Red Hat Enterprise Linux 9.0 Beta

Configuring and using database servers

A guide to configuring and using database servers in Red Hat Enterprise Linux 9

Last Updated: 2021-11-15

Red Hat Enterprise Linux 9.0 Beta Configuring and using database servers

A guide to configuring and using database servers in Red Hat Enterprise Linux 9

Legal Notice

Abstract

This document describes the basics of configuring database servers using MariaDB and PostgreSQL on Red Hat Enterprise Linux 9.

Table of Contents

RHEL BETA RELEASE

Red Hat provides Red Hat Enterprise Linux Beta access to all subscribed Red Hat accounts. The purpose of Beta access is to:

- Provide an opportunity to customers to test major features and capabilities prior to the general availability release and provide feedback or report issues.
- Provide Beta product documentation as a preview. Beta product documentation is under development and is subject to substantial change.

Note that Red Hat does not support the usage of RHEL Beta releases in production use cases. For more information, see What does Beta mean in Red Hat Enterprise Linux and can I upgrade a RHEL Beta installation to a General Availability (GA) release?.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages:
  1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.
  2. Use your mouse cursor to highlight the part of text that you want to comment on.
  3. Click the **Add Feedback** pop-up that appears below the highlighted text.
  4. Follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
  1. Go to the Bugzilla website.
  2. As the Component, use **Documentation**.
  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click **Submit Bug**.

CHAPTER 1. INTRODUCTION TO DATABASE SERVERS

A database server is a service that provides features of a database management system (DBMS). DBMS provides utilities for database administration and interacts with end users, applications, and databases.

Red Hat Enterprise Linux 9 provides the following database management systems:

- MariaDB 10.5
- MySQL 8.0
- PostgreSQL 13
- Redis 6

CHAPTER 2. USING MARIADB

2.1. GETTING STARTED WITH MARIADB

The **MariaDB** server is an open source fast and robust database server that is based on the MySQL technology.

MariaDB is a relational database that converts data into structured information and provides an SQL interface for accessing data. It includes multiple storage engines and plug-ins, as well as geographic information system (GIS) and JavaScript Object Notation (JSON) features.

For Red Hat Enterprise Linux 9 this part describes:

- How to install the **MariaDB** server in Installing MariaDB.
- How to adjust **MariaDB** configuration in Configuring MariaDB.
- How to set up TLS encryption on **MariaDB** in Setting up TLS encryption on MariaDB.
- How to enable TLS encryption in **MariaDB** clients globally in Globally enabling TLS encryption in MariaDB clients.
- How to back up **MariaDB** data in Backing up MariaDB data.
- How to migrate from a RHEL 8 version of MariaDB 10.3 to RHEL 9 version of MariaDB 10.5, in Migrating to MariaDB 10.5.
- How to replicate a database using the MariaDB Galera Cluster in Replicating MariaDB with Galera.

2.2. INSTALLING MARIADB

RHEL 9.0 provides **MariaDB 10.5** as the initial version of this Application Stream, which can be installed easily as an RPM package. Additional **MariaDB** versions will be provided as modules with a shorter life cycle in future minor releases of RHEL 9.

To install **MariaDB**, use the following procedure.

Procedure

1. Install **MariaDB** server packages:

```
# yum install mariadb-server
```

2. Start the **mariadb** service:

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

3. Enable the **mariadb** service to start at boot:

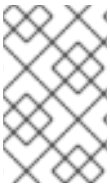
```
# systemctl enable mariadb.service
```

4. *Recommended:* To improve security when installing **MariaDB**, run the following command:

```
# mariadb-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)



NOTE

The **MariaDB** and **MySQL** database servers cannot be installed in parallel in RHEL 9 due to conflicting RPM packages. In RHEL 9, different versions of database servers can be used in containers.

2.3. CONFIGURING MARIADB

To configure the **MariaDB** server for networking, use the following procedure.

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. Restart the **mariadb** service:

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

2.4. 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.

2.4.1. 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/
```

2.4.2. Configuring TLS on a MariaDB server

To improve security, enable TLS support on the MariaDB server. As a result, clients can transmit data with the server using TLS encryption.

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.

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

2.4.3. Requiring TLS encrypted connections for specific user accounts

Users that have access to sensitive data should always use a TLS-encrypted connection to avoid sending data unencrypted over the network.

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 using a TLS-encrypted connection:

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

Verification

1. Connect to the server as the **example** user 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**).

Additional resources

- Configuring TLS on a MariaDB server

2.5. GLOBALLY ENABLING TLS ENCRYPTION IN MARIADB CLIENTS

If your MariaDB server supports TLS encryption, configure your clients to establish only secure connections and to verify the server certificate. This procedure describes how to enable TLS support for all users on the server.

2.5.1. 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.

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 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
```

Additional resources

- The **--ssl*** parameter descriptions in the **mysql(1)** man page.

2.6. BACKING UP MARIADB DATA

There are two main ways to back up data in Red Hat Enterprise Linux 9 from a MariaDB database:

- Logical backup
- Physical backup

Logical backup consists of the SQL statements necessary to restore the data. This type of backup exports information and records in plain text files.

The main advantage of logical backup over physical backup is portability and flexibility. The data can be restored on other hardware configurations, MariaDB versions or Database Management System (DBMS), which is not possible with physical backups.

Note that logical backup can be performed if the **mariadb.service** is running. Logical backup does not include log and configuration files.

Physical backup consists of copies of files and directories that store the content.

Physical backup has the following advantages compared to logical backup:

- Output is more compact.
- Backup is smaller in size.
- Backup and restore are faster.
- Backup includes log and configuration files.

Note that physical backup must be performed when the **mariadb.service** is not running or all tables in the database are locked to prevent changes during the backup.

You can use one of the following **MariaDB** backup approaches to back up data from a **MariaDB** database:

- Logical backup with mariadb-dump
- Physical online backup using the Mariabackup utility
- File system backup
- Replication as a backup solution

2.6.1. Performing logical backup with mariadb-dump

The **mariadb-dump** client is a backup utility, which can be used to dump a database or a collection of databases for the purpose of a backup or transfer to another database server. The output of **mariadb-dump** typically consists of SQL statements to re-create the server table structure, populate it with data, or both. **mariadb-dump** can also generate files in other formats, including XML and delimited text formats, such as CSV.

To perform the **mariadb-dump** backup, you can use one of the following options:

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

Procedure

- To dump an entire database, run:

```
# mariadb-dump [options] db_name > backup-file.sql
```

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

```
# mariadb-dump [options] db_name [tbl_name …] > backup-file.sql
```

- To load the dump file back into a server, use either of these commands:

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

```
# mariadb -e "source /path-to-backup/backup-file.sql" db_name
```

- To populate databases by copying data from one **MariaDB** server to another, run:

```
# mariadb-dump --opt db_name | mariadb --host=remote_host -C db_name
```

- To dump multiple databases at once, run:

```
# mariadb-dump [options] --databases db_name1 [db_name2 …] > my_databases.sql
```

- To dump all databases, run:

```
# mariadb-dump [options] --all-databases > all_databases.sql
```

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

```
$ mariadb-dump --help
```

Additional resources

- For more information on logical backup with **mariadb-dump**, see the MariaDB Documentation.

2.6.2. Performing physical online backup using the Mariabackup utility

Mariabackup is a utility based on the Percona XtraBackup technology, which enables performing physical online backups of InnoDB, Aria, and MyISAM tables. This utility is provided by the **mariadb-backup** package from the AppStream repository.

Mariabackup supports full backup capability for **MariaDB** server, which includes encrypted and compressed data.

Prerequisites

- The **mariadb-backup** package is installed on the system:

```
# yum install mariadb-backup
```

- 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.

To create a backup of a database using **Mariabackup**, use the following procedure.

Procedure

- To create a backup while providing credentials on the command line, run:

```
$ 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:

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

  2. Add the following lines into the **[xtrabackup]** or **[mysqld]** section of the new file:

```
[xtrabackup]
user=myuser
password=mypassword
```

  3. Perform the backup:

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

IMPORTANT

Mariabackup does not read options in the **[mariadb]** section of configuration files. If a non-default data directory is specified on a **MariaDB** server, you must specify this directory in the **[xtrabackup]** or **[mysqld]** sections of configuration files so that **Mariabackup** is able to find the data directory.

To specify a non-default data directory, include the following line in the **[xtrabackup]** or **[mysqld]** sections of **MariaDB** configuration files:

```
datadir=/var/mycustomdatadir
```

Additional resources

- For more information on performing backups with **Mariabackup**, see Full Backup and Restore with Mariabackup.

2.6.3. Restoring data using the Mariabackup utility

When the backup is complete, you can restore the data from the backup by using the **mariabackup** command with one of the following options:

- **--copy-back** allows you to keep the original backup files.
- **--move-back** moves the backup files to the data directory and removes the original backup files.

To restore data using the **Mariabackup** utility, use the following procedure.

Prerequisites

- Make sure that the **mariadb** service is not running:

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

- Make sure that the data directory is empty.
- Users of **Mariabackup** must have the **RELOAD**, **LOCK TABLES**, and **REPLICATION CLIENT** privileges.

Procedure

1. Run the **mariabackup** command:
   - To restore data and keep the original backup files, use the **--copy-back** option:

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

   - To restore data and remove the original backup files, use the **--move-back** option:

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

2. Fix the file permissions.
   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. After restoring a backup, you may need to adjust the owner of the data directory to match the user and group for the **MariaDB** server, typically **mysql** for both.

   For example, to recursively change ownership of the files to the **mysql** user and group:

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

3. Start the **mariadb** service:

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

Additional resources

- For more information see Full Backup and Restore with Mariabackup .

2.6.4. Performing file system backup

To create a file system backup of **MariaDB** data files, switch to the **root** user and copy the content of the **MariaDB** data directory to your backup location.

To back up also your current configuration or the log files, use the optional steps of the following procedure.

Procedure

1. Stop the **mariadb** service:

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

2. Copy the data files to the required location:

   ```
   # cp -r /var/lib/mysql /backup-location
   ```

3. Optionally, copy the configuration files to the required location:

```
# cp -r /etc/my.cnf /etc/my.cnf.d /backup-location/configuration
```

4. Optionally, copy the log files to the required location:

```
# cp /var/log/mariadb/* /backup-location/logs
```

5. Start the **mariadb** service:

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

2.6.5. Replication as a backup solution

Replication is an alternative backup solution for source servers. If a source server replicates to a replica server, backups can be run on the replica without any impact on the source. The source can still run while you shut down the replica and back the data up from the replica.

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. It is recommended that you use any other backup solution on the replica together with this method.

Additional resources

- For information instructions regarding replicating a **MariaDB** database using **MariaDB Galera Cluster**, see Replicating MariaDB with Galera.
- For more information on replication as a backup solution, see MariaDB Documentation.

2.7. MIGRATING TO MARIADB 10.5

In RHEL 8, the **MariaDB** server is available in versions 10.3 and 10.5, each provided by a separate module stream. RHEL 9 provides **MariaDB 10.5** and **MySQL 8.0.** This part describes migration from a RHEL 8 version of **MariaDB 10.3** to RHEL 9 version of **MariaDB 10.5.**

2.7.1. Notable differences between MariaDB 10.3 and MariaDB 10.5

Significant changes between **MariaDB 10.3** and **MariaDB 10.5** include:

- **MariaDB** now uses the **unix_socket** authentication plug-in by default. The plug-in enables users to use operating system credentials when connecting to **MariaDB** through the local Unix socket file.
- **MariaDB** adds **mariadb-*** named binaries and **mysql*** symbolic links pointing to the **mariadb-*** binaires. For example, the **mysqladmin**, **mysqlaccess**, and **mysqlshow** symlinks point to the **mariadb-admin**, **mariadb-access**, and **mariadb-show** binaries, respectively.

- The **SUPER** privilege has been split into several privileges to better align with each user role. As a result, certain statements have changed required privileges.
- In parallel replication, the **slave_parallel_mode** now defaults to **optimistic**.
- In the **InnoDB** storage engine, defaults of the following variables have been changed: **innodb_adaptive_hash_index** to **OFF** and **innodb_checksum_algorithm** to **full_crc32**.
- **MariaDB** now uses the **libedit** implementation of the underlying software managing the **MariaDB** command history (the **.mysql_history** file) instead of the previously used **readline** library. This change impacts users working directly with the **.mysql_history** file. Note that **.mysql_history** is a file managed by the **MariaDB** or **MySQL** applications, and users should not work with the file directly. The human-readable appearance is coincidental.

NOTE

To increase security, you can consider not maintaining a history file. To disable the command history recording:

1. Remove the **.mysql_history** file if it exists.
2. Use either of the following approaches:
   - Set the **MYSQL_HISTFILE** variable to **/dev/null** and include this setting in any of your shell's startup files.
   - Change the **.mysql_history** file to a symbolic link to **/dev/null**:

```
$ ln -s /dev/null $HOME/.mysql_history
```

MariaDB Galera Cluster has been upgraded to version 4 with the following notable changes:

- **Galera** adds a new streaming replication feature, which supports replicating transactions of unlimited size. During an execution of streaming replication, a cluster replicates a transaction in small fragments.
- **Galera** now fully supports Global Transaction ID (GTID).
- The default value for the **wsrep_on** option in the **/etc/my.cnf.d/galera.cnf** file has changed from **1** to **0** to prevent end users from starting **wsrep** replication without configuring required additional options.

Changes to the PAM plug-in in **MariaDB 10.5** include:

- **MariaDB 10.5** adds a new version of the Pluggable Authentication Modules (PAM) plug-in. The PAM plug-in version 2.0 performs PAM authentication using a separate **setuid root** helper binary, which enables **MariaDB** to utilize additional PAM modules.
- The helper binary can be executed only by users in the **mysql** group. By default, the group contains only the **mysql** user. Red Hat recommends that administrators do not add more users to the **mysql** group to prevent password-guessing attacks without throttling or logging through this helper utility.
- In **MariaDB 10.5**, the Pluggable Authentication Modules (PAM) plug-in and its related files have been moved to a new package, **mariadb-pam**. As a result, no new **setuid root** binary is introduced on systems that do not use PAM authentication for **MariaDB**.

- The **mariadb-pam** package contains both PAM plug-in versions: version 2.0 is the default, and version 1.0 is available as the **auth_pam_v1** shared object library.
- The **mariadb-pam** package is not installed by default with the MariaDB server. To make the PAM authentication plug-in available in **MariaDB 10.5**, install the **mariadb-pam** package manually.

2.7.2. Migrating from a RHEL 8 version of MariaDB 10.3 to a RHEL 9 version of MariaDB 10.5

This procedure describes migrating from the **MariaDB 10.3** to the **MariaDB 10.5** using the **mariadb-upgrade** utility.

The **mariadb-upgrade** utility is provided by the **mariadb-server-utils** subpackage, which is installed as a dependency of the **mariadb-server** package.

Prerequisites

- Before performing the upgrade, back up all your data stored in the **MariaDB** databases.

Procedure

1. Ensure that the mariadb-server package is installed on the RHEL 9 system:

```
# yum install mariadb-server
```

2. Ensure that the mariadb service is not running on either of the source and target systems at the time of copying data:

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

3. Copy the data from the source location to the **/var/lib/mysql/** directory on the RHEL 9 target system.
4. Set the appropriate permissions and SELinux context for copied files on the target system:

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

5. Ensure that **mysql:mysql** is owner of all data in the **/var/lib/mysql**:

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

6. Adjust configuration so that option files located in **/etc/my.cnf.d/** include only options valid for **MariaDB 10.5**. For details, see upstream documentation for MariaDB 10.4 and MariaDB 10.5.
7. Start the **MariaDB** server on the target system.
   - When upgrading a database running standalone:

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

   - When upgrading a **Galera** cluster node:

```
# galera_new_cluster
```

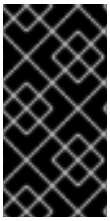
The **mariadb** service will be started automatically.

8. Execute the **mariadb-upgrade** utility to check and repair internal tables.
   - When upgrading a database running standalone:

     ```
     # mariadb-upgrade
     ```

   - When upgrading a **Galera** cluster node:

     ```
     # mariadb-upgrade --skip-write-binlog
     ```



IMPORTANT

There are certain risks and known problems related to an in-place upgrade. For example, some queries might not work or they will be run in different order than before the upgrade. For more information on these risks and problems, and for general information about an in-place upgrade, see MariaDB 10.5 Release Notes .

2.8. REPLICATING MARIADB WITH GALERA

This section describes how to replicate a MariaDB database using the Galera solution on Red Hat Enterprise Linux 9.

2.8.1. Introduction to MariaDB Galera Cluster

Galera replication is based on the creation of a synchronous multi-source **MariaDB Galera Cluster** consisting of multiple MariaDB servers. Unlike the traditional primary/replica setup where replicas are usually read-only, nodes in the MariaDB Galera Cluster can be all writable.

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

For more detailed information, see the upstream documentation:

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

2.8.2. Components to build MariaDB Galera Cluster

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.

Additional resources

For more detailed information, see upstream documentation:

- Galera Replication Library
- Galera Arbitrator
- mysql-wsrep project

2.8.3. Deploying MariaDB Galera Cluster

Prerequisites

- Install **MariaDB Galera Cluster** packages. For example:

```
# yum install galera
```

  As a result, the following packages are installed:

  - **mariadb-server-galera**
  - **mariadb-server**
  - **galera**
    The **mariadb-server-galera** package pulls the **mariadb-server** and **galera** packages as its dependency.

    For more information on components to build **MariaDB Galera Cluster**, see Section 2.8.2, "Components to build MariaDB Galera Cluster"

- The MariaDB server replication configuration must be updated 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 on how to set Galera Cluster address, see Galera Cluster Address.

Procedure

1. 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
```

2. 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.

Additional resources

- For more information, see Getting started with MariaDB Galera Cluster .

2.8.4. Adding a new node to MariaDB Galera Cluster

To add a new node to **MariaDB Galera Cluster**, use the following procedure.

Note that you can also use this procedure to reconnect an already existing node.

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

- For more information, see Getting started with MariaDB Galera Cluster .
- For detailed information on State Snapshot Transfers (SSTs), see Introduction to State Snapshot Transfers.

2.8.5. Restarting MariaDB Galera Cluster

If you shut down all nodes at the same time, you terminate 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 Section 2.8.3, "Deploying MariaDB Galera Cluster"

WARNING

If the cluster is not bootstrapped, and **mariadbd** 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

- For more information, see Getting started with MariaDB Galera Cluster .

CHAPTER 3. USING POSTGRESQL

This document describes the use of PostgreSQL in Red Hat Enterprise Linux 9.

3.1. GETTING STARTED WITH POSTGRESQL

The **PostgreSQL** server is an open source robust and highly-extensible database server based on the SQL language. It provides an object-relational database system, which allows you to manage extensive datasets and a high number of concurrent users. For these reasons, the PostgreSQL servers can be used in clusters to manage high amounts of data.

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

This part describes:

- How to install **PostgreSQL** in Installing PostgreSQL.
- Users, roles, and privileges in PostgreSQL users.
- How to adjust PostgreSQL configuration in Configuring PostgreSQL.
- How to back up your databases in Backing up PostgreSQL data.
- How to migrate to the RHEL 9 version of **PostgreSQL 13** in Migrating to a RHEL 9 version of PostgreSQL. One of the prerequisites of migration is performing a data backup.

3.2. INSTALLING POSTGRESQL

RHEL 9.0 provides **PostgreSQL 13** as the initial version of this Application Stream, which can be installed easily as an RPM package. Additional **PostgreSQL** versions will be provided as modules with a shorter life cycle in the future minor releases of RHEL 9.

To install **PostgreSQL**, use the following procedure.

Procedure

1. Install the **PostgreSQL** server packages:

```
# yum install postgresql-server
```

   The **postgres** superuser is created automatically.

2. Initialize the database cluster:

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

   Red Hat recommends storing the data in the default **/var/lib/pgsql/data** directory.

3. Start the **postgresql** service:

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

4. Enable the **postgresql** service to start at boot:

```
# systemctl enable postgresql.service
```

3.3. POSTGRESQL USERS

PostgreSQL users are of the following types:

- The **postgres** UNIX system user - should be used 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 **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 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

Red Hat recommends performing 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.

Additional resources

- For more information about database roles, see the PostgreSQL documentation.
- For more information about PostgreSQL privileges, see the PostgreSQL documentation.

3.4. CONFIGURING POSTGRESQL

In a **PostgreSQL** database, all data and configuration files are stored in a single directory called a database cluster. Red Hat recommends storing all data in the default **/var/lib/pgsql/data/** directory.

PostgreSQL configuration consists of the following files:

- **postgresql.conf** - is used for setting the database cluster parameters.
- **postgresql.auto.conf** - holds basic **PostgreSQL** settings similarly to **postgresql.conf**. However, this file is under the server control. It is edited by the **ALTER SYSTEM** queries, and cannot be edited manually.

- **pg_ident.conf** - is used for mapping user identities from external authentication mechanisms into the **PostgreSQL** user identities.
- **pg_hba.conf** - is used for configuring client authentication for **PostgreSQL** databases.

To change the **PostgreSQL** configuration, use the following procedure.

Procedure

1. Edit the respective configuration file, for example, **/var/lib/pgsql/data/postgresql.conf**.
2. Restart the **postgresql** service so that the changes become effective:

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

Example 3.1. Configuring PostgreSQL database cluster parameters

This example shows basic settings of the database cluster parameters in the **/var/lib/pgsql/data/postgresql.conf** file.

```
# This is a comment
log_connections = yes
log_destination = 'syslog'
search_path = '"$user", public'
shared_buffers = 128MB
```

Example 3.2. Setting client authentication in PostgreSQL

This example demonstrates how to set client authentication in the **/var/lib/pgsql/data/pg_hba.conf** file.

```
# 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
```

3.5. BACKING UP POSTGRESQL DATA

To back up **PostgreSQL** data, use one of the following approaches:

- SQL dump - see Section 3.5.1, “Backing up PostgreSQL data with an SQL dump”
- File system level backup - see Section 3.5.2, “Backing up PostgreSQL data with a file system level backup”
- Continuous archiving - see Section 3.5.3, “Backing up PostgreSQL data by continuous archiving”

3.5.1. Backing up PostgreSQL data with an SQL dump

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.

3.5.1.1. Advantages and disadvantages of an SQL dump

An SQL dump has the following advantages compared to other **PostgreSQL** backup methods:

- 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.

A disadvantage of an SQL dump is that it takes more time compared to file system level backup.

3.5.1.2. Performing an SQL dump using pg_dump

To dump a single database without cluster-wide information, use the **pg_dump** utility.

Prerequisites

- You must have read access to all tables that you want to dump. To dump the entire database, you must run the commands as the **postgres** superuser or a user with database administrator privileges.

Procedure

- Dump a database without cluster-wide information:

```
$ pg_dump dbname > dumpfile
```

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.

3.5.1.3. Performing an SQL dump using pg_dumpall

To dump each database in a given database cluster and to preserve cluster-wide data, use the **pg_dumpall** utility.

Prerequisites

- You must run the commands 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 > dumpfile
```

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.

3.5.1.4. Restoring a database dumped using pg_dump

To restore a database from an SQL dump that you dumped using the **pg_dump** utility, follow the steps below.

Prerequisites

- You must run the commands as the **postgres** superuser or a user with database administrator privileges.

Procedure

1. Create a new database:

```
$ createdb dbname
```

2. Make sure 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 recreate 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 dbname < dumpfile
```

where **dumpfile** 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
```

3.5.1.5. Restoring databases dumped using pg_dumpall

To restore data from a database cluster that you dumped using the **pg_dumpall** utility, follow the steps below.

Prerequisites

- You must run the commands as the **postgres** superuser or a user with database administrator privileges.

Procedure

1. Make sure 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 recreate 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 < dumpfile
```

where **dumpfile** is the output of the **pg_dumpall** command.

3.5.1.6. Performing an SQL dump of a database on another server

Dumping a database directly from one server to another is possible because **pg_dump** and **psql** can write to and read from pipes.

Procedure

- To dump a database from one server to another, run:

```
$ pg_dump -h host1 dbname | psql -h host2 dbname
```

3.5.1.7. Handling SQL errors during restore

By default, **psql** continues to execute if an SQL error occurs, causing the database to restore only partially.

To change the default behavior, use one of the following approaches when restoring a dump.

Prerequisites

- You must run the commands 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 dbname < dumpfile
```

- 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 using the **psql** utility:

```
$ psql -1
```

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

```
$ pg_restore -e
```

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

3.5.1.8. Additional resources

- For more information about the SQL dump, see PostgreSQL Documentation.

3.5.2. Backing up PostgreSQL data with a file system level backup

To perform a file system level backup, copy **PostgreSQL** database files to another location. For example, you can use any of the following approaches:

- Create an archive file using the **tar** utility.
- Copy the files to a different location using the **rsync** utility.
- Create a consistent snapshot of the data directory.

3.5.2.1. Advantages and disadvantages of a file system level backup

A file system level backup has the following advantage compared to other **PostgreSQL** backup methods:

- File system level backup is usually faster than an SQL dump.

File system level backup has the following disadvantages compared to other **PostgreSQL** backup methods:

- This backup method is not suitable when you want to upgrade from RHEL 8 to RHEL 9 and migrate your data to the upgraded system. File system level backup is architecture-specific and RHEL major-version-specific. You can restore your data on your RHEL 8 system if the upgrade is not successful but you cannot restore the data on a RHEL 9 system.
- The database server must be shut down before data backup and before data restore as well.
- Backup and restore of certain individual files or tables is impossible. The file system backups only work for a complete backup and restoration of an entire database cluster.

3.5.2.2. Performing a file system level backup

To perform a file system level backup, use the following procedure.

Procedure

1. Choose the location of a database cluster and initialize this cluster:

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

2. Stop the postgresql service:

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

3. Use any method to make a file system backup, for example a **tar** archive:

```
$ tar -cf backup.tar /var/lib/pgsql/data
```

4. Start the postgresql service:

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

3.5.2.3. Additional resources

- For more information about the file system level backup, see PostgreSQL Documentation.

3.5.3. Backing up PostgreSQL data by continuous archiving

3.5.3.1. Introduction to continuous archiving

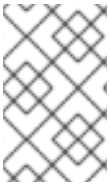
PostgreSQL records every change made to the database's data files into a write ahead log (WAL) file that is available in the **pg_wal/** subdirectory of the cluster's data directory. This log is intended primarily for a crash recovery. After a crash, the log entries made since the last checkpoint can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

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 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. Thus 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 thus 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.

To perform a database backup and restore using the continuous archiving method, follow these instructions:

1. Set up and test your procedure for archiving WAL files - see Section 3.5.3.3, "Setting up WAL archiving".
2. Perform a base backup - see Section 3.5.3.4, "Making a base backup" .

To restore your data, follow instructions in Section 3.5.3.5, "Restoring the database using a continuous archive backup".

3.5.3.2. Advantages and disadvantages of continuous archiving

Continuous archiving has the following advantages compared to other **PostgreSQL** backup methods:

- 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. Thus 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.

Continuous archiving has the following disadvantages compared to other **PostgreSQL** backup methods:

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

3.5.3.3. Setting up WAL archiving

A running **PostgreSQL** server produces a sequence of write ahead log (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.

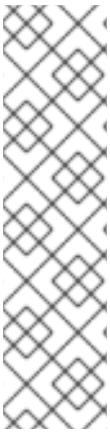
To enable WAL archiving, use the following procedure.

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.
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 non-zero 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.



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.

Example 3.3. Shell command for archiving WAL segments

This example shows a simple shell command you can set in the **archive_command** configuration parameter.

The following command copies a completed segment file to the required location:

```
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/00000001000000A900000065 && cp
pg_wal/00000001000000A900000065 /mnt/server/archivedir/00000001000000A900000065
```

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

Additional resources

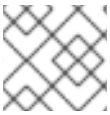
- For more information about setting WAL archiving, see PostgreSQL 10 Documentation.

3.5.3.4. Making a base backup

You can create a base backup in several ways. This section describes the simplest way of performing a base backup using the **pg_basebackup** utility on a running **PostgreSQL** server.

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.

To perform a base backup, use the following procedure.

Prerequisites

- You must run the commands 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 desired 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 desired backup location.

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

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.

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.

Additional resources

- For more information about making a base backup, see PostgreSQL Documentation.

- For more information about the **pg_basebackup** utility, see PostgreSQL Documentation.

3.5.3.5. Restoring the database using a continuous archive backup

To restore a database using a continuous backup, use the following procedure.

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.
   Make sure 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**/, recreate 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 will enter the recovery mode and proceed 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 accidental re-entering the recovery mode after it starts normal database operations.

9. Check the contents of the database to make sure 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.

For more information about restoring using the continuous backup, see PostgreSQL Documentation.

3.5.3.6. Additional resources

- For more information on continuous archiving method, see PostgreSQL Documentation.

3.6. MIGRATING TO A RHEL 9 VERSION OF POSTGRESQL

Red Hat Enterprise Linux 8 provides **PostgreSQL** in multiple module streams: **PostgreSQL 10** (the default postgresql stream), **PostgreSQL 9.6**, **PostgreSQL 12**, and **PostgreSQL 13**. In RHEL 9, **PostgreSQL 13** is available.

Users of **PostgreSQL** on Red Hat Enterprise Linux can use two migration paths for the database files:

- Fast upgrade using the pg_upgrade utility
- Dump and restore upgrade

The fast upgrade method is quicker than the dump and restore process. However, in certain cases, the fast upgrade does not work, and you can only use the dump and restore process, for example in case of cross-architecture upgrades.

As a prerequisite for migration to a later version of **PostgreSQL**, back up all your **PostgreSQL** databases.

Dumping the databases and performing backup of the SQL files is required for the dump and restore process and recommended for the fast upgrade method.

Before migrating to a later version of **PostgreSQL**, see the upstream compatibility notes for the version of **PostgreSQL** to which you want to migrate, as well as for all skipped **PostgreSQL** versions between the one you are migrating from and the target version.

3.6.1. Fast upgrade using the pg_upgrade utility

During a fast upgrade, you must copy binary data files to the **/var/lib/pgsql/data/** directory and use the **pg_upgrade** utility.

You can use this method for migrating data from the RHEL 8 version of **PostgreSQL 12** to the RHEL 9 version of **PostgreSQL 13**.

The following procedure describes migration from the RHEL 8 version of **PostgreSQL 12** to the RHEL 9 version of **PostgreSQL 13** using the fast upgrade method. For migration from **postgresql** streams other than **12**, use one of the following approaches:

- Update your **PostgreSQL** server to version 12 on RHEL 8 and then use the **pg_upgrade** utility to perform the fast upgrade to RHEL 9 version of **PostgreSQL 13**. See Migrating to a RHEL 9 version of PostgreSQL for more information.
- Use the dump and restore upgrade directly between any RHEL 8 version of **PostgreSQL** and **PGPostgreSQL 13** in RHEL 9.

Prerequisites

- Before performing the upgrade, back up all your data stored in the **PostgreSQL** databases. By default, all data is stored in the **/var/lib/pgsql/data/** directory on both the RHEL 8 and RHEL 9 systems.

Procedure

1. On the RHEL 9 system, install the **postgresql-server** and **postgresql-upgrade** packages:

```
# yum install postgresql-server postgresql-upgrade
```

   Optionally, if you used any **PostgreSQL** server modules on RHEL 8, install them also on the RHEL 9 system in two versions, compiled both against **PostgreSQL 12** (installed as the **postgresql-upgrade** package) and the target version of **PostgreSQL 13** (installed as the **postgresql-server** package). If you need to compile a third-party **PostgreSQL** server module, build it both against the **postgresql-devel** and **postgresql-upgrade-devel** packages.

2. Check the following items:
   - Basic configuration: On the RHEL 9 system, check whether your server uses the default **/var/lib/pgsql/data** directory and the database is correctly initialized and enabled. In addition, the data files must be stored in the same path as mentioned in the **/usr/lib/systemd/system/postgresql.service** file.
   - **PostgreSQL** servers: Your system can run multiple **PostgreSQL** servers. Make sure that the data directories for all these servers are handled independently.
   - **PostgreSQL** server modules: Ensure that the **PostgreSQL** server modules that you used on RHEL 8 are installed on your RHEL 9 system as well. Note that plug-ins are installed in the **/usr/lib64/pgsql/** directory.
3. Ensure that the **postgresql** service is not running on either of the source and target systems at the time of copying data.

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

4. Copy the database files from the source location to the **/var/lib/pgsql/data/** directory on the RHEL 9 system.

5. Perform the upgrade process by running the following command as the **PostgreSQL** user:

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

   This launches the **pg_upgrade** process in the background.

   In case of failure, **postgresql-setup** provides an informative error message.

6. Copy the prior configuration from **/var/lib/pgsql/data-old** to the new cluster.
   Note that the fast upgrade does not reuse the prior configuration in the newer data stack and the configuration is generated from scratch. If you want to combine the old and new configurations manually, use the *.conf files in the data directories.

7. Start the new **PostgreSQL** server:

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

8. Run the **analyze_new_cluster.sh** script located in the **PostgreSQL** home directory:

```
su postgres -c '~/analyze_new_cluster.sh'
```

9. If you want the new **PostgreSQL** server to be automatically started on boot, run:

```
# systemctl enable postgresql.service
```

3.6.2. Dump and restore upgrade

When using the dump and restore upgrade, you must dump all databases contents into an SQL file dump file. Note that the dump and restore upgrade is slower than the fast upgrade method and it may require some manual fixing in the generated SQL file.

You can use this method for migrating data from any RHEL 8 version of **PostgreSQL** to the RHEL 9 version of **PostgreSQL 13**.

On RHEL 8 and RHEL 9 systems, **PostgreSQL** data is stored in the **/var/lib/pgsql/data/** directory by default.

To perform the dump and restore upgrade, change the user to **root**.

The following procedure describes migration from the RHEL 8 default version of **Postgreql 10** to the RHEL 9 version of **PostgreSQL 13**.

Procedure

1. On your RHEL 8 system, start the **PostgreSQL 10** server:

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

2. On the RHEL 8 system, dump all databases contents into the **pgdump_file.sql** file:

```
su - postgres -c "pg_dumpall > ~/pgdump_file.sql"
```

3. Make sure that the databases were dumped correctly:

```
su - postgres -c 'less "$HOME/pgdump_file.sql"'
```

   As a result, the path to the dumped sql file is displayed: **/var/lib/pgsql/pgdump_file.sql**.

4. On the RHEL 9 system, install the **postgresql-server** package:

```
# yum install postgresql-server
```

   Optionally, if you used any **PostgreSQL** server modules on RHEL 8, install them also on the RHEL 9 system. If you need to compile a third-party **PostgreSQL** server module, build it against the **postgresql-devel** package.

5. On the RHEL 9 system, initialize the data directory for the new **PostgreSQL** server:

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

6. On the RHEL 9 system, copy the **pgdump_file.sql** into the **PostgreSQL** home directory, and check that the file was copied correctly:

```
su - postgres -c 'test -e "$HOME/pgdump_file.sql" && echo exists'
```

7. Copy the configuration files from the RHEL 8 system:

```
su - postgres -c 'ls -1 $PGDATA/*.conf'
```

   The configuration files to be copied are:

   - **/var/lib/pgsql/data/pg_hba.conf**
   - **/var/lib/pgsql/data/pg_ident.conf**
   - **/var/lib/pgsql/data/postgresql.conf**

8. On the RHEL 9 system, start the new **PostgreSQL** server:

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

9. On the RHEL 9 system, import data from the dumped sql file:

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