing credentials on the command line, enter:

    ```
    $ mariabackup --backup --target-dir <backup_directory> --user <backup_user> --password <backup_passwd>
    ```

    The **target-dir** option defines the directory where the backup files are stored. If you want to perform a full backup, the target directory must be empty or not exist.

    The **user** and **password** options allow you to configure the user name and the password.

  - To create a backup with credentials set in a configuration file:

    i. Create a configuration file in the **/etc/my.cnf.d/** directory, for example, **/etc/my.cnf.d/mariabackup.cnf**.

    ii. Add the following content to the file:

    ```
    [mysqld]
    user=<backup_username>
    password=<password>
    ```

    iii. Perform the backup:

    ```
    $ mariabackup --backup --target-dir <backup_directory>
    ```

Additional resources

- Full Backup and Restore with mariabackup

1.7.2. Restoring data by using mariabackup

If you have a MariaDB backup created by the **mariabackup** utility, you can use the same utility to restore the data.

Prerequisites

- The **mariadb** service is stopped.
- The data directory is empty.
- Users of **mariabackup** must have the **RELOAD**, **LOCK TABLES**, and **REPLICATION CLIENT** privileges.

Procedure

1. Use one of the following options to restore the data:
   - To restore data from the backup in the **/var/mariadb/backup/** directory and keep the original backup files, enter:

     ```
     $ mariabackup --copy-back --target-dir=/var/mariadb/backup/
     ```

   - To restore data from the backup in the **/var/mariadb/backup/** directory and remove the original backup files, enter:

     ```
     $ mariabackup --move-back --target-dir=/var/mariadb/backup/
     ```

2. Fix the file permissions. For example, to recursively change ownership of the files to the **mysql** user and group, enter:

   ```
   # chown -R mysql:mysql /var/lib/mysql/
   ```

   When restoring a database, **mariabackup** preserves the file and directory privileges of the backup. However, **mariabackup** writes the files to disk as the user and group restoring the database. Therefore, after restoring a backup, you must adjust the owner of the data directory to match the user and group for the MariaDB server.

3. Start the **mariadb** service:

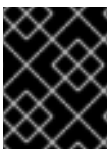
   ```
   # systemctl start mariadb.service
   ```

Additional resources

- Full Backup and Restore with mariabackup

1.7.3. Performing a file system backup on a MariaDB server

A file system level backup is a fast way to back up a complete MariaDB instance. This method requires a shutdown of the **mariadb** service for data consistency.



IMPORTANT

A file system-level backup is specific to an architecture and MariaDB version. You cannot restore data backed up by this method on a different architecture or MariaDB version.

Procedure

1. Stop the **mariadb** service:

   ```
   # systemctl stop mariadb.service
   ```

2. Create a backup directory, for example:

   ```
   # mkdir -p /root/mariadb-backup/{data,config}/
   ```

3. Back up the data directory:

```
# cp -rp /var/lib/mysql/ /root/mariadb-backup/data/
```

4. Back up the configuration files:

```
# cp -rp /etc/my.cnf /etc/my.cnf.d/ /root/mariadb-backup/config/
```

5. Start the **mariadb** service:

```
# systemctl start mariadb.service
```

1.7.4. Restoring a file system backup on a MariaDB server

If your MariaDB instance has been corrupted, and you previously performed a file system backup that includes the data directory and the configuration files, you can restore the instance from this backup.

Prerequisites

- You performed a file system backup on a MariaDB server .
- The target server must meet the following conditions of the backup source:
  - The MariaDB version must be identical or higher.
  - The system architecture must be identical.

Procedure

1. Stop the **mariadb** service:

```
# systemctl stop mariadb.service
```

2. Remove the current **/var/lib/mysql/** directory:

```
# rm -rf /var/lib/mysql/
```

3. Restore the data directory from your backup:

```
# cp -rp /root/mariadb-backup/data/mysql/ /var/lib/
```

4. Ensure the correct ownership of the **/var/lib/mysql/** directory:

```
# chown -R mysql:mysql /var/lib/mysql/
```

5. Restore the SELinux context of the **/var/lib/mysql/** directory:

```
# restorecon -Rv /var/lib/mysql/
```

6. Remove the current configuration files:

```
# rm -rf /etc/my.cnf /etc/my.cnf.d/
```

7. Restore the configuration files from your backup:

```
# cp -rp /root/mariadb-backup/config/my.cnf /root/mariadb-backup/config/my.cnf.d/
/etc/
```

8. Ensure the correct ownership of the configuration files:

```
# chown -R root:root /etc/my.cnf /etc/my.cnf.d/
```

9. Restore the SELinux context of the configuration files:

```
# restorecon -Rv /etc/my.cnf /etc/my.cnf.d/
```

10. Start the **mariadb** service:

```
# systemctl start mariadb.service
```

Verification

- Connect to a MariaDB database and query data, for example:

```
# mariadb -u root -p <database> -e "*SELECT * FROM <table>;"
```

1.8. REPLICATING MARIADB WITH GALERA

You can replicate a MariaDB database using the Galera solution to create a synchronous multi-source cluster for high availability and data consistency.

WARNING

Replication itself is not a sufficient backup solution. Replication protects source servers against hardware failures, but it does not ensure protection against data loss.

1.8.1. Introduction to MariaDB Galera Cluster

MariaDB Galera Cluster provides synchronous multi-source replication that allows all nodes to be writable and ensures data consistency across the cluster.

The interface between Galera replication and a MariaDB database is defined by the write set replication API (**wsrep API**).

The main features of MariaDB Galera Cluster are:

- Synchronous replication
- Active-active multi-source topology

- Read and write to any cluster node
- Automatic membership control, failed nodes drop from the cluster
- Automatic node joining
- Parallel replication on row level
- Direct client connections: users can log on to the cluster nodes, and work with the nodes directly while the replication runs

Synchronous replication means that a server replicates a transaction at commit time by broadcasting the write set associated with the transaction to every node in the cluster. The client (user application) connects directly to the Database Management System (DBMS), and experiences behavior that is similar to native MariaDB.

Synchronous replication guarantees that a change that happened on one node in the cluster happens on other nodes in the cluster at the same time.

Therefore, synchronous replication has the following advantages over asynchronous replication:

- No delay in propagation of the changes between particular cluster nodes
- All cluster nodes are always consistent
- The latest changes are not lost if one of the cluster nodes crashes
- Transactions on all cluster nodes are executed in parallel
- Causality across the whole cluster

Additional resources

- What is Galera replication
- What is MariaDB Galera Cluster
- Getting started with MariaDB Galera Cluster

1.8.2. Components to build MariaDB Galera Cluster

Before deploying a functional, synchronously replicated MariaDB Galera Cluster, you must first install and understand the function of the core software components, specifically the MariaDB Server, the Galera Replication library, and the supporting Galera packages.

To build **MariaDB Galera Cluster**, you must install the following packages on your system:

- **mariadb-server-galera** - contains support files and scripts for **MariaDB Galera Cluster**.
- **mariadb-server** - is patched by **MariaDB** upstream to include the write set replication API (**wsrep API**). This API provides the interface between **Galera** replication and **MariaDB**.
- **galera** - is patched by **MariaDB** upstream to add full support for **MariaDB**. The **galera** package contains the following:
  - **Galera Replication Library** provides the whole replication functionality.

- The **Galera Arbitrator** utility can be used as a cluster member that participates in voting in split-brain scenarios. However, **Galera Arbitrator** cannot participate in the actual replication.
- **Galera Systemd service** and **Galera wrapper script** which are used for deploying the Galera Arbitrator utility. RHEL 10 provides the upstream version of these files, located at **/usr/lib/systemd/system/garbd.service** and **/usr/sbin/garb-systemd**.

Additional resources

- Galera Replication Library
- Galera Arbitrator

1.8.3. Deploying MariaDB Galera Cluster

You can deploy MariaDB Galera Cluster by installing the required packages, configuring cluster settings, and bootstrapping the first node to create a new cluster.

Prerequisites

- All of the nodes in the cluster have TLS set up.
- All certificates on all nodes must have the **Extended Key Usage** field set to:

```
TLS Web Server Authentication, TLS Web Client Authentication
```

Procedure

1. Install the MariaDB Galera Cluster packages:

```
# dnf install mariadb-server-galera
```

   As a result, the following packages are installed together with their dependencies:

   - **mariadb-server-galera**
   - **mariadb-server**
   - **galera**
     For more information about these packages required to build MariaDB Galera Cluster, see Components to build MariaDB Cluster .

2. Update the MariaDB server replication configuration before the system is added to a cluster for the first time. The default configuration is distributed in the **/etc/my.cnf.d/galera.cnf** file. Before deploying MariaDB Galera Cluster, set the **wsrep_cluster_address** option in the **/etc/my.cnf.d/galera.cnf** file on all nodes to start with the following string:

```
gcomm://
```

   - For the initial node, it is possible to set **wsrep_cluster_address** as an empty list:

```
wsrep_cluster_address="gcomm://"
```

- For all other nodes, set **wsrep_cluster_address** to include an address to any node which is already a part of the running cluster. For example:

```
wsrep_cluster_address="gcomm://10.0.0.10"
```

For more information about how to set Galera Cluster address, see Galera Cluster Address .

3. Enable the **wsrep** API on every node by setting the **wsrep_on=1** option in the **/etc/my.cnf.d/galera.cnf** configuration file.

4. Add the **wsrep_provider_options** variable to the Galera configuration file with the TLS keys and certificates. For example:

```
wsrep_provider_options="socket.ssl_cert=/etc/pki/tls/certs/source.crt;socket.ssl_key=/etc/pki/tls/
rivate/source.key;socket.ssl_ca=/etc/pki/tls/certs/ca.crt"
```

5. Bootstrap a first node of a new cluster by running the following wrapper on that node:

```
# galera_new_cluster
```

This wrapper ensures that the MariaDB server daemon (**mariadbd**) runs with the **--wsrep-new-cluster** option. This option provides the information that there is no existing cluster to connect to. Therefore, the node creates a new UUID to identify the new cluster.

NOTE

The **mariadb** service supports a systemd method for interacting with multiple MariaDB server processes. Therefore, in cases with multiple running MariaDB servers, you can bootstrap a specific instance by specifying the instance name as a suffix:

```
# galera_new_cluster mariadb@node1
```

6. Connect other nodes to the cluster by running the following command on each of the nodes:

```
# systemctl start mariadb
```

As a result, the node connects to the cluster, and synchronizes itself with the state of the cluster.

Verification

- See Checking the status of a MariaDB Galera cluster .

Additional resources

- Getting started with MariaDB Galera Cluster

1.8.4. Checking the status of a MariaDB Galera cluster

It is important to monitor and ensure the health, performance, and synchronization of a MariaDB Galera cluster. For that, you can query status variables on each node to monitor the node and the cluster.

To check the status of a MariaDB Galera cluster, you can use the following queries:

- Display the number of nodes in the cluster:

```
# mysql -u root -p -e 'show status like "wsrep_cluster_size";'
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 4     |
+--------------------+-------+
```

- Display the node's cluster component status:

```
# mysql -u root -p -e 'show status like "wsrep_cluster_status";'
+----------------------+---------+
| Variable_name        | Value   |
+----------------------+---------+
| wsrep_cluster_status | Primary |
+----------------------+---------+
```

  The value of the **wsrep_cluster_status** variable indicates the status of the cluster component the current node belongs to. Possible values are:

  - **Primary**: The cluster is functioning normally. A quorum is present. In a healthy cluster, all nodes report **Primary**.
  - **Non-primary**: The node has lost the connection to the primary component of the cluster and is no longer part of the active cluster. However, the node still can serve read queries but cannot process write operations.
  - **Disconnected**: The node is not connected to any cluster component. Consequently, it cannot accept queries and is not replicating any data.

- Display the node's status:

```
# mysql -u root -p -e 'show status like "wsrep_local_state_comment";'
+---------------------------+--------+
| Variable_name             | Value  |
+---------------------------+--------+
| wsrep_local_state_comment | Synced |
+---------------------------+--------+
```

  The following are frequent values of the **wsrep_local_state_comment** variable:

  - **Synced**: The node is fully synchronized within the cluster and actively participating in replication.
  - **Desynced**: The node is still part of the cluster but it is primarily busy with the state transfer.
  - **Joining**: The node is in the process of joining a cluster.
  - **Joined**: The node has successfully joined a cluster. It can receive and apply write sets from the cluster.

- **Donor**: The node currently provides a State Snapshot Transfer (SST) to a joining node. When a new node joins and requires a full state transfer, the cluster selects an existing node to send the necessary data.

- Check whether the node accepts write sets from the cluster:

```
# mysql -u root -p -e 'show status like "wsrep_ready";'
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wsrep_ready   | ON    |
+---------------+-------+
```

When the **wsrep_ready** variable is **ON**, the node has successfully initialized its components and is connected to a cluster. Additionally, the node is synchronized or has reached a state where it can serve queries.

- Check whether the node has network connectivity with other hosts:

```
# mysql -u root -p -e 'show status like "wsrep_connected";'
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| wsrep_connected | ON    |
+-----------------+-------+
```

The **ON** value means that node has connectivity to at least one member in the cluster.

- Display the average size of the local received queue for write sets since the last **FLUSH STATUS** command or since the server started:

```
# mysql -u root -p -e 'show status like "wsrep_local_recv_queue_avg";'
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| wsrep_local_recv_queue_avg | 0.012 |
+----------------------------+-------+
```

A value near 0 is the ideal state and indicates that the node continues applying write sets as they are received. A persistently high or growing value can be an indicator of performance bottlenecks, such as slow disk I/O.

- Display the flow control status:

```
# mysql -u root -p -e 'show status like "wsrep_flow_control_paused";'
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| wsrep_flow_control_paused | 0     |
+--------------------------+-------+
```

This variable represents the fraction of time a node has been paused and is unable to process new incoming transactions because its local receive queue was too full, triggering flow control. A value close to 0 indicates the node continues with the replication workload efficiently. A value

approaching 1.0 means that the node frequently or constantly encounters difficulty in applying write sets and can be a bottleneck for the cluster.

If the node is frequently pausing, you can adjust the **wsrep_slave_threads** parameter in the **/etc/my.cnf.d/galera.cnf** file.

- Display the average distance between the lowest and highest sequence numbers the node can apply in parallel:

```
# mysql -u root -p -e 'show status like "wsrep_cert_deps_distance";'
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| wsrep_cert_deps_distance | 1     |
+--------------------------+-------+
```

A higher value indicates a greater degree of parallelism. It is the optimal value you can use in the **wsrep_slave_threads** parameter in the **/etc/my.cnf.d/galera.cnf** file.

1.8.5. Adding a new node to MariaDB Galera Cluster

You can add a new node to your MariaDB Galera Cluster or reconnect an existing one by configuring the cluster address in the node's configuration file.

Procedure

- On the particular node, provide an address to one or more existing cluster members in the **wsrep_cluster_address** option within the **[mariadb]** section of the **/etc/my.cnf.d/galera.cnf** configuration file :

```
[mariadb]
wsrep_cluster_address="gcomm://192.168.0.1"
```

When a new node connects to one of the existing cluster nodes, it is able to see all nodes in the cluster.

However, preferably list all nodes of the cluster in **wsrep_cluster_address**.

As a result, any node can join a cluster by connecting to any other cluster node, even if one or more cluster nodes are down. When all members agree on the membership, the cluster's state is changed. If the new node's state is different from the state of the cluster, the new node requests either an Incremental State Transfer (IST) or a State Snapshot Transfer (SST) to ensure consistency with the other nodes.

Additional resources

- Getting started with MariaDB Galera Cluster
- Introduction to State Snapshot Transfers
- Securing Communications in Galera Cluster

1.8.6. Restarting MariaDB Galera Cluster

If you shut down all nodes at the same time, you stop the cluster, and the running cluster no longer exists. However, the cluster's data still exist.

To restart the cluster, bootstrap a first node as described in Deploying MariaDB Galera Cluster

WARNING

If the cluster is not bootstrapped, and **mariadb** on the first node is started with only the **systemctl start mariadb** command, the node tries to connect to at least one of the nodes listed in the **wsrep_cluster_address** option in the **/etc/my.cnf.d/galera.cnf** file. If no nodes are currently running, the restart fails.

Additional resources

- Getting started with MariaDB Galera Cluster

1.9. MIGRATING A MARIADB INSTANCE FROM A PREVIOUS RHEL VERSION TO MARIADB 10.11 ON RHEL 10

RHEL 10 provides MariaDB 10.11. If you run a MariaDB instance on a previous RHEL version, you can set up RHEL 10 on a new host and migrate the instance to it.

Prerequisites

- You set up RHEL 10 on a new host.
- You performed a file system backup of the MariaDB instance on the RHEL 8 or RHEL 9 host.

Procedure

1. Install the **mariadb-server** package:

```
# dnf install mariadb-server
```

2. Stop the service if it is already running:

```
# systemctl stop mariadb.service
```

3. Copy the content of the **/var/lib/mysql/** directory from the previous host to the same location on the RHEL 10 host.
4. Copy the configuration files from the previous host to the **/etc/my.cnf.d/** directory, and ensure that the files includes only options valid for MariaDB 10.11. For details, see the upstream documentation.
5. Restore the SELinux context:

```
# restorecon -rv /var/lib/mysql/
# restorecon -rv /etc/my.cnf.d/
```

6. Ensure the correct ownership of **/var/lib/mysql/** and its subdirectories:

```
# chown -R mysql:mysql /var/lib/mysql/
```

7. Enable and start the **mariadb** service:

```
# systemctl enable --now mariadb.service
```

When the service starts, MariaDB automatically checks, repairs, and updates internal tables.

Verification

- Establish a connection to the MariaDB server:

```
# mysql -u root -p -h <hostname>
```

CHAPTER 2. USING MYSQL

MySQL server is a high-performance, open source relational database management system (RDBMS). It offers an SQL interface for data access and includes advanced features, such as support for multiple storage engines.

Learn how to install and configure MySQL on a RHEL system, how to back up MySQL data, how to migrate from an earlier MySQL version, and how to replicate a MySQL database.

2.1. INSTALLING MYSQL

RHEL 10 provides MySQL 8.4 as the initial version of the Application Stream, which can install as an RPM package. Additional MySQL versions are provided as modules with a shorter life cycle in minor releases of RHEL 10.

NOTE

By design, you can install only one version (stream) of the same module and, because of conflicting RPM packages, you cannot install MariaDB and MySQL on the same host. As an alternative, you can run the database server services in a container. See Using containers to run multiple MariaDB and MySQL instances on a single host.

Procedure

1. Install MySQL server packages:

```
# dnf install mysql8.4-server
```

2. Enable and start the **mysqld** service:

```
# systemctl enable --now mysqld.service
```

3. Improve the security after the installation:

```
$ mysql_secure_installation
```

The command launches a fully interactive script, which prompts for each step in the process. The script enables you to improve security in the following ways:

- Setting a password for root accounts
- Removing anonymous users
- Disallowing remote root logins (outside the local host)

2.2. USING CONTAINERS TO RUN MULTIPLE MARIADB AND MYSQL INSTANCES ON A SINGLE HOST

If you install MariaDB or MySQL from packages, you can only run one of these services and only a single version of it on the same host. As an alternative, you can run the services in a container.

You can configure the following scenarios:

- You want to run multiple instances of MariaDB or MySQL on the same host.
- You want to run both MariaDB and MySQL on the same host.

Prerequisites

- The **podman** package is installed.

Procedure

1. Authenticate to the **registry.redhat.io** registry by using your Red Hat Customer Portal account:

```
# podman login registry.redhat.io
```

   Skip this step if you are already logged in to the container registry.

2. Start the containers you want to use:

   - MariaDB 10.11:

```
$ podman run -d --name <container_name_1> -e
MYSQL_ROOT_PASSWORD=<password> -p <host_port_1>:3306 rhel10/mariadb-
1011
```

     For more information about the usage of this container image, see the Red Hat Ecosystem Catalog.

   - MySQL 8.4:

```
$ podman run -d --name <container_name_2> -e
MYSQL_ROOT_PASSWORD=<password> -p <host_port_2>:3306 rhel10/mysql-84
```

     For more information about the usage of this container image, see the Red Hat Ecosystem Catalog.

IMPORTANT

The container names and host ports of the two database servers must differ.

3. To ensure that clients can access the database server on the network, open the host ports in the firewall:

```
# firewall-cmd --permanent --add-port={<host_port_1>/tcp,<host_port_2>/tcp,...}
# firewall-cmd --reload
```

Verification

1. Connect to the database server and log in as root:

```
# mysql -u root -p -h localhost -P <host_port> --protocol tcp
```

2. Optional: Display information about the running containers:

```
$ podman ps
```

Additional resources

- Building, running, and managing containers
- Browse containers in the Red Hat Ecosystem Catalog

2.3. CONFIGURING NETWORK ACCESS TO MYSQL

You must configure MySQL to listen on network interfaces and open firewall ports to allow remote client access to the database server.

Procedure

1. Edit the **mysqld** section of the **/etc/my.cnf.d/mysql-server.cnf** file. You can set the following configuration directives:
   - **bind-address** - is the address on which the server listens. Possible options are:
     - a host name
     - an IPv4 address
     - an IPv6 address
   - **skip-networking** - controls whether the server listens for TCP/IP connections. Possible values are:
     - 0 - to listen for all clients
     - 1 - to listen for local clients only
   - **port** - the port on which MySQL listens for TCP/IP connections.
2. To ensure that clients can access the database server on the network, open the port in the firewall:

```
# firewall-cmd --permanent --add-service=mysql
# firewall-cmd --reload
```

3. Restart the **mysqld** service:

```
# systemctl restart mysqld.service
```

2.4. SETTING UP TLS ENCRYPTION ON A MYSQL SERVER

By default, MySQL uses unencrypted connections. For secure connections, enable TLS support on the MySQL server and configure your clients to establish encrypted connections.

2.4.1. Placing the CA certificate, server certificate, and private key on the MySQL server

Before you can enable TLS encryption on the MySQL server, store the certificate authority (CA) certificate, the server certificate, and the private key on the MySQL server.

Prerequisites

- The following files in Privacy Enhanced Mail (PEM) format have been copied to the server:
  - The private key of the server: **server.example.com.key.pem**
  - The server certificate: **server.example.com.crt.pem**
  - The Certificate Authority (CA) certificate: **ca.crt.pem**

  For details about creating a private key and certificate signing request (CSR), as well as about requesting a certificate from a CA, see your CA's documentation.

Procedure

1. Store the CA and server certificates in the **/etc/pki/tls/certs/** directory:

```
# mv <path>/server.example.com.crt.pem /etc/pki/tls/certs/
# mv <path>/ca.crt.pem /etc/pki/tls/certs/
```

2. Set permissions on the CA and server certificate that enable the MySQL server to read the files:

```
# chmod 644 /etc/pki/tls/certs/server.example.com.crt.pem /etc/pki/tls/certs/ca.crt.pem
```

   Because certificates are part of the communication before a secure connection is established, any client can retrieve them without authentication. Therefore, you do not need to set strict permissions on the CA and server certificate files.

3. Store the server's private key in the **/etc/pki/tls/private/** directory:

```
# mv <path>/server.example.com.key.pem /etc/pki/tls/private/
```

4. Set secure permissions on the server's private key:

```
# chmod 640 /etc/pki/tls/private/server.example.com.key.pem
# chgrp mysql /etc/pki/tls/private/server.example.com.key.pem
```

   If unauthorized users have access to the private key, connections to the MySQL server are no longer secure.

5. Restore the SELinux context:

```
# restorecon -Rv /etc/pki/tls/
```

2.4.2. Configuring TLS encryption on a MySQL server

By default, MySQL uses unencrypted connections. For more secure connections, you can enable Transport Layer Security (TLS) support on the MySQL server and configure your clients to establish encrypted connections.

Prerequisites

- You installed the MySQL server.
- The **mysqld** service is running.
- The following files in Privacy Enhanced Mail (PEM) format exist on the server and are readable by the **mysql** user:
  - The private key of the server: **/etc/pki/tls/private/server.example.com.key.pem**
  - The server certificate: **/etc/pki/tls/certs/server.example.com.crt.pem**
  - The Certificate Authority (CA) certificate **/etc/pki/tls/certs/ca.crt.pem**
- The subject distinguished name (DN) or the subject alternative name (SAN) field in the server certificate matches the server's host name.

Procedure

1. Create the **/etc/my.cnf.d/mysql-server-tls.cnf** file:
   a. Add the following content to configure the paths to the private key, server and CA certificate:

```
[mysqld]
ssl_key = /etc/pki/tls/private/server.example.com.key.pem
ssl_cert = /etc/pki/tls/certs/server.example.com.crt.pem
ssl_ca = /etc/pki/tls/certs/ca.crt.pem
```

   b. If you have a Certificate Revocation List (CRL), configure the MySQL server to use it:

```
ssl_crl = /etc/pki/tls/certs/example.crl.pem
```

   c. Optional: Reject connection attempts without encryption. To enable this feature, append:

```
require_secure_transport = on
```

   d. Optional: Set the TLS versions the server should support. For example, to support only TLS 1.3, append:

```
tls_version = TLSv1.3
```

   By default, the server supports TLS 1.2 and TLS 1.3.

2. Restart the **mysqld** service:

```
# systemctl restart mysqld
```

Verification

To simplify troubleshooting, perform the following steps on the MySQL server before you configure the local client to use TLS encryption:

1. Verify that MySQL now has TLS encryption enabled:

```
# mysql -u root -p -h <MySQL_server_hostname> -e "SHOW session status LIKE 'Ssl_cipher';"
+---------------+-----------------------+
| Variable_name | Value                 |
+---------------+-----------------------+
| Ssl_cipher    | TLS_AES_256_GCM_SHA384 |
+---------------+-----------------------+
```

2. If you configured the MySQL server to only support specific TLS versions, display the **tls_version** variable:

```
# mysql -u root -p -e "SHOW GLOBAL VARIABLES LIKE 'tls_version';"
+---------------+---------+
| Variable_name | Value   |
+---------------+---------+
| tls_version   | TLSv1.3 |
+---------------+---------+
```

3. Verify that the server uses the correct CA certificate, server certificate, and private key files:

```
# mysql -u root -e "SHOW GLOBAL VARIABLES WHERE Variable_name REGEXP '{caret}ssl_ca|{caret}ssl_cert|{caret}ssl_key';"
+-----------------+-------------------------------------------------+
| Variable_name   | Value                                           |
+-----------------+-------------------------------------------------+
| ssl_ca          | /etc/pki/tls/certs/ca.crt.pem                   |
| ssl_capath      |                                                 |
| ssl_cert        | /etc/pki/tls/certs/server.example.com.crt.pem   |
| ssl_key         | /etc/pki/tls/private/server.example.com.key.pem |
+-----------------+-------------------------------------------------+
```

Additional resources

- Placing the CA certificate, server certificate, and private key on the MySQL server

2.4.3. Requiring TLS encrypted connections for specific user accounts on a MySQL server

You can configure specific MySQL user accounts to require TLS-encrypted connections to protect sensitive data transmission.

If you cannot configure on the server that a secure transport is required for all connections (**require_secure_transport = on**), configure individual user accounts to require TLS encryption.

Prerequisites

- The MySQL server has TLS support enabled.
- The user you configure to require secure transport exists.
- The CA certificate is stored on the client.

Procedure

1. Connect as an administrative user to the MySQL server:

```
# mysql -u root -p -h server.example.com
```

If your administrative user has no permissions to access the server remotely, perform the command on the MySQL server and connect to **localhost**.

2. Use the **REQUIRE SSL** clause to enforce that a user must connect by using a TLS-encrypted connection:

```
MySQL [(none)]> ALTER USER 'example'@'%' REQUIRE SSL;
```

Verification

1. Connect to the server as the **example** user by using TLS encryption:

```
# mysql -u example -p -h server.example.com
...
MySQL [(none)]>
```

If no error is shown and you have access to the interactive MySQL console, the connection with TLS succeeds.

By default, the client automatically uses TLS encryption if the server provides it. Therefore, the **--ssl-ca=ca.crt.pem** and **--ssl-mode=VERIFY_IDENTITY** options are not required, but improve the security because, with these options, the client verifies the identity of the server.

2. Attempt to connect as the **example** user with TLS disabled:

```
# mysql -u example -p -h server.example.com --ssl-mode=DISABLED
ERROR 1045 (28000): Access denied for user 'example'@'server.example.com' (using password: YES)
```

The server rejected the login attempt because TLS is required for this user but disabled (**--ssl-mode=DISABLED**).

2.5. CONFIGURING THE MYSQL CLIENT TO USE TLS ENCRYPTION BY DEFAULT

On RHEL, you can globally configure that the MySQL client uses TLS encryption and verifies that the Common Name (CN) in the server certificate matches the hostname the user connects to. This prevents man-in-the-middle attacks.

Prerequisites

- The MySQL server has TLS support enabled.
- The CA certificate is stored in the **/etc/pki/tls/certs/ca.crt.pem** file on the client.

Procedure

- Create the **/etc/my.cnf.d/mysql-client-tls.cnf** file with the following content:

```
[client]
ssl-mode=VERIFY_IDENTITY
ssl-ca=/etc/pki/tls/certs/ca.crt.pem
```

These settings define that the MySQL client uses TLS encryption and that the client compares the hostname with the CN in the server certificate (**ssl-mode=VERIFY_IDENTITY**). Additionally, it specifies the path to the CA certificate (**ssl-ca**).

Verification

- Connect to the server using the hostname, and display the server status:

```
# mysql -u root -p -h server.example.com -e status
...
SSL:        Cipher in use is TLS_AES_256_GCM_SHA384
```

If the **SSL** entry contains **Cipher in use is…**, the connection is encrypted.

Note that the user you use in this command has permissions to authenticate remotely.

If the hostname you connect to does not match the hostname in the TLS certificate of the server, the **ssl-mode=VERIFY_IDENTITY** parameter causes the connection to fail. For example, if you connect to **localhost**:

```
# mysql -u root -p -h localhost -e status
ERROR 2026 (HY000): SSL connection error: error:0A000086:SSL routines::certificate verify failed
```

2.6. BACKING UP AND RESTORING MYSQL DATA WITH LOGICAL DUMPS

A logical backup of MySQL data consists of the SQL statements necessary to restore the data. The advantage of logical backup over physical backup is that the data can be restored on other hardware configurations and MySQL versions.

2.6.1. Performing a logical backup by using mysqldump

The **mysqldump** utility is a backup utility that can export one or more databases. The output of **mysqldump** typically consists of SQL statements to re-create the server table structure, populate it with data, or both.

To perform the **mysqldump** backup, you can use one of the following options:

- Back up one or more selected databases
- Back up all databases
- Back up a subset of tables from one database

Procedure

- To dump a single database, run:

```
# mysqldump [options] --databases db_name > backup-file.sql
```

- To dump multiple databases at once, run:

```
# mysqldump [options] --databases db_name1 [db_name2 ...] > backup-file.sql
```

- To dump all databases, run:

```
# mysqldump [options] --all-databases > backup-file.sql
```

- To load one or more dumped full databases back into a server, run:

```
# mysql < backup-file.sql
```

- To load a database to a remote MySQL server, run:

```
# mysql --host=remote_host < backup-file.sql
```

- To dump a subset of tables from one database, add a list of the chosen tables at the end of the **mysqldump** command:

```
# mysqldump [options] db_name [tbl_name ...] > backup-file.sql
```

- To load a subset of tables dumped from one database, run:

```
# mysql db_name < backup-file.sql
```

NOTE

The *db_name* database must exist at this point.

- To see a list of the options that **mysqldump** supports, run:

```
$ mysqldump --help
```

Additional resources

- Logical backup with mysqldump

2.6.2. Restoring MySQL data from a dump in SQL format

If you backed up one or multiple databases to an SQL file, you can use this file to re-create the database structure and its data.

Prerequisites

- The **mysqld** service is running.
- You have credentials with permissions to restore data, for example, the **root** account.

Procedure

1. If the database you want to restore already exists and your SQL file does not contain **DROP TABLE IF EXISTS** statements, you must manually remove the tables or the entire database:
   - To remove a table, enter:

     ```
     # mysql -u root -p -e "DROP TABLE <database>.<table>;"
     ```

     Repeat this command for every table that the SQL file will re-create.
   - To remove a database, enter:

     ```
     # mysql -u root -p -e "DROP DATABASE <database>;"
     ```

     Repeat this command for every database that the SQL file will re-create.
2. Import the SQL file:

   ```
   # mysql -u root -p < backup.sql"
   ```

Verification

- Connect to a MySQL database and query data, for example:

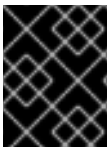
  ```
  # mysql -u root -p <database> -e "*SELECT * FROM <table>;"
  ```

2.7. BACKING UP AND RESTORING MYSQL DATA WITH PHYSICAL COPIES

A physical backup of MySQL data contains files and directories that store the content. This method is typically faster and smaller in size.

2.7.1. Performing a file system backup on a MySQL server

A file system-level backup is a fast way to back up a complete MySQL instance. This method requires a shutdown of the **mysqld** service for data consistency.



IMPORTANT

A file system-level backup is specific to an architecture and MySQL version. You cannot restore data backed up by this method on a different architecture or MySQL version.

Procedure

1. Stop the **mysqld** service:

   ```
   # systemctl stop mysqld.service
   ```
2. Create a backup directory, for example:

   ```
   # mkdir -p /root/mysqld-backup/{data,config}/
   ```

3. Back up the data directory:

```
# cp -rp /var/lib/mysql/ /root/mysqld-backup/data/
```

4. Back up the configuration files:

```
# cp -rp /etc/my.cnf /etc/my.cnf.d/ /root/mysqld-backup/config/
```

5. Start the **mysqld** service:

```
# systemctl start mysqld.service
```

2.7.2. Restoring a file system backup on a MySQL server

If your MySQL instance has been corrupted, and you previously performed a file system backup that includes the data directory and the configuration files, you can restore the instance from this backup.

Prerequisites

- You performed a file system backup on a MySQL server .
- The target server must meet the following conditions of the backup source:
  - The MySQL version must be identical or higher.
  - The system architecture must be identical.

Procedure

1. Stop the **mysqld** service:

```
# systemctl stop mysqld.service
```

2. Remove the current **/var/lib/mysql/** directory:

```
# rm -rf /var/lib/mysql/
```

3. Restore the data directory from your backup:

```
# cp -rp /root/mysqld-backup/data/mysql/ /var/lib/
```

4. Ensure the correct ownership of the **/var/lib/mysql/** directory:

```
# chown -R mysql:mysql /var/lib/mysql/
```

5. Restore the SELinux context of the **/var/lib/mysql/** directory:

```
# restorecon -Rv /var/lib/mysql/
```

6. Remove the current configuration files:

```
# rm -rf /etc/my.cnf /etc/my.cnf.d/
```

7. Restore the configuration files from your backup:

```
# cp -rp /root/mysqld-backup/config/my.cnf /root/mysqld-backup/config/my.cnf.d/ /etc/
```

8. Ensure the correct ownership of the configuration files:

```
# chown -R root:root /etc/my.cnf /etc/my.cnf.d/
```

9. Restore the SELinux context of the configuration files:

```
# restorecon -Rv /etc/my.cnf /etc/my.cnf.d/
```

10. Start the **mysqld** service:

```
# systemctl start mysqld.service
```

Verification

- Connect to a MySQL database and query data, for example:

```
# mysql -u root -p <database> -e "*SELECT * FROM <table>;"
```

2.8. REPLICATING MYSQL WITH TLS ENCRYPTION

You can set up MySQL replication with TLS encryption to create secure data replication between source and replica servers.

WARNING

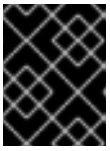
Replication itself is not a sufficient backup solution. Replication protects source servers against hardware failures, but it does not ensure protection against data loss.

2.8.1. Configuring a MySQL source server

You can set configuration options required for a MySQL source server to properly run and replicate all changes made on the database server through the TLS protocol.

Prerequisites

- The source server is installed.
- The source server has TLS set up.



IMPORTANT

The source and replica certificates must be signed by the same certificate authority.

Procedure

1. Include the following options in the **/etc/my.cnf.d/mysql-server.cnf** file under the **[mysqld]** section:

   - **bind-address=*source_ip_adress***
     This option is required for connections made from replicas to the source.
   - **server-id=*id***
     The *id* must be unique.
   - **log_bin=*path_to_source_server_log***
     This option defines a path to the binary log file of the MySQL source server. For example: **log_bin=/var/log/mysql/mysql-bin.log**.
   - **gtid_mode=ON**
     This option enables global transaction identifiers (GTIDs) on the server.
   - **enforce-gtid-consistency=ON**
     The server enforces GTID consistency by allowing execution of only statements that can be safely logged by using a GTID.
   - *Optional:* **binlog_do_db=*db_name***
     Use this option if you want to replicate only selected databases. To replicate more than one selected database, specify each of the databases separately:

     ```
     binlog_do_db=db_name1
     binlog_do_db=db_name2
     binlog_do_db=db_name3
     ```

   - *Optional:* **binlog_ignore_db=*db_name***
     Use this option to exclude a specific database from replication.

2. Restart the **mysqld** service:

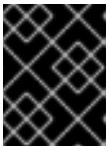
   ```
   # systemctl restart mysqld.service
   ```

2.8.2. Configuring a MySQL replica server

You can set configuration options required for a MySQL replica server to ensure a successful replication.

Prerequisites

- The replica server is installed.
- The replica server has TLS set up.



IMPORTANT

The source and replica certificates must be signed by the same certificate authority.

Procedure

1. Include the following options in the **/etc/my.cnf.d/mysql-server.cnf** file under the **[mysqld]** section:

   - **server-id=*id***
     The *id* must be unique.
   - **relay-log=*path_to_replica_server_log***
     The relay log is a set of log files created by the MySQL replica server during replication.
   - **log_bin=*path_to_replica_sever_log***
     This option defines a path to the binary log file of the MySQL replica server. For example: **log_bin=/var/log/mysql/mysql-bin.log**.

     This option is not required in a replica but strongly recommended.
   - **gtid_mode=ON**
     This option enables global transaction identifiers (GTIDs) on the server.
   - **enforce-gtid-consistency=ON**
     The server enforces GTID consistency by allowing execution of only statements that can be safely logged by using a GTID.
   - **log-replica-updates=ON**
     This option ensures that updates received from the source server are logged in the replica's binary log.
   - **skip-replica-start=ON**
     This option ensures that the replica server does not start the replication threads when the replica server starts.
   - *Optional:* **binlog_do_db=*db_name***
     Use this option if you want to replicate only certain databases. To replicate more than one database, specify each of the databases separately:

     ```
     binlog_do_db=db_name1
     binlog_do_db=db_name2
     binlog_do_db=db_name3
     ```

   - *Optional:* **binlog_ignore_db=*db_name***
     Use this option to exclude a specific database from replication.

2. Restart the **mysqld** service:

   ```
   # systemctl restart mysqld.service
   ```

2.8.3. Creating a replication user on the MySQL source server

You must create a replication user with appropriate permissions on the MySQL source server to enable replica servers to connect and receive data changes.

Prerequisites

- The source server is installed and configured as described in Configuring a MySQL source server.

Procedure

1. Create a replication user:

```
mysql> CREATE USER 'replication_user'@'replica_server_hostname' IDENTIFIED WITH mysql_native_password BY 'password';
```

2. Grant the user replication permissions:

```
mysql> GRANT REPLICATION SLAVE ON *.* TO 'replication_user'@'replica_server_hostname';*
```

3. Reload the grant tables in the MySQL database:

```
mysql> FLUSH PRIVILEGES;
```

4. Set the source server to read-only state:

```
mysql> SET @@GLOBAL.read_only = ON;
```

2.8.4. Connecting the MySQL replica server to the source server

You must configure the replica server with source server credentials and start replication to establish a connection between the MySQL replica and source servers.

Prerequisites

- The source server is installed and configured as described in Configuring a MySQL source server.
- The replica server is installed and configured as described in Configuring a MySQL replica server.
- You have created a replication user. See Creating a replication user on the MySQL source server.

Procedure

1. Set the replica server to read-only state:

```
mysql> SET @@GLOBAL.read_only = ON;
```

2. Configure the replication source:

```
mysql> CHANGE REPLICATION SOURCE TO
```

```
SOURCE_HOST='source_hostname',
SOURCE_USER='replication_user',
SOURCE_PASSWORD='password',
SOURCE_AUTO_POSITION=1,
SOURCE_SSL=1,
SOURCE_SSL_CA='path_to_ca_on_source',
SOURCE_SSL_CAPATH='path_to_directory_with_certificates',
SOURCE_SSL_CERT='path_to_source_certificate',
SOURCE_SSL_KEY='path_to_source_key';
```

3. Start the replica thread in the MySQL replica server:

```
mysql> START REPLICA;
```

4. Unset the read-only state on both the source and replica servers:

```
mysql> SET @@GLOBAL.read_only = OFF;
```

5. Optional: Inspect the status of the replica server for debugging purposes:

```
mysql> SHOW REPLICA STATUS\G;
```

NOTE

If the replica server fails to start or connect, you can skip a certain number of events following the binary log file position displayed in the output of the **SHOW MASTER STATUS** command. For example, skip the first event from the defined position:

```
mysql> SET GLOBAL SQL_SLAVE_SKIP_COUNTER=1;
```

Afterwards, try to start the replica server again.

6. Optional: Stop the replica thread in the replica server:

```
mysql> STOP REPLICA;
```

2.8.5. Verifying replication on a MySQL server

You can verify that MySQL replication works correctly by creating test databases and checking replication status on source and replica servers.

Procedure

1. Create an example database on the source server:

```
mysql> CREATE DATABASE test_db_name;
```

2. Verify that the ***test_db_name*** database replicates on the replica server.

3. Display status information about the binary log files of the MySQL server by executing the following command on either of the source or replica servers:

```
mysql> SHOW MASTER STATUS;
```

The **Executed_Gtid_Set** column, which shows a set of GTIDs for transactions executed on the source, must not be empty.

NOTE

The same set of GTIDs is displayed in the **Executed_Gtid_Set** row when you use the **SHOW REPLICA STATUS** on the replica server.

Additional resources

- MySQL Replication documentation
- Replication with Global Transaction Identifiers

2.9. MIGRATING A MYSQL INSTANCE FROM A PREVIOUS RHEL VERSION TO MYSQL 8.4 ON RHEL 10

RHEL 10 provides MySQL 8.4. If you run a MySQL instance on a previous RHEL version, you can set up RHEL 10 on a new host and migrate the instance to it.

Prerequisites

- You set up RHEL 10 on a new host.
- You performed a file system backup of the MySQL instance on the RHEL 8 or RHEL 9 host.

Procedure

1. Install the **mysql8.4-server** package:

```
# dnf install mysql8.4-server
```

2. Stop the service if it is already running:

```
# systemctl stop mysqld.service
```

3. Copy the content of the **/var/lib/mysql/** directory from the previous host to the same location on the RHEL 10 host.
4. Copy the configuration files from the previous host to the **/etc/my.cnf.d/** directory, and ensure that the files includes only options valid for MySQL 8.4. For details, see the upstream documentation.
5. Restore the SELinux context:

```
# restorecon -rv /var/lib/mysql/
# restorecon -rv /etc/my.cnf.d/
```

6. Ensure the correct ownership of **/var/lib/mysql/** and its subdirectories:

```
# chown -R mysql:mysql /var/lib/mysql/
```

7. Enable and start the **mysqld** service:

```
# systemctl enable --now mysqld.service
```

When the service starts, MySQL automatically checks, repairs, and updates internal tables.

Verification

- Establish a connection to the MySQL server:

```
# mysql -u root -p -h <hostname>
```

CHAPTER 3. USING POSTGRESQL

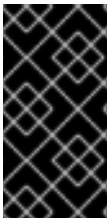
The PostgreSQL server is an open source robust and highly-extensible database server based on the SQL language. The PostgreSQL server provides an object-relational database system that can manage extensive datasets and a high number of concurrent users.

The PostgreSQL server includes features for ensuring data integrity, building fault-tolerant environments and applications. With the PostgreSQL server, you can extend a database with your own data types, custom functions, or code from different programming languages without the need to recompile the database.

Learn how to install and configure PostgreSQL on a RHEL system, how to back up PostgreSQL data, and how to migrate from an earlier PostgreSQL version.

3.1. INSTALLING POSTGRESQL

RHEL 10 provides PostgreSQL 16 as the initial version of the Application Stream, which can install as an RPM package. Additional PostgreSQL versions are provided as alternative versions with a shorter life cycle in minor releases of RHEL 10.



IMPORTANT

By design, you can install only one version (stream) of the same module and, because of conflicting RPM packages, you cannot install multiple PostgreSQL instances on the same host. As an alternative, you can run the database server services in a container. See Using containers to run multiple PostgreSQL instances on a single host.

Procedure

1. Install the PostgreSQL server packages:

```
# dnf install postgresql-server
```

   The **postgres** superuser is created automatically.

2. Initialize the database cluster:

```
# postgresql-setup --initdb
```

   Store the data in the default **/var/lib/pgsql/data** directory.

3. Enable and start the **postgresql** service:

```
# systemctl enable --now postgresql.service
```

3.2. USING CONTAINERS TO RUN MULTIPLE POSTGRESQL INSTANCES ON A SINGLE HOST

If you install PostgreSQL from packages, you can run only a single version of it on the same host. Tu run multiple instances or different versions of PostgreSQL, you can run the service in a container.

Prerequisites

- The **podman** package is installed.

Procedure

1. Authenticate to the **registry.redhat.io** registry by using your Red Hat Customer Portal account:

```
# podman login registry.redhat.io
```

   Skip this step if you are already logged in to the container registry.

2. Start the containers you want to use. For each container, enter:

```
$ podman run -d --name <container_name> -e POSTGRESQL_USER=<user_name> -e
POSTGRESQL_PASSWORD=<password> -p <host_port_1>:5432 rhel10/postgresql-16
```

   For more information about the usage of this container image, see the Red Hat Ecosystem Catalog.

IMPORTANT

The container names and host ports of the two database servers must differ.

3. To ensure that clients can access the database server on the network, open the host ports in the firewall:

```
# firewall-cmd --permanent --add-port={<host_port_1>/tcp,<host_port_2>/tcp,...}
# firewall-cmd --reload
```

Verification

1. Connect to the database server and log in as root:

```
# psql -u postgres -p -h localhost -P <host_port> --protocol tcp
```

2. Display information about running containers:

```
$ podman ps
```

Additional resources

- Building, running, and managing containers
- Browse containers in the Red Hat Ecosystem Catalog

3.3. CREATING POSTGRESQL USERS

You can create PostgreSQL users with specific permissions to manage database access and control user privileges for secure database administration.

PostgreSQL users are of the following types:

- The **postgres** Linux system user: Use it only to run the PostgreSQL server and client applications, such as **pg_dump**. Do not use the **postgres** system user for any interactive work on PostgreSQL administration, such as database creation and user management.
- A database superuser: The default **postgres** PostgreSQL superuser is not related to the **postgres** system user. You can limit access of the **postgres** superuser in the **/var/lib/pgsql/data/pg_hba.conf** file, otherwise no other permission limitations exist. You can also create other database superusers.
- A role with specific database access permissions:
  - A database user: Has a permission to log in by default.
  - A group of users: Enables managing permissions for the group as a whole.

Roles can own database objects (for example, tables and functions) and can assign object privileges to other roles by using SQL commands.

Standard database management privileges include **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **TRUNCATE**, **REFERENCES**, **TRIGGER**, **CREATE**, **CONNECT**, **TEMPORARY**, **EXECUTE**, and **USAGE**.

Role attributes are special privileges, such as **LOGIN**, **SUPERUSER**, **CREATEDB**, and **CREATEROLE**.

IMPORTANT

Perform most tasks as a role that is not a superuser. A common practice is to create a role that has the **CREATEDB** and **CREATEROLE** privileges and use this role for all routine management of databases and roles.

Prerequisites

- The PostgreSQL server is installed.
- The database cluster is initialized.
- The **password_encryption** parameter in the **/var/lib/pgsql/data/postgresql.conf** file is set to **scram-sha-256**.
- Entries in the **/var/lib/pgsql/data/pg_hba.conf** file use the **scram-sha-256** hashing algorithm as authentication method.

Procedure

1. Log in as the **postgres** system user, or switch to this user:

```
# su - postgres
```

2. Start the PostgreSQL interactive terminal:

```
$ psql
psql (16.4)
Type "help" for help.

postgres=#
```

3. Optional: Obtain information about the current database connection:

```
postgres=# \conninfo
You are connected to database "postgres" as user "postgres" via socket in
"/var/run/postgresql" at port "5432".
```

4. Create a user named **mydbuser**, set a password for it, and assign the **CREATEROLE** and **CREATEDB** permissions to the user:

```
postgres=# CREATE USER mydbuser WITH PASSWORD '<password>' CREATEROLE
CREATEDB;
CREATE ROLE
```

The **mydbuser** user now can perform routine database management operations: create databases and manage user indexes.

5. Log out of the interactive terminal by using the **\q** meta command:

```
postgres=# \q
```

Verification

1. Log in to the PostgreSQL terminal as **mydbuser**, specify the hostname, and connect to the default **postgres** database, which was created during initialization:

```
# psql -U mydbuser -h 127.0.0.1 -d postgres
Password for user mydbuser:
Type the password.
psql (16.4)
Type "help" for help.

postgres=>
```

2. Create a database:

```
postgres=> CREATE DATABASE <db_name>;
```

3. Log out of the session:

```
postgres=# \q
```

4. Connect to new database as **mydbuser**:

```
# psql -U mydbuser -h 127.0.0.1 -d <db_name>
Password for user mydbuser:
psql (16.4)
Type "help" for help.
mydatabase=>
```

Additional resources

- PostgreSQL database roles

- PostgreSQL privileges

3.4. CONFIGURING POSTGRESQL

You can configure PostgreSQL by editing the configuration files in the database cluster directory to set database parameters, authentication, and client access. By default, PostgreSQL uses the **/var/lib/pgsql/data/** directory.

- **/var/lib/pgsql/data/postgresql.conf** - is used for setting the database cluster parameters.
- **/var/lib/pgsql/data/postgresql.auto.conf** - holds basic PostgreSQL settings similarly to **postgresql.conf**. However, this file is under the server's control. It is edited by the **ALTER SYSTEM** queries, and cannot be edited manually.
- **/var/lib/pgsql/data/pg_ident.conf** - is used for mapping user identities from external authentication mechanisms into the PostgreSQL user identities.
- **/var/lib/pgsql/data/pg_hba.conf** - is used for configuring client authentication for PostgreSQL databases.

Procedure

1. Edit the **/var/lib/pgsql/data/postgresql.conf** file and configure basic settings of the database cluster parameters, for example:

```
log_connections = yes
log_destination = 'syslog'
search_path = '"$user", public'
shared_buffers = 128MB
password_encryption = scram-sha-256
```

2. Edit the **/var/lib/pgsql/data/pg_hba.conf** file and configure client authentication, for example:

```
# TYPE    DATABASE      USER        ADDRESS             METHOD
local     all           all                             trust
host      postgres      all         192.168.93.0/24     ident
host      all           all         .example.com        scram-sha-256
```

3. Restart the **postgresql** service so that the changes become effective:

```
# systemctl restart postgresql.service
```

3.5. CONFIGURING TLS ENCRYPTION ON A POSTGRESQL SERVER

By default, PostgreSQL uses unencrypted connections. For more secure connections, you can enable Transport Layer Security (TLS) support on the PostgreSQL server and configure your clients to establish encrypted connections.

Prerequisites

- You created a TLS private key and a certificate authority (CA) issued a server certificate for your PostgreSQL server.

- The PostgreSQL server is installed.
- The database cluster is initialized.
- If FIPS mode is enabled, clients must either support the Extended Master Secret (EMS) extension or use TLS 1.3. TLS 1.2 connections without EMS fail. For more information, see the Red Hat Knowledgebase solution TLS extension "Extended Master Secret" enforced on RHEL 9.2 and later.

Procedure

1. Store the private key and the server certificate in the **/var/lib/pgsql/data/** directory:

```
# cp server.{key,crt} /var/lib/pgsql/data/
```

2. Set the ownership of the private key and certificate:

```
# chown postgres:postgres /var/lib/pgsql/data/server.{key,crt}
```

3. Set permissions on the server certificate that enable only the PostgreSQL server to read the file:

```
# chmod 0400 /var/lib/pgsql/data/server.key
```

   Because certificates are part of the communication before a secure connection is established, any client can retrieve them without authentication. Therefore, you do not need to set strict permissions on the server certificate file.

4. Edit the **/var/lib/pgsql/data/postgresql.conf** file and make the following changes:

   a. Set the **scram-sha-256** hashing algorithm:

```
password_encryption = scram-sha-256
```

   b. Enable TLS encryption:

```
ssl = on
```

5. Edit the **/var/lib/pgsql/data/pg_hba.conf** file and update the authentication entries to use TLS encryption and the **scram-sha-256** hashing algorithm. For example, change **host** entries to **hostssl** to enable TLS encryption, and set the **scram-sha-256** hashing algorithm in the last column:

```
hostssl    all    all    192.0.2.0/24    scram-sha-256
```

6. Restart the **postgresql** service:

```
# systemctl restart postgresql.service
```

Verification

- Use the **postgres** super user to connect to a PostgreSQL server and execute the **\conninfo** meta command:

```
# psql "postgresql://postgres@localhost:5432" -c '\conninfo'
Password for user postgres:
You are connected to database "postgres" as user "postgres" on host "192.0.2.1" at port
"5432".
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression:
off)
```

3.6. BACKING UP AND RESTORING POSTGRESQL DATA WITH LOGICAL DUMPS

A logical backup of PostgreSQL data consists of the SQL statements necessary to restore the data. The advantage of logical backup over physical backup is that the data can be restored on other hardware configurations and PostgreSQL versions.

The SQL dump method is based on generating a dump file with SQL commands. When a dump is uploaded back to the database server, it recreates the database in the same state as it was at the time of the dump.

The SQL dump is ensured by the following PostgreSQL client applications:

- **pg_dump** dumps a single database without cluster-wide information about roles or tablespaces
- **pg_dumpall** dumps each database in a given cluster and preserves cluster-wide data, such as role and tablespace definitions.

By default, the **pg_dump** and **pg_dumpall** commands write their results into the standard output. To store the dump in a file, redirect the output to an SQL file. The resulting SQL file can be either in a text format or in other formats that allow for parallelism and for more detailed control of object restoration.

You can perform the SQL dump from any remote host that has access to the database.

3.6.1. Advantages and disadvantages of an SQL dump

SQL dumps are text files containing a database's structure and data in the form of SQL statements.

Advantages:

- An SQL dump is the only PostgreSQL backup method that is not server version-specific. The output of the **pg_dump** utility can be reloaded into later versions of PostgreSQL, which is not possible for file system level backups or continuous archiving.
- An SQL dump is the only method that works when transferring a database to a different machine architecture, such as going from a 32-bit to a 64-bit server.
- An SQL dump provides internally consistent dumps. A dump represents a snapshot of the database at the time **pg_dump** began running.
- The **pg_dump** utility does not block other operations on the database when it is running.

Disadvantage:

- An SQL dump takes more time compared to a file system level backup.

3.6.2. Backing up a single PostgreSQL database by using `pg_dump`

You can create a backup of a single PostgreSQL database by using the **pg_dump** utility to export the database structure and data to a file.

Prerequisites

- You are logged in as the **postgres** superuser or a user with database administrator privileges.

Procedure

- Dump a database without cluster-wide information:

```
$ pg_dump <db_name> > <dump_file>
```

  To specify which database server **pg_dump** will contact, use the following command-line options:

  - The **-h** option to define the host.
    The default host is either the local host or what is specified by the **PGHOST** environment variable.
  - The **-p** option to define the port.
    The default port is indicated by the **PGPORT** environment variable or the compiled-in default.

3.6.3. Restoring a single PostgreSQL database by using pg_dump

You can restore a PostgreSQL database from an SQL dump file by using the **pg_restore** utility to re-create the database structure and data.

Prerequisites

- You are logged in as the **postgres** superuser or a user with database administrator privileges.

Procedure

1. Create a new database:

```
$ createdb <db_name>
```

2. Verify that all users who own objects or were granted permissions on objects in the dumped database already exist. If such users do not exist, the restore fails to re-create the objects with the original ownership and permissions.

3. Run the **psql** utility to restore a text file dump created by the **pg_dump** utility:

```
$ psql <db_name> < <dump_file>
```

   where **<dump_file>** is the output of the **pg_dump** command. To restore a non-text file dump, use the **pg_restore** utility instead:

```
$ pg_restore <non-plain_text_file>
```

3.6.4. Backing up all databases on a PostgreSQL server by using pg_dumpall

You can create a backup of all databases on a PostgreSQL server by using the **pg_dumpall** utility to export all databases and cluster-wide data to a single file.

Prerequisites

- You are logged in as the **postgres** superuser or a user with database administrator privileges.

Procedure

- Dump all databases in the database cluster and preserve cluster-wide data:

```
$ pg_dumpall > <dump_file>
```

  To specify which database server **pg_dumpall** will contact, use the following command-line options:

  - The **-h** option to define the host.
    The default host is either the local host or what is specified by the **PGHOST** environment variable.
  - The **-p** option to define the port.
    The default port is indicated by the **PGPORT** environment variable or the compiled-in default.
  - The **-l** option to define the default database.
    This option enables you to choose a default database different from the **postgres** database created automatically during initialization.

3.6.5. Restoring all databases on a PostgreSQL server by using pg_dumpall

You can restore all databases on a PostgreSQL server from a **pg_dumpall** file by using the **psql** utility to re-create the entire database cluster.

Prerequisites

- You are logged in as the **postgres** superuser or a user with database administrator privileges.

Procedure

1. Ensure that all users who own objects or were granted permissions on objects in the dumped databases already exist. If such users do not exist, the restore fails to re-create the objects with the original ownership and permissions.
2. Run the **psql** utility to restore a text file dump created by the **pg_dumpall** utility:

```
$ psql < <dump_file>
```

   where **<dump_file>** is the output of the **pg_dumpall** command.

3.6.6. Handling SQL errors during restore

By default, the **psql** utility continues to execute if an SQL error occurs, causing the database to restore only partially. Alternatively, you can configure **psql** to stop on errors to ensure data integrity.

Prerequisites

- You are logged in as the **postgres** superuser or a user with database administrator privileges.

Procedure

- Make **psql** exit with an exit status of 3 if an SQL error occurs by setting the **ON_ERROR_STOP** variable:

```
$ psql --set ON_ERROR_STOP=on <db_name> < <dump_file>
```

- Specify that the whole dump is restored as a single transaction so that the restore is either fully completed or canceled.
  - When restoring a text file dump by using the **psql** utility:

```
$ psql -1
```

  - When restoring a non-text file dump by using the **pg_restore** utility:

```
$ pg_restore -e
```

  Note that when you use this approach, even a minor error can cancel a restore operation that has already run for many hours.

Additional resources

- PostgreSQL Documentation - SQL dump

3.7. BACKING UP AND RESTORING POSTGRESQL DATA WITH PHYSICAL COPIES

A physical backup of PostgreSQL data contains files and directories that store the content. This method is typically faster and smaller in size.

3.7.1. Performing a file system backup on a PostgreSQL server

A file system-level backup is a fast way to back up a complete PostgreSQL instance. This method requires a shutdown of the **postgresql** service for data consistency.

IMPORTANT

A PostgreSQL file system-level backup is specific to an architecture and RHEL major version. You cannot restore data backed up by this method on a different architecture or RHEL major version.

Procedure

1. Stop the **postgresql** service:

```
# systemctl stop postgresql.service
```

2. Create a backup directory, for example:

```
# mkdir -p /root/postgresql-backup/
```

3. Back up the **/var/lib/pgsql/** directory:

```
# cp -rp /var/lib/pgsql/ /root/postgresql-backup/
```

The **/var/lib/pgsql/** contains all essential files of the PostgreSQL database server, including configuration files, data files, and logs.

4. Start the **postgresql** service:

```
# systemctl start postgresql.service
```

3.7.2. Restoring a file system backup on a PostgreSQL server

If your PostgreSQL instance has been corrupted, and you previously performed a file system backup that includes the data directory, you can restore the instance from this backup.

Prerequisites

- You performed a file system backup on a PostgreSQL server .
- The target server must meet the following conditions of the backup source:
  - The PostgreSQL version must be identical.
  - The system architecture must be identical.

Procedure

1. Stop the **postgresql** service:

```
# systemctl stop postgresql.service
```

2. Remove the current **/var/lib/pgsql/** directory:

```
# rm -rf /var/lib/pgsql/
```

3. Restore the data directory from your backup:

```
# cp -rp /root/postgresql-backup/pgsql/ /var/lib/
```

4. Ensure the correct ownership of the **/var/lib/pgsql/** directory:

```
# chown -R postgres:postgres /var/lib/pgsql/
```

5. Restore the SELinux context of the **/var/lib/pgsql/** directory:

```
# restorecon -Rv /var/lib/pgsql/
```

6. Start the **postgresql** service:

```
# systemctl start postgresql.service
```

Verification

1. Log in as the **postgres** user.
2. Connect to a database:

```
$ psql <database>
```

3. Access data in the database:

```
<database>=# SELECT * FROM <table>;
```

4. Disconnect from the PostgreSQL service:

```
<database>=# \q
```

3.8. BACKING UP AND RESTORING POSTGRESQL DATA WITH CONTINUOUS ARCHIVING

You can use continuous archiving to create robust PostgreSQL backups by combining WAL files with base backups for point-in-time recovery and high availability.

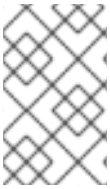
PostgreSQL records every change to the database's data files to a write-ahead log (WAL) file that is available in the **pg_wal/** subdirectory of the cluster's data directory. This log is intended primarily for a crash recovery. After a crash, you can use log entries made since the last checkpoint to restore the database to a consistent state.

The continuous archiving method, also known as an online backup, combines the WAL files with a copy of the database cluster in the form of a base backup performed on a running server or a file system level backup.

If a database recovery is needed, you can restore the database from the copy of the database cluster and then replay the log from the backed up WAL files to bring the system to the current state.

With the continuous archiving method, you must keep a continuous sequence of all archived WAL files that extends at minimum back to the start time of your last base backup. Therefore the ideal frequency of base backups depends on:

- The storage volume available for archived WAL files.
- The maximum possible duration of data recovery in situations when recovery is necessary. In cases with a long period since the last backup, the system replays more WAL segments, and the recovery therefore takes more time.



NOTE

You cannot use **pg_dump** and **pg_dumpall** SQL dumps as a part of a continuous archiving backup solution. SQL dumps produce logical backups and do not contain enough information to be used by a WAL replay.

3.8.1. Advantages and disadvantages of continuous archiving

Continuous archiving is a feature that provides a robust strategy for data backup, high availability, and point-in-time recovery (PITR) by continuously saving the database's transaction log files.

Advantages:

- With the continuous backup method, it is possible to use a base backup that is not entirely consistent because any internal inconsistency in the backup is corrected by the log replay. Therefore you can perform a base backup on a running PostgreSQL server.
- A file system snapshot is not needed; **tar** or a similar archiving utility is sufficient.
- Continuous backup can be achieved by continuing to archive the WAL files because the sequence of WAL files for the log replay can be indefinitely long. This is particularly valuable for large databases.
- Continuous backup supports point-in-time recovery. It is not necessary to replay the WAL entries to the end. The replay can be stopped at any point and the database can be restored to its state at any time since the base backup was taken.
- If the series of WAL files are continuously available to another machine that has been loaded with the same base backup file, it is possible to restore the other machine with a nearly-current copy of the database at any point.

Disadvantages:

- Continuous backup method supports only restoration of an entire database cluster, not a subset.
- Continuous backup requires extensive archival storage.

3.8.2. Setting up WAL archiving

You can enable write ahead log (WAL) archiving on your PostgreSQL server to capture and save WAL segment files for backup and point-in-time recovery purposes.

A running **PostgreSQL** server produces a sequence of WAL records. The server physically divides this sequence into WAL segment files, which are given numeric names that reflect their position in the WAL sequence. Without WAL archiving, the segment files are reused and renamed to higher segment numbers.

When archiving WAL data, the contents of each segment file are captured and saved at a new location before the segment file is reused. You have multiple options where to save the content, such as an NFS-mounted directory on another machine, a tape drive, or a CD.

Note that WAL records do not include changes to configuration files.

Procedure

1. In the **/var/lib/pgsql/data/postgresql.conf** file:

   a. Set the **wal_level** configuration parameter to **replica** or higher.

   b. Set the **archive_mode** parameter to **on**.

   c. Specify the shell command in the **archive_command** configuration parameter. You can use the **cp** command, another command, or a shell script.
   For example:

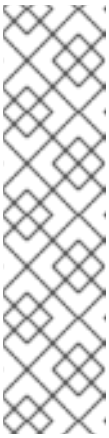
   ```
   archive_command = 'test ! -f /mnt/server/archivedir/%f && cp %p
   /mnt/server/archivedir/%f'
   ```

   where the **%p** parameter is replaced by the relative path to the file to archive and the **%f** parameter is replaced by the file name.

   This command copies archivable WAL segments to the **/mnt/server/archivedir/** directory. After replacing the **%p** and **%f** parameters, the executed command looks as follows:

   ```
   test ! -f /mnt/server/archivedir/000000010000000A900000065 && cp
   pg_wal/000000010000000A900000065
   /mnt/server/archivedir/000000010000000A900000065
   ```

   A similar command is generated for each new file that is archived.

   NOTE

   The archive command is executed only on completed WAL segments. A server that generates little WAL traffic can have a substantial delay between the completion of a transaction and its safe recording in archive storage. To limit how old unarchived data can be, you can:

   - Set the **archive_timeout** parameter to force the server to switch to a new WAL segment file with a given frequency.
   - Use the **pg_switch_wal** parameter to force a segment switch to ensure that a transaction is archived immediately after it finishes.

2. Restart the **postgresql** service to enable the changes:

   ```
   # systemctl restart postgresql.service
   ```

3. Test your archive command and ensure it does not overwrite an existing file and that it returns a nonzero exit status if it fails.

4. To protect your data, ensure that the segment files are archived into a directory that does not have group or world read access.

Additional resources

- PostgreSQL 16 Documentation

3.8.3. Making a base backup

You can create a PostgreSQL base backup by using the **pg_basebackup** utility to capture a consistent snapshot of your database for backup and recovery purposes.

The base backup process creates a backup history file that is stored into the WAL archive area and is named after the first WAL segment file that you need for the base backup.

The backup history file is a small text file containing the starting and ending times, and WAL segments of the backup. If you used the label string to identify the associated dump file, you can use the backup history file to determine which dump file to restore.

NOTE

Consider keeping several backup sets to be certain that you can recover your data.

Prerequisites

- You are logged in as the **postgres** superuser, a user with database administrator privileges, or another user with at least **REPLICATION** permissions.
- You must keep all the WAL segment files generated during and after the base backup.

Procedure

1. Use the **pg_basebackup** utility to perform the base backup.
   - To create a base backup as individual files (plain format):

     ```
     $ pg_basebackup -D <backup_directory> -Fp
     ```

     Replace *backup_directory* with your chosen backup location.

     If you use tablespaces and perform the base backup on the same host as the server, you must also use the **--tablespace-mapping** option, otherwise the backup will fail upon an attempt to write the backup to the same location.
   - To create a base backup as a **tar** archive (**tar** and compressed format):

     ```
     $ pg_basebackup -D <backup_directory> -Ft -z
     ```

     Replace *backup_directory* with your chosen backup location.

     To restore such data, you must manually extract the files in the correct locations.

   To specify which database server **pg_basebackup** will contact, use the following command-line options:

   - The **-h** option to define the host.
     The default host is either the local host or a host specified by the **PGHOST** environment variable.
   - The **-p** option to define the port.
     The default port is indicated by the **PGPORT** environment variable or the compiled-in default.

2. After the base backup process is complete, safely archive the copy of the database cluster and the WAL segment files used during the backup, which are specified in the backup history file.

3. Delete WAL segments numerically lower than the WAL segment files used in the base backup because these are older than the base backup and no longer needed for a restore.

Additional resources

- PostgreSQL Documentation - base backup
- PostgreSQL Documentation - pg_basebackup utility

3.8.4. Restoring the database by using a continuous archive backup

You can restore a PostgreSQL database by restoring the base backup and applying archived WAL files for point-in-time recovery.

Procedure

1. Stop the server:

```
# systemctl stop postgresql.service
```

2. Copy the necessary data to a temporary location.
Preferably, copy the whole cluster data directory and any tablespaces. Note that this requires enough free space on your system to hold two copies of your existing database.

   If you do not have enough space, save the contents of the cluster's **pg_wal** directory, which can contain logs that were not archived before the system went down.

3. Remove all existing files and subdirectories under the cluster data directory and under the root directories of any tablespaces you are using.

4. Restore the database files from your base backup.
Ensure that:

   - The files are restored with the correct ownership (the database system user, not **root**).
   - The files are restored with the correct permissions.
   - The symbolic links in the **pg_tblspc**/ subdirectory are restored correctly.

5. Remove any files present in the **pg_wal**/ subdirectory.
These files resulted from the base backup and are therefore obsolete. If you did not archive **pg_wal**/, re-create it with proper permissions.

6. Copy any unarchived WAL segment files that you saved in step 2 into **pg_wal**/.

7. Create the **recovery.conf** recovery command file in the cluster data directory and specify the shell command in the **restore_command** configuration parameter. You can use the **cp** command, another command, or a shell script. For example:

```
restore_command = 'cp /mnt/server/archivedir/%f "%p"'
```

8. Start the server:

```
# systemctl start postgresql.service
```

The server enters the recovery mode and proceeds to read through the archived WAL files that it needs.

If the recovery is terminated due to an external error, the server can be restarted, and it will continue the recovery. When the recovery process is completed, the server renames **recovery.conf** to **recovery.done**. This prevents the server from accidentally re-entering the recovery mode after it starts normal database operations.

9. Check the contents of the database to verify that the database has recovered into the required state.
If the database has not recovered into the required state, return to step 1. If the database has recovered into the required state, allow the users to connect by restoring the client authentication configuration in the **pg_hba.conf** file.

Additional resources

- Continuous archiving method

3.9. DIRECTLY TRANSFERRING A POSTGRESQL DATABASE FROM ONE SERVER TO ANOTHER

You can use the **pg_dump** and **psql** utilities to back up a PostgreSQL database and directly restore it on another PostgreSQL server. With this method you can transfer a database in a single step without intermediate files.

Prerequisites

- You are logged in as the **postgres** user.

Procedure

- Transfer a database from the source server to a destination server:

```
$ pg_dump -h <source_server> <db_name> | psql -h <destination_server> <db_name>
```

3.10. MIGRATING A POSTGRESQL INSTANCE FROM A PREVIOUS RHEL VERSION TO POSTGRESQL ON RHEL 10

If you already run PostgreSQL on RHEL 9 and want to move the database software to a host that runs RHEL 10, you can migrate the databases.

The following migration methods are available:

- Backup and restore upgrade - This method might require more time but works in most scenarios.
- Fast upgrade by using the **pg_upgrade** utility - This method is faster but works only if you migrate from PostgreSQL 13 to 16 and the hardware architecture stays the same.

IMPORTANT

Always back up the **/var/lib/pgsql/data/** directory on the source host before a PostgreSQL migration.

3.10.1. Migrating to PostgreSQL on RHEL 10 by using the backup and restore method

You can use the backup and restore method to migrate data from any RHEL 8 or RHEL 9 version of PostgreSQL to any equal or later version of PostgreSQL on RHEL 10.

Prerequisites

- The existing database server runs on RHEL 8 or RHEL 9 and uses a PostgreSQL version installed from the RHEL repositories.
- The locale settings on both hosts are the same. To verify this, compare the output of the **echo $LANG** command on both hosts.

Procedure

1. On the host with the existing PostgreSQL instance that you want to migrate:
   a. Export all databases to the **/var/lib/pgsql/pgdump_file.sql** file:

```
# su - postgres -c "pg_dumpall > /var/lib/pgsql/pgdump_file.sql"
```

   b. Check the exported file:

```
# su - postgres -c 'less "/var/lib/pgsql/pgdump_file.sql"'
```

   c. Copy the database dump that you created in an earlier step and the PostgreSQL configuration files to the RHEL 10 host, for example:

```
# scp /var/lib/pgsql/pgdump_file.sql \
      /var/lib/pgsql/data/pg_hba.conf \
      /var/lib/pgsql/data/pg_ident.conf \
      /var/lib/pgsql/data/postgresql.conf \
      <user>@<rhel_10_host>:/tmp/
```

2. On the RHEL 10 host:
   a. Install the **postgresql-server** package:

```
# dnf install postgresql-server
```

   b. Initialize the **/var/lib/pgsql/data/** directory:

```
# postgresql-setup --initdb
```

   c. Move the copied configuration files to the **/var/lib/pgsql/data/** directory:

```
# mv /tmp/pg_hba.conf \
     /tmp/pg_ident.conf \
```

```
/tmp/postgresql.conf \
/var/lib/pgsql/data/
```

d. Ensure a correct ownership of the content in the **/var/lib/pgsql/data/ directory**:

```
# chown -R postgres:postgres /var/lib/pgsql/data/
```

e. Restore the SELinux context on **/var/lib/pgsql/data/**:

```
# restorecon -Rv /var/lib/pgsql/data/
```

f. Enable and start the **postgresql** service:

```
# systemctl enable --now postgresql.service
```

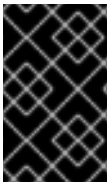
g. Import the data as the **postgres** user:

```
# su - postgres -c 'psql -f /tmp/pgdump_file.sql postgres'
```

h. Verify your databases and ensure that your applications that use the PostgreSQL server work as expected.

3.10.2. Migrating PostgreSQL 13 from a previous RHEL version to PostgreSQL 16 on RHEL 10 by using pg_update

If you want to migrate a PostgreSQL 13 instance from a previous RHEL version to PostgreSQL 16 on RHEL 10, you can use the fast upgrade method. With this method, you copy the content of the **/var/lib/pgsql/data/** directory to the RHEL 10 host and the **pg_update** utility converts the databases.



IMPORTANT

This method works only if your existing PostgreSQL instance is version 13 and the hardware architecture is the same on the source and destination host. In other cases, use the backup and restore method.

Prerequisites

- The existing database server uses PostgreSQL 13.
- The hardware architecture of the current and future server is the same.
- The RHEL 10 host has enough free space on the disk that holds the **/var/lib/pgsql/** directory. For example, if the size of the directory on the PostgreSQL server want to migrate is 10 GiB, you require at least 20 GiB free disk space on the RHEL 10 host during the migration.
- The locale settings on both hosts are the same. To verify this, compare the output of the **echo $LANG** command on both hosts.

Procedure

1. On the host with the existing PostgreSQL instance that you want to migrate:
   a. Stop the **postgresql** service:

```
# systemctl stop postgresql.service
```

b. Change into the **/var/lib/pgsql/** directory, and back up the **data** subdirectory:

```
# cd /var/lib/pgsql/
# tar -zcf ~/pgdata.bak.tar.gz data/
```

c. Copy the **~/pgdata.bak.tar.gz** archive to the RHEL 10 host, for example:

```
# scp ~/pgdata.bak.tar.gz <user>@<rhel_10_host>:/tmp/
```

2. On the RHEL 10 host:

a. Install the required packages:

```
# dnf install postgresql-server postgresql-upgrade
```

The **postgresql-upgrade** package provides a PostgreSQL 13 server which is required during the migration.

b. If you use third party PostgreSQL server modules, build them against both the **postgresql-devel** and **postgresql-upgrade-devel** packages, and install them.

c. Ensure that the **postgresql** service is stopped:

```
# systemctl stop postgresql.service
```

d. Change into the **/var/lib/pgsql/** directory, and extract the backed up data directory from the previous host:

```
# cd /var/lib/pgsql/
# tar -zxf /tmp/pgdata.bak.tar.gz
```

e. Optional: Remove the **/tmp/pgdata.bak.tar.gz** archive:

```
# rm /tmp/pgdata.bak.tar.gz
```

f. Perform the upgrade process:

```
# postgresql-setup --upgrade
```

The **postgresql-setup** shell script renames the **/var/lib/pgsql/data/** directory to **/var/lib/pgsql/data-old/** and uses the **pg_upgrade** utility to migrate the databases to a re-created **/var/lib/pgsql/data/** directory.

IMPORTANT

The **pg_upgrade** utility migrates only the databases and not the configuration files. After the migration, **/var/lib/pgsql/data/** contains only the default **.conf** files. If you, previously, had custom configuration files, copy them from the **/var/lib/pgsql/data-old/** directory and ensure that they are compatible with the new PostgreSQL version.

g. Enable and start the **postgresql** service:

```
# systemctl enable --now postgresql.service
```

h. Clean up and analyze all databases:

```
# su postgres -c 'vacuumdb --all --analyze-in-stages'
```

i. Verify your databases and ensure that your applications that use the PostgreSQL server work as expected.

j. Optional: Remove the **/var/lib/pgsql/data-old/** directory which contains the databases and configuration file from before the migration.

```
# rm -r /var/lib/pgsql/data-old/
```

k. Optional: Remove the **postgresql-upgrade** package:

```
# dnf remove postgresql-upgrade
```