Red Hat Enterprise Linux 9

データベースサーバーの設定および使用

データベースサーバーでのデータのインストール、設定、バックアップ、および移行

Last Updated: 2024-06-26

Red Hat Enterprise Linux 9 データベースサーバーの設定および使用

データベースサーバーでのデータのインストール、設定、バックアップ、および移行

法律上の通知

概要

Red Hat Enterprise Linux 9 に MariaDB、MySQL、または PostgreSQL データベースサーバーをインストールします。選択したデータベースサーバーを設定し、データのバックアップを作成し、データを新しいバージョンのデータベースサーバーに移行します。

目次

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

Jira からのフィードバック送信 (アカウントが必要)

1. Jira の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
5. ダイアログの下部にある **Create** をクリックします。

第1章 データベースサーバーの概要

データベースサーバーは、データベース管理システム (DBMS) の機能を提供するサービスです。DBMS は、データベース管理のためのユーティリティーを提供し、エンドユーザー、アプリケーション、およびデータベースと対話します。

Red Hat Enterprise Linux 9 は、以下のデータベース管理システムを提供します。

- **MariaDB 10.5**
- **MariaDB 10.11** - RHEL 9.4 以降で利用可能
- **MySQL 8.0**
- **PostgreSQL 13**
- **PostgreSQL 15** - RHEL 9.2 以降で利用可能
- **PostgreSQL 16** - RHEL 9.4 以降で利用可能
- **Redis 6**

第2章 MARIADB の使用

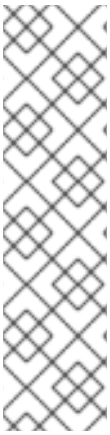
MariaDB サーバーは、MySQL テクノロジーに基づくオープンソースの高速で堅牢なデータベースサーバーです。MariaDB は、データを構造化情報に変換して、データにアクセスする SQL インターフェイスを提供するリレーショナルデータベースです。これには、複数のストレージエンジンとプラグインに加え、地理情報システム (GIS) と JavaScript Object Notation (JSON) 機能も含まれています。

RHEL システムに MariaDB をインストールして設定する方法、MariaDB データをバックアップする方法、MariaDB の以前のバージョンから移行する方法、および MariaDB Galera クラスター を使用してデータベースを複製する方法について説明します。

2.1. MARIADB のインストール

RHEL 9.0 は、この Application Stream の初期バージョンとして MariaDB 10.5 を提供します。これは、RPM パッケージとして簡単にインストールできます。追加の MariaDB バージョンは、RHEL 9 のマイナーリリースで、ライフサイクルが短いモジュールとして提供されます。

RHEL 9.4 で、**mariadb:10.11** モジュールストリームとして MariaDB 10.11 が導入されました。



注記

設計上、同じモジュールの複数のバージョン (ストリーム) を並行してインストールすることはできません。したがって、**mariadb** モジュールから利用可能なストリームのいずれかを選択する必要があります。コンテナー内では、別々のバージョンの MariaDB データベースサーバーを使用できます。コンテナー内で複数の MariaDB バージョンを実行する を参照してください。

RPM パッケージが競合しているため、RHEL 9 では MariaDB および MySQL データベースサーバーを同時にインストールすることはできません。コンテナー内では、MariaDB および MySQL データベースサーバーを並行して使用できます。コンテナー内で複数の MySQL および MariaDB バージョンを実行する を参照してください。

MariaDB をインストールするには、以下の手順を行います。

手順

1. MariaDB サーバーパッケージをインストールします。
   a. MariaDB 10.5 の場合は、RPM パッケージからインストールします。

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

   b. MariaDB 10.11 の場合は、**mariadb** モジュールからストリーム (バージョン) **11** を選択し、server プロファイルを指定します。以下に例を示します。

      ```
      # dnf module install mariadb:10.11/server
      ```

2. **mariadb** サービスを起動します。

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

3. **mariadb** サービスが、システムの起動時に起動するようにします。

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

2.1.1. コンテナー内で複数の MariaDB バージョンを実行する

同じホスト上で別々のバージョンの MariaDB を実行するには、コンテナー内で実行してください。同じモジュールの複数のバージョン (ストリーム) を並行してインストールすることはできないためです。

前提条件

- **container-tools** メタパッケージがインストールされている。

手順

1. Red Hat カスタマーポータルアカウントを使用して、**registry.redhat.io** レジストリーに認証します。

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

すでにコンテナーレジストリーにログインしている場合は、このステップをスキップしてください。

2. コンテナー内で MariaDB 10.5 を実行します。

```
$ podman run -d --name <container_name> -e
MYSQL_ROOT_PASSWORD=<mariadb_root_password> -p <host_port_1>:3306
rhel9/mariadb-105
```

このコンテナーイメージを使用する方法の詳細は、Red Hat Ecosystem Catalog を参照してください。

3. コンテナー内で MariaDB 10.11 を実行します。

```
$ podman run -d --name <container_name> -e
MYSQL_ROOT_PASSWORD=<mariadb_root_password> -p <host_port_2>:3306
rhel9/mariadb-1011
```

このコンテナーイメージを使用する方法の詳細は、Red Hat Ecosystem Catalog を参照してください。

注記

2 つのデータベースサーバーのコンテナー名とホストポートが異なっている必要があります。

4. クライアントがネットワーク上のデータベースサーバーにアクセスできるように、ファイアウォールでホストポートを開きます。

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

検証手順

1. 実行中のコンテナーに関する情報を表示します。

```
$ podman ps
```

2. データベースサーバーに接続し、root としてログインします。

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

関連情報

- コンテナーの構築、実行、および管理
- Red Hat Ecosystem Catalog でコンテナーを参照する

2.2. MARIADB の設定

MariaDB サーバーをネットワーク用に設定するには、以下の手順に従います。

手順

1. **/etc/my.cnf.d/mariadb-server.cnf** ファイルの **[mysqld]** セクションを編集します。以下の設定ディレクティブを設定できます。
   - **bind-address**: サーバーがリッスンするアドレスです。設定可能なオプションは以下のとおりです。
     - ホスト名
     - IPv4 アドレス
     - IPv6 アドレス
   - **skip-networking**: サーバーが TCP/IP 接続をリッスンするかどうかを制御します。以下の値が使用できます。
     - 0 - すべてのクライアントをリッスンする
     - 1 - ローカルクライアントのみをリッスンする
   - **port**: MariaDB が TCP/IP 接続をリッスンするポート。
2. **mariadb** サービスを再起動します。

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

2.3. MARIADB サーバーでの TLS 暗号化の設定

デフォルトでは、MariaDB は暗号化されていない接続を使用します。安全な接続のために、MariaDB サーバーで TLS サポートを有効にし、暗号化された接続を確立するようにクライアントを設定します。

2.3.1. MariaDB サーバーに CA 証明書、サーバー証明書、および秘密鍵を配置する

MariaDB サーバーで TLS 暗号化を有効にする前に、認証局 (CA) 証明書、サーバー証明書、および秘密鍵を MariaDB サーバーに保存します。

前提条件

- Privacy Enhanced Mail(PEM) 形式の以下のファイルがサーバーにコピーされています。
  - サーバーの秘密鍵: **server.example.com.key.pem**
  - サーバー証明書: **server.example.com.crt.pem**
  - 認証局 (CA) 証明書: **ca.crt.pem**

  秘密鍵および証明書署名要求 (CSR) の作成や、CA からの証明書要求に関する詳細は、CA のドキュメントを参照してください。

手順

1. CA およびサーバー証明書を **/etc/pki/tls/certs/** ディレクトリーに保存します。

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

2. MariaDB サーバーがファイルを読み込めるように、CA およびサーバー証明書にパーミッションを設定します。

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

   証明書は、セキュアな接続が確立される前は通信の一部であるため、任意のクライアントは認証なしで証明書を取得できます。そのため、CA およびサーバーの証明書ファイルに厳密なパーミッションを設定する必要はありません。

3. サーバーの秘密鍵を **/etc/pki/tls/private/** ディレクトリーに保存します。

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

4. サーバーの秘密鍵にセキュアなパーミッションを設定します。

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

   承認されていないユーザーが秘密鍵にアクセスできる場合は、MariaDB サーバーへの接続は安全ではなくなります。

5. SELinux コンテキストを復元します。

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

2.3.2. MariaDB サーバーでの TLS の設定

セキュリティーを向上させるには、MariaDB サーバーで TLS サポートを有効にします。その結果、クライアントは TLS 暗号化を使用してサーバーでデータを送信できます。

前提条件

- MariaDB サーバーをインストールしている。
- **mariadb** サービスが実行している。
- Privacy Enhanced Mail(PEM) 形式の以下のファイルがサーバー上にあり、**mysql** ユーザーが読み取りできます。
  - サーバーの秘密鍵: **/etc/pki/tls/private/server.example.com.key.pem**
  - サーバー証明書: **/etc/pki/tls/certs/server.example.com.crt.pem**
  - 認証局 (CA) 証明書 **/etc/pki/tls/certs/ca.crt.pem**
- サーバー証明書のサブジェクト識別名 (DN) またはサブジェクトの別名 (SAN) フィールドは、サーバーのホスト名と一致します。

手順

1. **/etc/my.cnf.d/mariadb-server-tls.cnf** ファイルを作成します。
   a. 以下の内容を追加して、秘密鍵、サーバー、および CA 証明書へのパスを設定します。

```
[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. 証明書失効リスト (CRL) がある場合は、それを使用するように MariaDB サーバーを設定します。

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

   c. オプション: 暗号化なしの接続試行を拒否します。この機能を有効にするには、以下を追加します。

```
require_secure_transport = on
```

   d. オプション: サーバーがサポートする必要がある TLS バージョンを設定します。たとえば、TLS 1.2 および TLS 1.3 をサポートするには、以下を追加します。

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

   デフォルトでは、サーバーは TLS 1.1、TLS 1.2、および TLS 1.3 をサポートします。

2. **mariadb** サービスを再起動します。

```
# systemctl restart mariadb
```

検証

トラブルシューティングを簡素化するには、ローカルクライアントが TLS 暗号化を使用するように設定する前に、MariaDB サーバーで以下の手順を実行します。

1. MariaDB で TLS 暗号化が有効になっていることを確認します。

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

have_ssl 変数が **yes** に設定されている場合、TLS 暗号化が有効になります。

2. MariaDB サービスが特定の TLS バージョンのみをサポートするように設定している場合は、**tls_version** 変数を表示します。

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

関連情報

- MariaDB サーバーに CA 証明書、サーバー証明書、および秘密鍵を配置する

2.3.3. 特定のユーザーアカウントに TLS で暗号化された接続を要求する

機密データにアクセスできるユーザーは、ネットワーク上で暗号化されていないデータ送信を回避するために、常に TLS で暗号化された接続を使用する必要があります。

すべての接続にセキュアなトランスポートが必要なサーバーで設定できない場合は (**require_secure_transport = on**)、TLS 暗号化を必要とするように個別のユーザーアカウントを設定します。

前提条件

- MariaDB サーバーで TLS サポートが有効になっている。
- セキュアなトランスポートを必要とするように設定するユーザーが存在する。

手順

1. 管理ユーザーとしてMariaDB サーバーに接続します。

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

管理ユーザーがリモートでサーバーにアクセスする権限を持たない場合は、MariaDB サーバーでコマンドを実行し、**localhost** に接続します。

2. **REQUIRE SSL** 句を使用して、ユーザーが TLS 暗号化接続を使用して接続する必要があるよう強制します。

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

検証

1. TLS 暗号化を使用して、**example** ユーザーとしてサーバーに接続します。

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

エラーが表示されず、インタラクティブな MariaDB コンソールにアクセスできる場合は、TLS との接続は成功します。

2. TLS を無効にして、**example** ユーザーとして接続を試みます。

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

このユーザーに TLS が必要だが無効になっているため、サーバーはログインの試行を拒否しました (**--skip-ssl**)。

関連情報

- MariaDB サーバーでの TLS 暗号化の設定

2.4. MARIADB クライアントでの TLS 暗号化のグローバルな有効化

MariaDB サーバーが TLS 暗号化に対応している場合は、安全な接続のみを確立し、サーバー証明書を検証するようにクライアントを設定します。この手順では、サーバー上のすべてのユーザーで TLS サポートを有効にする方法を説明します。

2.4.1. デフォルトで TLS 暗号化を使用するように MariaDB クライアントを設定する

RHEL では、MariaDB クライアントが TLS 暗号化を使用するようにグローバルに設定でき、サーバー証明書の Common Name (CN) が、ユーザーが接続するホスト名と一致することを検証します。これにより、man-in-the-middle 攻撃 (中間者攻撃) を防ぎます。

前提条件

- MariaDB サーバーで TLS サポートが有効になっている。
- サーバー証明書を発行した認証局 (CA) が RHEL で信頼されていない場合は、CA 証明書がクライアントにコピーされています。
- MariaDB サーバーが RHEL 9.2 以降を実行し、FIPS モードが有効になっている場合、このクライアントは Extended Master Secret (EMS) 拡張機能をサポートするか、TLS 1.3 を使用します。EMS を使用しない TLS 1.2 接続は失敗します。詳細は、ナレッジベースの記事 TLS extension "Extended Master Secret" enforced を参照してください。

手順

1. RHEL が、サーバー証明書を発行した CA を信頼しない場合は、以下を行います。
   a. CA 証明書を **/etc/pki/ca-trust/source/anchors/** ディレクトリーにコピーします。

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

b. すべてのユーザーが CA 証明書ファイルを読み取りできるようにするパーミッションを設定します。

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

c. CA 信頼データベースを再構築します。

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

2. 以下の内容で **/etc/my.cnf.d/mariadb-client-tls.cnf** ファイルを作成します。

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

これらの設定は、**MariaDB** クライアントが TLS 暗号化 (**ssl**) を使用し、クライアントがホスト名をサーバー証明書 (**ssl-verify-server-cert**) の CN と比較することを定義します。

検証

- ホスト名を使用してサーバーに接続し、サーバーの状態を表示します。

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

SSL エントリーに **Cipher in use is…** が含まれている場合、接続は暗号化されています。

このコマンドで使用するユーザーには、リモートで認証するパーミッションがあることに注意してください。

接続するホスト名がサーバーの TLS 証明書のホスト名と一致しない場合、**ssl-verify-server-cert** パラメーターにより接続が失敗します。たとえば、**localhost** に接続する場合は、以下のようになります。

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

関連情報

- **mysql(1)** man ページの **--ssl*** パラメーターの説明。

2.5. MARIADB データのバックアップ

Red Hat EnterpriseLinux 9 で **MariaDB** データベースからデータをバックアップする主な方法は 2 つあります。

- 論理バックアップ
- 物理バックアップ

論理バックアップ は、データの復元に必要な SQL ステートメントで設定されます。この種類のバックアップは、情報およびレコードをプレーンテキストファイルにエクスポートします。

物理バックアップに対する論理バックアップの主な利点は、移植性と柔軟性です。データは、物理バックアップではできない他のハードウェア設定である MariaDB バージョンまたはデータベース管理システム (DBMS) で復元できます。

mariadb.service が稼働している場合は、論理バックアップを実行できることに注意してください。論理バックアップには、ログと設定ファイルが含まれません。

物理バックアップ は、コンテンツを格納するファイルおよびディレクトリーのコピーで設定されます。

物理バックアップは、論理バックアップと比較して、以下の利点があります。

- 出力が少なくなる。
- バックアップのサイズが小さくなる。
- バックアップおよび復元が速くなる。
- バックアップには、ログファイルと設定ファイルが含まれる。

mariadb.service が実行していない場合や、データベースのすべてのテーブルがロックされていて、バックアップ中に変更しないようにする場合は、物理バックアップを実行する必要があります。

以下のいずれかの MariaDB バックアップ方法で、MariaDB データベースのデータのバックアップを使用できます。

- **mariadb-dump** を使用した論理バックアップ
- **Mariabackup** ユーティリティーを使用した物理的なオンラインバックアップ
- ファイルシステムのバックアップ
- バックアップソリューションとしてレプリケーションを使用

2.5.1. mariadb-dump を使用した論理バックアップの実行

mariadb-dumpクライアントはバックアップユーティリティーで、バックアップ目的でデータベースまたはデータベースの集合をダンプしたり、別のデータベースサーバーに転送したりできます。通常、mariadb-dump の出力は、サーバーテーブル構造を再作成する、それにデータを取り込む、またはその両方の SQL ステートメントで設定されます。mariadb-dump は、XML および (CSV などの) コンマ区切りテキスト形式など、他の形式でファイルを生成することもできます。

mariadb-dump バックアップを実行するには、以下のいずれかのオプションを使用できます。

- 選択したデータベースを 1 つまたは複数バックアップ
- すべてのデータベースをバックアップする。
- あるデータベースのテーブルのサブセットのバックアップを作成する。

手順

- 単一のデータベースをダンプするには、以下を実行します。

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

- 複数のデータベースを一度にダンプするには、次のコマンドを実行します。

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

- すべてのデータベースをダンプするには、以下を実行します。

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

- 1 つ以上のダンプされたフルデータベースをサーバーにロードし直すには、以下を実行します。

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

- データベースをリモート MariaDB サーバーにロードするには、以下を実行します。

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

- あるデータベースでテーブルのサブセットをダンプするには、**mariadb-dump** コマンドの末尾に、選択したテーブルのリストを追加します。

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

- 1 つのデータベースからダンプされたテーブルのサブセットをロードするには、以下を実行します。

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

注記

この時点で、db_name データベースが存在している必要があります。

- mariadb-dump がサポートするオプションのリストを表示するには、以下のコマンドを実行します。

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

関連情報

- MariaDB ドキュメント - mariadb-dump

2.5.2. Mariabackup ユーティリティーを使用した物理的なオンラインバックアップの実行

Mariabackup は、Percona XtraBackup テクノロジーをベースとしたユーティリティーです。これにより、InnoDB、Aria、および MyISAM テーブルの物理的なオンラインバックアップを実行できます。このユーティリティーは、AppStream リポジトリーから **mariadb-backup** パッケージで提供されます。

Mariabackup は、**MariaDB** サーバーの完全バックアップ機能に対応します。これには、暗号化されたデータおよび圧縮データが含まれます。

前提条件

- **mariadb-backup** パッケージがシステムにインストールされている。

```
# dnf install mariadb-backup
```

- Mariabackup には、バックアップを実行するユーザーの認証情報を指定する必要があります。認証情報はコマンドラインまたは設定ファイルで指定できます。
- Mariabackup のユーザーは、**RELOAD**、**LOCK TABLES**、および **REPLICATION CLIENT** の権限が必要です。

Mariabackup を使用してデータベースのバックアップを作成するには、以下の手順を行います。

手順

- コマンドラインで認証情報を提供する間にバックアップを作成するには、以下を実行します。

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

target-dir オプションは、バックアップファイルを格納するディレクトリーを定義します。完全バックアップを実行する場合は、ターゲットディレクトリーが空であるか、存在しない必要があります。

ユーザー オプションおよび **パスワード** オプションにより、ユーザー名とパスワードを設定できます。

- 設定ファイルに認証情報を設定してバックアップを作成するには、次のコマンドを実行します。

1. **/etc/my.cnf.d/** ディレクトリーに設定ファイルを作成します (例: **/etc/my.cnf.d/mariabackup.cnf**)。
2. 以下の行を新規ファイルの **[xtrabackup]** セクションまたは **[mysqld]** セクションに追加します。

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

3. バックアップを実行します。

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

関連情報

- Mariabackupによる完全バックアップと復元

2.5.3. Mariabackup ユーティリティーを使用したデータの復元

バックアップが完了したら、**mariabackup** コマンドに以下のいずれかのオプションを使用して、バックアップからデータを復元できます。

- **--copy-back** を使用すると、元のバックアップファイルを保持できます。
- **--move-back** は、バックアップファイルをデータディレクトリーに移動し、元のバックアップファイルを削除します。

Mariabackup ユーティリティーを使用してデータを復元するには、以下の手順に従います。

前提条件

- **mariadb** サービスが実行されていないことを確認します。

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

- データディレクトリーが空であることを確認します。
- Mariabackup のユーザーは、**RELOAD**、**LOCK TABLES**、および **REPLICATION CLIENT** の権限が必要です。

手順

1. **mariabackup** コマンドを実行します。
   - データを復元し、元のバックアップファイルを保持するには、**--copy-back** オプションを使用します。

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

   - データを復元し、元のバックアップファイルを削除するには、**--move-back** オプションを使用します。

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

2. ファイルの権限を修正します。
   データベースを復元するとき、Mariabackup は、バックアップのファイルおよびディレクトリーの権限を保持します。ただし、Mariabackup は、ユーザーおよびグループがデータベースを復元する際にファイルをディスクに書き込みます。バックアップの復元後、MariaDB サーバーのユーザーおよびグループ (通常は共に **mysql**) が一致するように、データディレクトリーの所有者の調整が必要になる場合があります。

   たとえば、ファイルの所有権を **mysql** ユーザーおよびグループに再帰的に変更するには、次のコマンドを実行します。

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

3. **mariadb** サービスを起動します。

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

関連情報

- Mariabackupによる完全バックアップと復元

2.5.4. ファイルシステムのバックアップの実行

MariaDB データファイルのファイルシステムバックアップを作成するには、MariaDB データディレクトリーの内容をバックアップ場所にコピーします。

現在の設定またはログファイルのバックアップも作成するには、以下の手順の中から任意の手順を選択します。

手順

1. **mariadb** サービスを停止します。

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

2. データファイルを必要な場所にコピーします。

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

3. 必要に応じて、設定ファイルを必要な場所にコピーします。

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

4. 必要に応じて、ログファイルを必要な場所にコピーします。

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

5. **mariadb** サービスを起動します。

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

6. バックアップされたデータをバックアップ場所から **/var/lib/mysql**ディレクトリーに読み込む際は、**mysql:mysql** が **/var/lib/mysql** 内のすべてのデータの所有者であることを確認してください。

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

2.5.5. バックアップソリューションとしてレプリケーションを使用

レプリケーションは、ソースサーバー用の代替バックアップソリューションです。ソースサーバーの複製となるレプリカサーバーを作成すると、ソースに影響を与えずにレプリカでバックアップを実行できます。ソースは、レプリカをシャットダウンする間に依然として実行でき、レプリカからデータのバックアップを作成できます。

警告

レプリケーション自体は、バックアップソリューションとしては十分ではありません。レプリケーションは、ハードウェア障害からソースサーバーを保護しますが、データ損失に対する保護は保証していません。この方法とともに、レプリカでその他のバックアップソリューションを使用することが推奨されます。

関連情報

- Galera で MariaDB を複製する

- バックアップソリューションとしてレプリケーションを使用

2.6. MARIADB 10.5 への移行

RHEL 8 では、バージョン 10.3、10.5、および 10.11 の **MariaDB** サーバーを利用できます。各バージョンは、それぞれ別のモジュールストリームによって提供されます。RHEL 9 では、**MariaDB 10.5**、**MariaDB 10.11**、および **MySQL 8.0** を使用できます。

ここでは、RHEL 8 バージョンの **MariaDB 10.3** から RHEL 9 バージョンの **MariaDB 10.5.** への移行を説明します。

2.6.1. MariaDB 10.3 と MariaDB 10.5 の主な相違点

MariaDB 10.3 と MariaDB 10.5 の間の重要な変更点には以下が含まれます。

- MariaDB がデフォルトで **unix_socket** 認証プラグインを使用するようになりました。このプラグインにより、ユーザーがローカルの UNIX ソケットファイルを介して MariaDB に接続するときにオペレーティングシステムの認証情報を使用できるようになります。

- **MariaDB** に、**mariadb-*** という名前のバイナリーと、**mariadb-*** バイナリーを参照する **mysql*** シンボリックリンクが追加されました。たとえば、**mysqladmin**、**mysqlaccess**、および **mysqlshow** のシンボリックリンクは、**mariadb-admin**、**mariadb-access**、および **mariadb-show** のバイナリーをそれぞれポイントします。

- 各ユーザーロールに合わせて、**SUPER** 特権が複数の特権に分割されました。その結果、一部のステートメントで必要な特権が変更されました。

- 並列のレプリケーションでは、**slave_parallel_mode** はデフォルトで **optimistic** に設定されるようになりました。

- InnoDB ストレージエンジンで、変数のデフォルトが変更されました (**innodb_adaptive_hash_index** は **OFF** へ、**innodb_checksum_algorithm** は **full_crc32** へ変更)。

- MariaDB は、以前使用された **readline** ライブラリーではなく、MariaDB コマンド履歴 (**.mysql_history** ファイル) を管理する基盤となるソフトウェアの **libedit** 実装を使用するようになりました。この変更は、**.mysql_history** ファイルを直接使用しているユーザーに影響します。**.mysql_history** は MariaDB または MySQL アプリケーションによって管理されるファイルであるため、ユーザーは直接ファイルでは機能しないことに注意してください。人間が判読可能な外観は偶然です。

注記

セキュリティーを強化するために、履歴ファイルの維持を考慮することができます。コマンド履歴の記録を無効にするには、以下を実行します。

1. 存在する場合は、**.mysql_history** ファイルを削除します。
2. 以下のいずれかの方法を使用します。
   - **MYSQL_HISTFILE** 変数を **/dev/null** に設定し、これをシェルの起動ファイルに追加します。
   - **.mysql_history** ファイルを **/dev/null** へのシンボリックリンクに変更します。

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

MariaDB Galera **クラスター** がバージョン 4 にアップグレードされ、以下の主な変更点が加えられました。

- Galera は、サイズ制限なしのトランザクションの複製をサポートする、新しいストリーミングレプリケーション機能を追加します。ストリーミングレプリケーションの実行時に、クラスターは小さなフラグメントでトランザクションを複製します。
- Galera が Global Transaction ID (GTID) に完全に対応するようになりました。
- **/etc/my.cnf.d/galera.cnf** ファイルの **wsrep_on** オプションのデフォルト値が **1** から **0** に変更され、エンドユーザーが必要な追加オプションを設定せずに **wsrep** レプリケーションを開始できないようにします。

MariaDB 10.5 の PAM プラグインが次のように変更されました。

- MariaDB 10.5 で、Pluggable Authentication Modules (PAM) プラグインの新しいバージョンが追加されました。PAM プラグインバージョン 2.0 は、個別の **setuid root** ヘルパーバイナリーを使用して PAM 認証を実行します。これにより、MariaDB が追加の PAM モジュールを使用できるようになります。
- ヘルパーバイナリーは、**mysql** グループのユーザーによってのみ実行できます。デフォルトでは、グループには **mysql** ユーザーのみが含まれます。Red Hat では、このヘルパーユーティリティーを介してスロットルまたはログの記録をせずにパスワード推測攻撃を防ぐために、管理者が **mysql** グループにユーザーをさらに追加しないことを推奨しています。
- MariaDB 10.5 で、Pluggable Authentication Modules (PAM) プラグインとその関連ファイルが新しいパッケージ **mariadb-pam** に移動しました。したがって、**MariaDB** に PAM 認証を使用しないシステムに、新しい **setuid root** バイナリーが導入されることはありません。
- **mariadb-pam** パッケージには、両方の PAM プラグインバージョンが含まれています。バージョン 2.0 はデフォルトで、バージョン 1.0 は **auth_pam_v1** 共有オブジェクトライブラリーとして利用できます。
- MariaDB サーバーでは、**mariadb-pam** パッケージはデフォルトでインストールされません。MariaDB 10.5 で PAM 認証プラグインを利用できるようにするには、**mariadb-pam** パッケージを手動でインストールします。

2.6.2. RHEL 8 バージョンの MariaDB 10.3 から、RHEL 9 バージョンの MariaDB 10.5 への移行

この手順では、**mariadb-upgrade** ユーティリティーを使用して、MariaDB 10.3 から MariaDB 10.5 に移行する方法を説明します。

mariadb-upgrade ユーティリティーは、**mariadb-server-utils** サブパッケージにより提供され、**mariadb-server** パッケージの依存関係としてインストールされます。

前提条件

- アップグレードを実行する前に、MariaDB データベースに保存されている全データのバックアップを作成します。

手順

1. **mariadb-server** パッケージが RHEL9 システムにインストールされていることを確認します。

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

2. データのコピー時に、**mariadb** サービスがソースおよびターゲットのシステムで稼働していないことを確認します。

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

3. ソースの場所から RHEL 9 ターゲットシステムの **/var/lib/mysql/** ディレクトリーにデータをコピーします。

4. ターゲットシステムでコピーされたファイルに適切なパーミッションと SELinux コンテキストを設定します。

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

5. **mysql:mysql** が **/var/lib/mysql** ディレクトリー内のすべてのデータの所有者であることを確認してください。

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

6. **/etc/my.cnf.d/** にあるオプションファイルに MariaDB 10.5 に対して有効なオプションのみが含まれるように、設定を調整します。詳細は、MariaDB 10.4 および MariaDB 10.5 のアップストリームドキュメントを参照してください。

7. ターゲットシステムで、MariaDB サーバーを起動します。

   - スタンドアロンを実行しているデータベースをアップグレードする場合:

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

   - Galera クラスターノードをアップグレードする場合:

   ```
   # galera_new_cluster
   ```

   mariadb サービスが自動的に起動します。

8. mariadb-upgrade ユーティリティーを実行して、内部テーブルをチェックし、修復します。

   - スタンドアロンを実行しているデータベースをアップグレードする場合:

```
$ mariadb-upgrade
```

- Galera クラスターノードをアップグレードする場合:

```
$ mariadb-upgrade --skip-write-binlog
```

重要

インプレースアップグレードには、特定のリスクと既知の問題があります。たとえば、一部のクエリーが動作しなかったり、アップグレード前とは異なる順序で実行される場合があります。これらのリスクと問題、およびインプレースアップグレードに関する全般的な情報は、MariaDB 10.5 Release Notes を参照してください。

2.7. MARIADB 10.5 から MARIADB 10.11 へのアップグレード

ここでは、RHEL 9 における MariaDB 10.5 から MariaDB 10.11 への移行について説明します。

2.7.1. MariaDB 10.5 と MariaDB 10.11 の主な相違点

MariaDB 10.5 と MariaDB 10.11 の間の重要な変更点は次のとおりです。

- 新しい **sys_schema** 機能。これは、データベースの使用状況に関する情報を提供するビュー、関数、およびプロシージャーのコレクションです。
- **CREATE TABLE**、**ALTER TABLE**、**RENAME TABLE**、**DROP TABLE**、**DROP DATABASE**、および関連するデータ定義言語 (DDL) ステートメントがアトミックになりました。ステートメントは完全に完結している必要があります。そうでない場合、変更が元に戻されます。**DROP TABLE** を使用して複数のテーブルを削除する場合、テーブルの全リストではなく、個々のドロップのみがアトミックであることに注意してください。
- 新しい **GRANT ... TO PUBLIC** 権限が利用可能になりました。
- **SUPER** 権限と **READ ONLY ADMIN** 権限が分離されました。
- 新しい **UUID** データベースデータ型に、ユニバーサル一意識別子を格納できるようになりました。
- MariaDB が Secure Socket Layer (SSL) プロトコルバージョン 3 をサポートするようになりました。
- MariaDB サーバーの起動に正しく設定された SSL が必要になりました。以前は、SSL の設定が誤っている場合、MariaDB は SSL を暗黙的に無効にし、セキュアでない接続を使用していました。
- MariaDB が **natural_sort_key()** 関数により自然なソート順序をサポートするようになりました。
- 新しい **SFORMAT** 関数を任意のテキスト書式設定に使用できるようになりました。
- **utf8** 文字セット (および関連する照合順序) が、デフォルトで **utf8mb3** のエイリアスになりました。
- MariaDB は、Unicode Collation Algorithm (UCA) 14 の照合順序をサポートしています。

- MariaDB の **systemd** ソケットのアクティベーションファイルが **/usr/share/** ディレクトリーで利用できるようになりました。アップストリームとは異なり、これらのファイルは RHEL のデフォルト設定の一部ではないことに注意してください。
- エラーメッセージに、**MySQL** ではなく **MariaDB** 文字列が含まれるようになりました。
- エラーメッセージが中国語で利用できるようになりました。
- デフォルトの logrotate ファイルが大幅に変更されました。MariaDB 10.11 に移行する前に設定を確認してください。
- MariaDB および MySQL クライアントの場合、コマンドラインで指定した接続プロパティー (例: **--port=3306**) によって、クライアントとサーバー間の通信のプロトコルタイプ (**tcp**、**socket**、**pipe**、**memory** など) が強制されるようになりました。たとえば、以前は MariaDB クライアントが UNIX ソケットを介して接続した場合、指定したポートが無視されていました。

2.7.2. RHEL 9 バージョンの MariaDB 10.5 から MariaDB 10.11 へのアップグレード

この手順では、**dnf** および **mariadb-upgrade** ユーティリティーを使用して、**mariadb-server** RPM パッケージで提供される MariaDB 10.5 から、**mariadb:10.11** モジュールストリームにアップグレードする方法について説明します。

mariadb-upgrade ユーティリティーは、**mariadb-server-utils** サブパッケージにより提供され、**mariadb-server** パッケージの依存関係としてインストールされます。

前提条件

- アップグレードを実行する前に、MariaDB データベースに保存されている全データのバックアップを作成します。

手順

1. MariaDB サーバーを停止します。

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

2. 非モジュールの MariaDB 10.5 からモジュールの MariaDB 10.11 に切り替えます。

```
# dnf module switch-to mariadb:10.11
```

3. **/etc/my.cnf.d/** にあるオプションファイルに MariaDB 10.11 に対して有効なオプションのみが含まれるように、設定を調整します。詳細は、MariaDB 10.6 および MariaDB 10.11 のアップストリームドキュメントを参照してください。
4. MariaDB サーバーを起動します。
   - スタンドアロンを実行しているデータベースをアップグレードする場合:

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

   - Galera クラスターノードをアップグレードする場合:

```
# galera_new_cluster
```

mariadb サービスが自動的に起動します。

5. **mariadb-upgrade** ユーティリティーを実行して、内部テーブルをチェックし、修復します。
   - スタンドアロンを実行しているデータベースをアップグレードする場合:

     ```
     # mariadb-upgrade
     ```

   - Galera クラスターノードをアップグレードする場合:

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

重要

インプレースアップグレードには、特定のリスクと既知の問題があります。たとえば、一部のクエリーが動作しなかったり、アップグレード前とは異なる順序で実行される場合があります。これらのリスクと問題、およびインプレースアップグレードに関する全般的な情報は、MariaDB 10.11 Release Notes を参照してください。

2.8. GALERA で MARIADB を複製する

Red Hat Enterprise Linux 9 の Galera ソリューションを使用して、MariaDB データベースをレプリケートできます。

2.8.1. MariaDB Galera クラスターの概要

Galera レプリケーションは、複数の MariaDB サーバーで設定される同期マルチソース **MariaDB Galera クラスター** の作成に基づいています。レプリカが通常読み取り専用である従来のプライマリー/レプリカ設定とは異なり、MariaDB Galera クラスターのノードはすべて書き込み可能にすることができます。

Galera レプリケーションと **MariaDB** データベースとの間のインターフェイスは、書き込みセットレプリケーション API (**wsrep API**) で定義されます。

MariaDB Galera クラスター の主な機能は以下のとおりです。

- 同期のレプリケーション
- アクティブ/アクティブのマルチソーストポロジー
- クラスターノードへの読み取りおよび書き込み
- 自動メンバーシップ制御、失敗したノードのクラスターからの削除
- 自動ノードの参加
- 行レベルの並列レプリケーション
- ダイレクトクライアント接続: ユーザーはクラスターノードにログインし、レプリケーションの実行中にノードを直接操作できます。

同期レプリケーションとは、サーバーがトランザクションに関連付けられた書き込みセットをクラスター内のすべてのノードにブロードキャストすることで、コミット時にトランザクションをレプリケートすることを意味します。クライアント (ユーザーアプリケーション) はデータベース管理システム (DBMS) に直接接続し、ネイティブの **MariaDB** と同様の動作が発生します。

同期レプリケーションは、クラスター内の1つのノードで発生した変更が、クラスター内の他のノードで同時に発生することを保証します。

そのため、同期レプリケーションには、非同期のレプリケーションと比べて次のような利点があります。

- 特定のクラスターノード間の変更の伝播に遅延がない
- すべてのクラスターノードには常に一貫性がある
- いずれかのクラスターノードがクラッシュしても、最新の変更は失われない
- すべてのクラスターノードのトランザクションが並列に実行する
- クラスター全体にわたる因果関係

関連情報

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

2.8.2. MariaDB Galera クラスターを構築するためのコンポーネント

MariaDB Galera **クラスター** を構築するには、システムに以下のパッケージをインストールする必要があります。

- **mariadb-server-galera**: MariaDB Galera **クラスター** のサポートファイルとスクリプトが含まれます。
- **mariadb-server**: MariaDB アップストリームがパッチを適用し、書き込みセットレプリケーション API (**wsrep API**) を組み込みます。この API は、**Galera** レプリケーションと **MariaDB** との間のインターフェイスを提供します。
- **galera**: MariaDB アップストリームがパッチを適用し、MariaDB の完全サポートを追加します。**galera** パッケージには、以下の内容が含まれます。
  - **Galera Replication Library** は、レプリケーション機能全体を提供します。
  - **Galera Arbitrator** ユーティリティーは、スプリットブレインのシナリオで投票に参加するクラスターメンバーとして使用できます。ただし、**Galera Arbitrator** は実際のレプリケーションには参加できません。
  - Galera Arbitrator ユーティリティーのデプロイに使用される **Galera Systemd service** および **Galera wrapper script**。RHEL 9 は、**/usr/lib/systemd/system/garbd.service** および **/usr/sbin/garb-systemd** にあるこれらのファイルのアップストリームバージョンを提供します。

関連情報

- Galera Replication Library
- Galera Arbitrator
- mysql-wsrep プロジェクト

2.8.3. MariaDB Galera クラスターのデプロイメント

MariaDB Galera Cluster パッケージをデプロイし、設定を更新できます。新しいクラスターを形成するには、クラスターの最初のノードをブートストラップする必要があります。

前提条件

- MariaDB Galera Cluster パッケージをインストールします。

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

その結果、次のパッケージが依存関係とともにインストールされます。

  - **mariadb-server-galera**
  - **mariadb-server**
  - **galera**
    MariaDB Galera Cluster を構築するのに必要なパッケージの詳細は、Components to build MariaDB Cluster を参照してください。

- MariaDB サーバーのレプリケーション設定は、システムを初めてクラスターに追加する前に更新する必要があります。
デフォルト設定は、**/etc/my.cnf.d/galera.cnf** ファイルで配布されます。

MariaDB Galera **クラスター** をデプロイする前に、以下の文字列で開始するように、すべてのノードの **/etc/my.cnf.d/galera.cnf** ファイルに **wsrep_cluster_address** オプションを設定します。

```
gcomm://
```

  - 初期ノードでは、**wsrep_cluster_address** を空のリストとして設定できます。

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

  - その他のすべてのノードに **wsrep_cluster_address** を設定して、実行中のクラスターに属するノードへのアドレスを追加します。以下に例を示します。

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

Galera Cluster アドレスの設定方法は、Galera Cluster Address を参照してください。

手順

1. ノードで以下のラッパーを実行して、新規クラスターの最初のノードをブートストラップします。

```
# galera_new_cluster
```

このラッパーにより、MariaDB サーバーデーモン (**mariadbd**) に **--wsrep-new-cluster** オプションが指定されて実行されるようになります。このオプションは、接続する既存クラスターがないという情報を提供します。したがって、ノードは新規 UUID を作成し、新しいクラスターを特定します。

注記

mariadb サービスは、複数の MariaDB サーバープロセスと対話する systemd メソッドをサポートします。したがって、複数の MariaDB サーバーを実行している場合は、インスタンス名を接尾辞として指定して、特定のインスタンスをブートストラップできます。

```
# galera_new_cluster mariadb@node1
```

2. 各ノードで次のコマンドを実行して、その他のノードをクラスターに接続します。

```
# systemctl start mariadb
```

その結果、ノードはクラスターに接続し、それ自体をクラスターの状態と同期します。

関連情報

- Getting started with MariaDB Galera Cluster

2.8.4. 新規ノードの MariaDB Galera クラスターへの追加

新規ノードを **MariaDB Galera クラスター** に追加するには、以下の手順に従います。

この手順に従って、既存のノードを再接続することもできます。

手順

- 特定のノードで、**/etc/my.cnf.d/galera.cnf** 設定ファイルの **[mariadb]** セクション内にある **wsrep_cluster_address** オプションで、1 つ以上の既存クラスターメンバーにアドレスを指定します。

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

新規ノードを既存クラスターノードのいずれかに接続すると、クラスター内のすべてのノードを表示できるようになります。

ただし、**wsrep_cluster_address** のクラスターの全ノードを表示することが推奨されます。

したがって、1 つ以上のクラスターノードがダウンしても、その他のクラスターノードに接続することでノードがクラスターに参加できます。すべてのメンバーがメンバーシップに同意すると、クラスターの状態が変更します。新規ノードの状態がクラスターの状態と異なる場合、新しいノードは Incremental State Transfer (IST) または State Snapshot Transfer (SST) のいずれかを要求し、他のノードとの一貫性を確保します。

関連情報

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

2.8.5. MariaDB Galera クラスターの再起動

すべてのノードを同時にシャットダウンすると、クラスターが終了し、実行中のクラスターは存在しなくなります。ただし、クラスターのデータは引き続き存在します。

クラスターを再起動するには、MariaDB Galera クラスターの設定 の説明に従って、最初のノードをブートストラップします。

警告

クラスターがブートストラップされず、最初のノードの **mariadbd** が **systemctl start mariadb** コマンドでのみ起動した場合、ノードは **/etc/my.cnf.d/galera.cnf** ファイルの **wsrep_cluster_address** オプションに記載されている少なくとも 1 つのノードに接続しようとします。ノードが現在実行していない場合は、再起動に失敗します。

関連情報

- Getting started with MariaDB Galera Cluster .

2.9. MARIADB クライアントアプリケーションの開発

Red Hat では、**MariaDB** クライアントライブラリーに対して **MariaDB** クライアントアプリケーションを開発することを推奨します。

MariaDB クライアントライブラリーに対してアプリケーションをビルドするために必要な開発ファイルとプログラムは、**mariadb-connector-c-devel** パッケージで提供されます。

直接ライブラリー名を使用する代わりに、**mariadb-connector-c-devel** パッケージで配布されている **mariadb_config** プログラムを使用します。このプログラムにより、正しいビルドフラグが確実に返されるようになります。

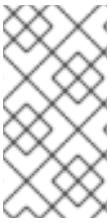
第3章 MYSQL の使用

MySQL サーバーは、オープンソースの高速で堅牢なデータベースサーバーです。**MySQL** は、データを構造化情報に変換して、データにアクセスする SQL インターフェイスを提供するリレーショナルデータベースです。これには、複数のストレージエンジンとプラグインに加え、地理情報システム (GIS) と JavaScript Object Notation (JSON) 機能も含まれています。

RHEL システムに **MySQL** をインストールして設定する方法、**MySQL** データをバックアップする方法、**MySQL** の以前のバージョンから移行する方法、および **MySQL** を複製する方法について説明します。

3.1. MYSQL のインストール

RHEL 9.0 は、この Application Stream の初期バージョンとして **MySQL 8.0** を提供します。これは、RPM パッケージとして簡単にインストールできます。



注記

RPM パッケージが競合しているため、RHEL 9 では **MySQL** および **MariaDB** データベースサーバーを同時にインストールすることはできません。コンテナー内では、**MySQL** および **MariaDB** データベースサーバーを並行して使用できます。コンテナー内で複数の MySQL および MariaDB バージョンを実行する を参照してください。

MySQL をインストールするには、以下の手順に従います。

手順

1. **MySQL** サーバーパッケージをインストールします。

```
# dnf install mysql-server
```

2. **mysqld** サービスを開始します。

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

3. **mysqld** サービスを有効にして、起動時に起動するようにします。

```
# systemctl enable mysqld.service
```

4. **推奨手順: MySQL** のインストール時にセキュリティーを強化するには、次のコマンドを実行します。

```
$ mysql_secure_installation
```

このコマンドは、完全にインタラクティブなスクリプトを起動して、プロセスの各ステップのプロンプトを表示します。このスクリプトを使用すると、次の方法でセキュリティーを改善できます。

- root アカウントのパスワードの設定
- 匿名ユーザーの削除
- リモート root ログインの拒否 (ローカルホスト外)

3.1.1. コンテナー内で複数の MySQL および MariaDB バージョンを実行する

MySQL と MariaDB の両方を同じホストで実行するには、コンテナー内で実行します。これは、RPM パッケージが競合し、これらのデータベースサーバーを並行してインストールできないためです。

この手順では、例として MySQL 8.0 と MariaDB 10.5 を記載していますが、Red Hat Ecosystem Catalog で利用可能な任意の MySQL または MariaDB コンテナーバージョンを使用できます。

前提条件

- **container-tools** メタパッケージがインストールされている。

手順

1. Red Hat カスタマーポータルアカウントを使用して、**registry.redhat.io** レジストリーに認証します。

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

すでにコンテナーレジストリーにログインしている場合は、このステップをスキップしてください。

2. コンテナー内で MySQL 8.0 を実行します。

```
$ podman run -d --name <container_name> -e
MYSQL_ROOT_PASSWORD=<mysql_root_password> -p <host_port_1>:3306
rhel9/mysql-80
```

このコンテナーイメージを使用する方法の詳細は、Red Hat Ecosystem Catalog を参照してください。

3. コンテナー内で MariaDB 10.5 を実行します。

```
$ podman run -d --name <container_name> -e
MYSQL_ROOT_PASSWORD=<mariadb_root_password> -p <host_port_2>:3306
rhel9/mariadb-105
```

このコンテナーイメージを使用する方法の詳細は、Red Hat Ecosystem Catalog を参照してください。

4. コンテナー内で MariaDB 10.11 を実行します。

```
$ podman run -d --name <container_name> -e
MYSQL_ROOT_PASSWORD=<mariadb_root_password> -p <host_port_3>:3306
rhel9/mariadb-1011
```

このコンテナーイメージを使用する方法の詳細は、Red Hat Ecosystem Catalog を参照してください。

注記

2 つのデータベースサーバーのコンテナー名とホストポートが異なっている必要があります。

5. クライアントがネットワーク上のデータベースサーバーにアクセスできるように、ファイアウォールでホストポートを開きます。

```
# firewall-cmd --permanent --add-port=
{<host_port_1>/tcp,<host_port_2>/tcp,<host_port_3>/tcp,...}
# firewall-cmd --reload
```

検証手順

1. 実行中のコンテナーに関する情報を表示します。

```
$ podman ps
```

2. データベースサーバーに接続し、root としてログインします。

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

関連情報

- コンテナーの構築、実行、および管理
- Red Hat Ecosystem Catalog でコンテナーを参照する

3.2. MYSQL の設定

MySQL サーバーをネットワーク用に設定するには、以下の手順に従います。

手順

1. **/etc/my.cnf.d/mysql-server.cnf** ファイルの **[mysqld]** セクションを編集します。以下の設定ディレクティブを設定できます。
   - **bind-address**: サーバーがリッスンするアドレスです。設定可能なオプションは以下のとおりです。
     - ホスト名
     - IPv4 アドレス
     - IPv6 アドレス
   - **skip-networking**: サーバーが TCP/IP 接続をリッスンするかどうかを制御します。以下の値が使用できます。
     - 0 - すべてのクライアントをリッスンする
     - 1 - ローカルクライアントのみをリッスンする
   - **port**: MySQL が TCP/IP 接続をリッスンするポート。
2. **mysqld** サービスを再起動します。

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

3.3. MYSQL サーバーでの TLS 暗号化の設定

デフォルトでは、MySQL は暗号化されていない接続を使用します。安全な接続のために、MySQL サーバーで TLS サポートを有効にし、暗号化された接続を確立するようにクライアントを設定します。

3.3.1. MySQL サーバーに CA 証明書、サーバー証明書、および秘密鍵を配置する

MySQL サーバーで TLS 暗号化を有効にする前に、認証局 (CA) 証明書、サーバー証明書、および秘密鍵を MySQL サーバーに保存します。

前提条件

- Privacy Enhanced Mail(PEM) 形式の以下のファイルがサーバーにコピーされています。
  - サーバーの秘密鍵: **server.example.com.key.pem**
  - サーバー証明書: **server.example.com.crt.pem**
  - 認証局 (CA) 証明書: **ca.crt.pem**

  秘密鍵および証明書署名要求 (CSR) の作成や、CA からの証明書要求に関する詳細は、CA のドキュメントを参照してください。

手順

1. CA およびサーバー証明書を **/etc/pki/tls/certs/** ディレクトリーに保存します。

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

2. MySQL サーバーがファイルを読み込めるように、CA およびサーバー証明書にパーミッションを設定します。

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

   証明書は、セキュアな接続が確立される前は通信の一部であるため、任意のクライアントは認証なしで証明書を取得できます。そのため、CA およびサーバーの証明書ファイルに厳密なパーミッションを設定する必要はありません。

3. サーバーの秘密鍵を **/etc/pki/tls/private/** ディレクトリーに保存します。

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

4. サーバーの秘密鍵にセキュアなパーミッションを設定します。

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

   承認されていないユーザーが秘密鍵にアクセスできる場合は、MySQL サーバーへの接続は安全ではなくなります。

5. SELinux コンテキストを復元します。

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

3.3.2. MySQL サーバーでの TLS の設定

セキュリティーを強化するには、MySQL サーバーで TLS サポートを有効にします。その結果、クライアントは TLS 暗号化を使用してサーバーでデータを送信できます。

前提条件

- MySQL サーバーをインストールしている。
- **mysqld** サービスが実行されている。
- Privacy Enhanced Mail(PEM) 形式の以下のファイルがサーバー上にあり、**mysql** ユーザーが読み取りできます。
  - サーバーの秘密鍵: **/etc/pki/tls/private/server.example.com.key.pem**
  - サーバー証明書: **/etc/pki/tls/certs/server.example.com.crt.pem**
  - 認証局 (CA) 証明書 **/etc/pki/tls/certs/ca.crt.pem**
- サーバー証明書のサブジェクト識別名 (DN) またはサブジェクトの別名 (SAN) フィールドは、サーバーのホスト名と一致します。

手順

1. **/etc/my.cnf.d/mysql-server-tls.cnf** ファイルを作成します。
   a. 以下の内容を追加して、秘密鍵、サーバー、および CA 証明書へのパスを設定します。

```
[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. 証明書失効リスト (CRL) がある場合は、それを使用するように MySQL サーバーを設定します。

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

   c. オプション: 暗号化なしの接続試行を拒否します。この機能を有効にするには、以下を追加します。

```
require_secure_transport = on
```

   d. オプション: サーバーがサポートする必要がある TLS バージョンを設定します。たとえば、TLS 1.2 および TLS 1.3 をサポートするには、以下を追加します。

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

   デフォルトでは、サーバーは TLS 1.1、TLS 1.2、および TLS 1.3 をサポートします。

2. **mysqld** サービスを再起動します。

```
# systemctl restart mysqld
```

検証

トラブルシューティングを簡素化するには、ローカルクライアントが TLS 暗号化を使用するように設定する前に、**MySQL** サーバーで以下の手順を実行します。

1. **MySQL** で TLS 暗号化が有効になっていることを確認します。

```
# 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. **MySQL** サーバーが特定の TLS バージョンのみをサポートするように設定している場合は、**tls_version** 変数を表示します。

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

3. サーバーが正しい CA 証明書、サーバー証明書、および秘密鍵ファイルを使用していることを確認します。

```
# mysql -u root -e "SHOW GLOBAL VARIABLES WHERE Variable_name REGEXP '^ssl_ca|^ssl_cert|^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 |
+-----------------+-------------------------------------------------+
```

関連情報

- MySQL サーバーに CA 証明書、サーバー証明書、および秘密鍵を配置する

3.3.3. 特定のユーザーアカウントに TLS で暗号化された接続を要求する

機密データにアクセスできるユーザーは、ネットワーク上で暗号化されていないデータ送信を回避するために、常に TLS で暗号化された接続を使用する必要があります。

すべての接続にセキュアなトランスポートが必要なサーバーで設定できない場合は (**require_secure_transport = on**)、TLS 暗号化を必要とするように個別のユーザーアカウントを設定します。

前提条件

- MySQL サーバーで TLS サポートが有効になっている。
- セキュアなトランスポートを必要とするように設定するユーザーが存在する。
- CA 証明書がクライアントに保存されている。

手順

1. 管理ユーザーとして MySQL サーバーに接続します。

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

管理ユーザーがリモートでサーバーにアクセスする権限を持たない場合は、MySQL サーバーでコマンドを実行し、**localhost** に接続します。

2. **REQUIRE SSL** 句を使用して、ユーザーが TLS 暗号化接続を使用して接続する必要があるよう強制します。

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

検証

1. TLS 暗号化を使用して、**example** ユーザーとしてサーバーに接続します。

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

エラーが表示されず、インタラクティブな MySQL コンソールにアクセスできる場合は、TLS との接続は成功します。

デフォルトでは、サーバーが TLS 暗号化を提供している場合、クライアントは自動的にその TLS 暗号化を使用します。したがって、**--ssl-ca=ca.crt.pem** および **--ssl-mode=VERIFY_IDENTITY** オプションは必須ではありません。ただし、これらのオプションを使用するとクライアントはサーバーの ID を検証するため、セキュリティーが向上します。

2. TLS を無効にして、**example** ユーザーとして接続を試みます。

```
# 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)
```

このユーザーには TLS が必要なのにもかかわらず無効になっているため、サーバーはログインの試行を拒否しました (**--ssl-mode=DISABLED**)。

関連情報

- MySQL サーバーでの TLS の設定

3.4. MYSQL クライアントで CA 証明書の検証を使用して TLS 暗号化をグローバルで有効にする

MySQL サーバーが TLS 暗号化に対応している場合は、安全な接続のみを確立し、サーバー証明書を検証するようにクライアントを設定します。この手順では、サーバー上のすべてのユーザーで TLS サポートを有効にする方法を説明します。

3.4.1. デフォルトで TLS 暗号化を使用するように MySQL クライアントを設定する

RHEL では、MySQL クライアントが TLS 暗号化を使用するようにグローバルに設定でき、サーバー証明書の Common Name (CN) が、ユーザーが接続するホスト名と一致することを検証します。これにより、man-in-the-middle 攻撃 (中間者攻撃) を防ぎます。

前提条件

- MySQL サーバーで TLS サポートが有効になっている。
- CA 証明書は、クライアントの **/etc/pki/tls/certs/ca.crt.pem** ファイルに保存されます。

手順

- 以下の内容で **/etc/my.cnf.d/mysql-client-tls.cnf** ファイルを作成します。

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

これらの設定は、MySQL クライアントが TLS 暗号化を使用すること、およびクライアントがホスト名をサーバー証明書の CN と比較すること (**ssl-mode=VERIFY_IDENTITY**) を定義します。さらに、CA 証明書 (**ssl-ca**) へのパスも指定します。

検証

- ホスト名を使用してサーバーに接続し、サーバーの状態を表示します。

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

SSL エントリーに **Cipher in use is…** が含まれている場合、接続は暗号化されています。

このコマンドで使用するユーザーには、リモートで認証するパーミッションがあることに注意してください。

接続するホスト名がサーバーの TLS 証明書のホスト名と一致しない場合、**ssl-mode=VERIFY_IDENTITY** パラメーターにより接続が失敗します。たとえば、**localhost** に接続する場合は、以下のようになります。

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

関連情報

- **mysql(1)** man ページの **--ssl*** パラメーターの説明。

3.5. MYSQL データのバックアップ

Red Hat Enterprise Linux 9 で**MySQL** データベースからデータをバックアップする主な方法は 2 つあります。

- 論理バックアップ
- 物理バックアップ

論理バックアップ は、データの復元に必要な SQL ステートメントで設定されます。この種類のバックアップは、情報およびレコードをプレーンテキストファイルにエクスポートします。

物理バックアップに対する論理バックアップの主な利点は、移植性と柔軟性です。データは、物理バックアップではできない他のハードウェア設定である **MySQL** バージョンまたはデータベース管理システム (DBMS) で復元できます。

mysqld.service が実行されている場合は、論理バックアップを実行できることに注意してください。論理バックアップには、ログと設定ファイルが含まれません。

物理バックアップ は、コンテンツを格納するファイルおよびディレクトリーのコピーで設定されます。

物理バックアップは、論理バックアップと比較して、以下の利点があります。

- 出力が少なくなる。
- バックアップのサイズが小さくなる。
- バックアップおよび復元が速くなる。
- バックアップには、ログファイルと設定ファイルが含まれる。

mysqld.service が実行されていない場合、またはバックアップ中の変更を防ぐためにデータベース内のすべてのテーブルがロックされている場合は、物理バックアップを実行する必要があることに注意してください。

以下の **MySQL** バックアップアプローチのいずれかを使用して、**MySQL** データベースからデータをバックアップできます。

- **mysqldump** を使用した論理バックアップ
- ファイルシステムのバックアップ
- バックアップソリューションとしてレプリケーションを使用

3.5.1. mysqldump を使用した論理バックアップの実行

mysqldump クライアントはバックアップユーティリティーで、バックアップ目的でデータベースまたはデータベースの集合をダンプしたり、別のデータベースサーバーに転送したりできます。通常、**mysqldump** の出力は、サーバーテーブル構造を再作成する、それにデータを取り込む、またはその両方の SQL ステートメントで設定されます。**mysqldump** は、XML および (CSV などの) コンマ区切りテキスト形式など、他の形式でファイルを生成することもできます。

mysqldump バックアップを実行するには、以下のいずれかのオプションを使用できます。

- 選択したデータベースを 1 つまたは複数バックアップ
- すべてのデータベースをバックアップする。

- あるデータベースのテーブルのサブセットのバックアップを作成する。

手順

- 単一のデータベースをダンプするには、以下を実行します。

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

- 複数のデータベースを一度にダンプするには、次のコマンドを実行します。

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

- すべてのデータベースをダンプするには、以下を実行します。

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

- 1 つ以上のダンプされたフルデータベースをサーバーにロードし直すには、以下を実行します。

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

- データベースをリモート MySQL サーバーにロードするには、以下を実行します。

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

- あるデータベースでリテラルなテーブルのサブセットをダンプするには、**mysqldump** コマンドの末尾に、選択したテーブルのリストを追加します。

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

- 1 つのデータベースからダンプされたリテラルなテーブルのサブセットをロードするには、次のコマンドを実行します。

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

注記

この時点で、db_name データベースが存在している必要があります。

- mysqldump がサポートするオプションのリストを表示するには、以下を実行します。

```
$ mysqldump --help
```

関連情報

- mysqldump を使用した論理バックアップ

3.5.2. ファイルシステムのバックアップの実行

MySQL データファイルのファイルシステムバックアップを作成するには、MySQL データディレクトリーの内容をバックアップ場所にコピーします。

現在の設定またはログファイルのバックアップも作成するには、以下の手順の中から任意の手順を選択します。

手順

1. **mysqld** サービスを停止します。

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

2. データファイルを必要な場所にコピーします。

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

3. 必要に応じて、設定ファイルを必要な場所にコピーします。

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

4. 必要に応じて、ログファイルを必要な場所にコピーします。

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

5. **mysqld** サービスを開始します。

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

6. バックアップされたデータをバックアップ場所から **/var/lib/mysql**ディレクトリーに読み込む際は、**mysql:mysql** が **/var/lib/mysql** 内のすべてのデータの所有者であることを確認してください。

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

3.5.3. バックアップソリューションとしてレプリケーションを使用

レプリケーションは、ソースサーバー用の代替バックアップソリューションです。ソースサーバーの複製となるレプリカサーバーを作成すると、ソースに影響を与えずにレプリカでバックアップを実行できます。ソースは、レプリカをシャットダウンする間に依然として実行でき、レプリカからデータのバックアップを作成できます。

MySQLデータベースを複製する方法の手順については、MySQL の複製 を参照してください。

警告

レプリケーション自体は、バックアップソリューションとしては十分ではありません。レプリケーションは、ハードウェア障害からソースサーバーを保護しますが、データ損失に対する保護は保証していません。この方法とともに、レプリカでその他のバックアップソリューションを使用することが推奨されます。

関連情報

- MySQL replication documentation

3.6. MYSQL8.0 の RHEL9 バージョンへの移行

RHEL 8 には、MySQL データベースファミリーのサーバーの **MySQL 8.0**、**MariaDB 10.3**、および**MariaDB 10.5** の実装が含まれています。RHEL 9 は、 **MySQL 8.0** および **MariaDB 10.5** を提供します。

この手順では、**mysql_upgrade**ユーティリティーを使用して、RHEL 8 バージョンの **MySQL 8.0** から RHEL 9 バージョンの **MySQL 8.0** への移行について説明します。**mysql_upgrade** ユーティリティーは、**mysql-server** パッケージによって提供されます。

前提条件

- アップグレードを実行する前に、**MySQL** データベースに保存されているすべてのデータをバックアップすること。MySQL データのバックアップ を参照してください。

手順

1. **mysql-server** パッケージが RHEL9 システムにインストールされていることを確認します。

```
# dnf install mysql-server
```

2. データのコピー時に、**mysqld** サービスがソースシステムとターゲットシステムのどちらでも実行されていないことを確認してください。

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

3. ソースの場所から RHEL 9 ターゲットシステムの **/var/lib/mysql/** ディレクトリーにデータをコピーします。

4. ターゲットシステムでコピーされたファイルに適切なパーミッションと SELinux コンテキストを設定します。

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

5. **mysql:mysql** が、**/var/lib/mysql** ディレクトリー内のすべてのデータの所有者であることを確認してください。

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

6. ターゲットシステムで **MySQL** サーバーを起動します。

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

注意: **MySQL** の以前のバージョンでは、内部テーブルをチェックおよび修復するために **mysql_upgrade** コマンドが必要でした。これは、サーバーの起動時に自動的に実行されるようになりました。

3.7. MYSQL の複製

MySQL には、基本的なものから高度なものまで、レプリケーション用のさまざまな設定オプションが用意されています。このセクションでは、グローバルトランザクション識別子 (GTID) を使用して、新

しくインストールした MySQL サーバーに **MySQL** でレプリケートするトランザクションベースの方法について説明します。GTID を使用すると、トランザクションの識別と整合性の検証が簡素化されます。

MySQL でレプリケーションを設定するには、以下を行う必要があります。

- ソースサーバーを設定する
- レプリカサーバーを設定する
- ソースサーバーにレプリケーションユーザーを作成する
- レプリカサーバーをソースサーバーに接続する

重要

レプリケーションに既存の **MySQL** サーバーを使用する場合は、最初にデータを同期する必要があります。詳細は、アップストリームのドキュメント を参照してください。

3.7.1. MySQL ソースサーバーの設定

MySQL ソースサーバーがデータベースサーバーで行われたすべての変更を適切に実行および複製するために必要な設定オプションを設定できます。

前提条件

- ソースサーバーがインストールされている。

手順

1. **[mysqld]** セクションの **/etc/my.cnf.d/mysql-server.cnf** ファイルに以下のオプションを含めます。
   - **bind-address=source_ip_adress**
     このオプションは、レプリカからソースへの接続に必要です。
   - **server-id=id**
     id は一意である必要があります。
   - **log_bin=path_to_source_server_log**
     このオプションは、**MySQL** ソースサーバーのバイナリーログファイルへのパスを定義します。例: **log_bin=/var/log/mysql/mysql-bin.log**
   - **gtid_mode=ON**
     このオプションは、サーバー上でグローバルトランザクション識別子 (GTID) を有効にします。
   - **enforce-gtid-consistency=ON**
     サーバーは、GTID を使用して安全にログに記録できるステートメントのみの実行を許可することにより、GTID の整合性を強化します。
   - **オプション: binlog_do_db=db_name**
     選択したデータベースのみを複製する場合は、このオプションを使用します。選択した複数のデータベースを複製するには、各データベースを個別に指定します。

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

- **オプション: binlog_ignore_db=db_name**
  このオプションを使用して、特定のデータベースをレプリケーションから除外します。

2. **mysqld** サービスを再起動します。

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

3.7.2. MySQL レプリカサーバーの設定

レプリケーションを成功させるために MySQL レプリカサーバーに必要な設定オプションを設定できます。

前提条件

- レプリカサーバーがインストールされている。

手順

1. **[mysqld]** セクションの **/etc/my.cnf.d/mysql-server.cnf** ファイルに以下のオプションを含めます。

   - **server-id=id**
     id は一意である必要があります。

   - **relay-log=path_to_replica_server_log**
     リレーログは、レプリケーション中に MySQL レプリカサーバーによって作成されたログファイルのセットです。

   - **log_bin=path_to_replica_sever_log**
     このオプションは、MySQL レプリカサーバーのバイナリーログファイルへのパスを定義します。例: **log_bin=/var/log/mysql/mysql-bin.log**

     このオプションはレプリカでは必須ではありませんが、強く推奨します。

   - **gtid_mode=ON**
     このオプションは、サーバー上でグローバルトランザクション識別子 (GTID) を有効にします。

   - **enforce-gtid-consistency=ON**
     サーバーは、GTID を使用して安全にログに記録できるステートメントのみの実行を許可することにより、GTID の整合性を強化します。

   - **log-replica-updates=ON**
     このオプションにより、ソースサーバーから受信した更新がレプリカのバイナリーログに記録されます。

   - **skip-replica-start=ON**
     このオプションは、レプリカサーバーの起動時に、レプリカサーバーがレプリケーションスレッドを開始しないようにします。

   - **オプション: binlog_do_db=db_name**

特定のデータベースのみを複製する場合は、このオプションを使用します。複数のデータベースを複製するには、各データベースを個別に指定します。

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

- **オプション: binlog_ignore_db=db_name**
  このオプションを使用して、特定のデータベースをレプリケーションから除外します。

2. **mysqld** サービスを再起動します。

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

3.7.3. MySQL ソースサーバーでのレプリケーションユーザーの作成

レプリケーションユーザーを作成し、このユーザーにレプリケーショントラフィックに必要なパーミッションを付与する必要があります。この手順は、適切なパーミッションを持つレプリケーションユーザーを作成する方法を示しています。これらの手順は、ソースサーバーでのみ実行してください。

前提条件

- ソースサーバーは、MySQL ソースサーバーの設定 で説明されているように、インストールおよび設定されている。

手順

1. レプリケーションユーザーを作成します。

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

2. ユーザーにレプリケーション権限を付与します。

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

3. MySQL データベースの付与テーブルを再読み込みします。

```
mysql> FLUSH PRIVILEGES;
```

4. ソースサーバーを読み取り専用状態に設定します。

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

3.7.4. レプリカサーバーをソースサーバーに接続する

MySQL レプリカサーバーでは、認証情報とソースサーバーのアドレスを設定する必要があります。次の手順を使用して、レプリカサーバーを実装します。

前提条件

- ソースサーバーは、MySQL ソースサーバーの設定 で説明されているように、インストールおよび設定されている。
- レプリカサーバーは、MySQL レプリカサーバーの設定 で説明されているように、インストールおよび設定されている。
- レプリケーションユーザーを作成している。MySQL ソースサーバーでのレプリケーションユーザーの作成 を参照してください。

手順

1. レプリカサーバーを読み取り専用状態に設定します。

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

2. レプリケーションソースを設定します。

```
mysql> CHANGE REPLICATION SOURCE TO
    -> SOURCE_HOST='source_ip_address',
    -> SOURCE_USER='replication_user',
    -> SOURCE_PASSWORD='password',
    -> SOURCE_AUTO_POSITION=1;
```

3. MySQL レプリカサーバーでレプリカスレッドを開始します。

```
mysql> START REPLICA;
```

4. ソースサーバーとレプリカサーバーの両方で、読み取り専用状態の設定を解除します。

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

5. **オプション:** デバッグの目的で、レプリカサーバーのステータスを検査します。

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

注記

レプリカサーバーの起動または接続に失敗した場合は、**SHOW MASTER STATUS** コマンドの出力に表示されるバイナリーログファイルの位置に続く特定の数のイベントをスキップできます。たとえば、定義された位置から最初のイベントをスキップします。

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

レプリカサーバーを再起動してみてください。

6. **オプション:** レプリカサーバーでレプリカスレッドを停止します。

```
mysql> STOP REPLICA;
```

3.7.5. 検証手順

1. ソースサーバーにサンプルデータベースを作成します。

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

2. **test_db_name** データベースが、レプリカサーバーで複製されていることを確認します。

3. ソースサーバーまたはレプリカサーバーのいずれかで以下のコマンドを実行して、**MySQL** サーバーのバイナリーログファイルに関するステータス情報を表示します。

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

ソースで実行されたトランザクションの GTID のセットを示す **Executed_Gtid_Set** 列は、空であってはなりません。

注記

レプリカサーバーで **SHOW SLAVE STATUS** を使用すると、同じ GTID のセットが **Executed_Gtid_Set** 行に表示されます。

3.7.6. 関連情報

- MySQL Replication documentation
- How To Set Up Replication in MySQL
- Replication with Global Transaction Identifiers

3.8. MYSQL クライアントアプリケーションの開発

Red Hat では、**MariaDB** クライアントライブラリーに対して **MySQL** クライアントアプリケーションを開発することを推奨します。クライアントとサーバー間の通信プロトコルは、**MariaDB** と **MySQL** の間で互換性があります。**MariaDB** クライアントライブラリーは、**MySQL** 実装に固有の限られた数の機能を除き、ほとんどの一般的な **MySQL** シナリオで機能します。

MariaDB クライアントライブラリーに対してアプリケーションをビルドするために必要な開発ファイルとプログラムは、**mariadb-connector-c-devel** パッケージで提供されます。

直接ライブラリー名を使用する代わりに、**mariadb-connector-c-devel** パッケージで配布されている **mariadb_config** プログラムを使用します。このプログラムにより、正しいビルドフラグが確実に返されるようになります。

第4章 POSTGRESQL の使用

PostgreSQL サーバーは、SQL 言語をベースにした、オープンソースの堅牢かつ拡張性に優れたデータベースサーバーです。**PostgreSQL** サーバーは、オブジェクトリレーショナルデータベースシステムを提供します。これにより、広範なデータセットと多数の同時ユーザーを管理できます。このような理由から、**PostgreSQL** サーバーは、大量のデータを管理するためにクラスターで使用できます。

PostgreSQL サーバーには、データの整合性の確保、耐障害性のある環境やアプリケーションの構築を行うための機能が含まれます。**PostgreSQL** サーバーを使用すると、データベースを再コンパイルすることなく、独自のデータ型、カスタム関数、またはさまざまなプログラミング言語のコードでデータベースを拡張できます。

RHEL システムに **PostgreSQL** をインストールして設定する方法、**PostgreSQL** データをバックアップする方法、および **PostgreSQL** の以前のバージョンから移行する方法について説明します。

4.1. POSTGRESQL のインストール

RHEL 9.0 は、この Application Stream の初期バージョンとして **PostgreSQL 13** を提供します。これは、RPM パッケージとして簡単にインストールできます。

RHEL 9 のマイナーリリースで、ライフサイクルがより短い追加の **PostgreSQL** バージョンが、モジュールとして提供されます。

- RHEL 9.2 で、**PostgreSQL 15** が **postgresql:15** モジュールストリームとして導入されました。
- RHEL 9.4 で、**PostgreSQL 16** が **postgresql:16** モジュールストリームとして導入されました。

PostgreSQL をインストールするには、以下の手順に従います。

注記

設計上、同じモジュールの複数のバージョン (ストリーム) を並行してインストールすることはできません。したがって、**postgresql** モジュールから利用可能なストリームのいずれかを選択する必要があります。コンテナー内では、別々のバージョンの **PostgreSQL** データベースサーバーを使用できます。コンテナー内で複数の PostgreSQL バージョンを実行する を参照してください。

手順

1. **PostgreSQL** サーバーパッケージをインストールします。
   - **PostgreSQL 13** の場合は、RPM パッケージからインストールします。

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

   - **PostgreSQL 15** または **PostgreSQL 16** の場合は、**postgresql** モジュールからストリーム (バージョン) 15 または 16 を選択し、**server** プロファイルを指定します。以下に例を示します。

     ```
     # dnf module install postgresql:16/server
     ```

     postgres のスーパーユーザーが自動的に作成されます。

2. データベースクラスターを初期化します。

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

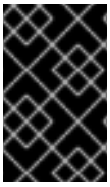
Red Hat は、デフォルトの **/var/lib/pgsql/data** ディレクトリーにデータを保存することを推奨します。

3. **postgresql** サービスを開始します。

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

4. **postgresql** サービスが、システムの起動時に起動するようにします。

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



重要

RHEL 9 内の以前の **postgresql** ストリームからアップグレードする場合は、後続のストリームへの切り替え と RHEL 9 バージョンの PostgreSQL への移行 の両方の手順に従ってください。

4.1.1. コンテナー内で複数の PostgreSQL バージョンを実行する

同じホスト上で別々のバージョンの PostgreSQL を実行するには、コンテナー内で実行してください。同じモジュールの複数のバージョン (ストリーム) を並行してインストールすることはできないためです。

この手順では、例として PostgreSQL 13 と PostgreSQL 15 を記載していますが、Red Hat Ecosystem Catalog で利用可能な任意の PostgreSQL コンテナーバージョンを使用できます。

前提条件

- **container-tools** メタパッケージがインストールされている。

手順

1. Red Hat カスタマーポータルアカウントを使用して、**registry.redhat.io** レジストリーに認証します。

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

すでにコンテナーレジストリーにログインしている場合は、このステップをスキップしてください。

2. コンテナー内で PostgreSQL 13 を実行します。

```
$ podman run -d --name <container_name> -e POSTGRESQL_USER=<user_name> -e
POSTGRESQL_PASSWORD=<password> -e
POSTGRESQL_DATABASE=<database_name> -p <host_port_1>:5432
rhel9/postgresql-13
```

このコンテナーイメージを使用する方法の詳細は、Red Hat Ecosystem Catalog を参照してください。

3. コンテナー内で PostgreSQL 15 を実行します。

```
$ podman run -d --name <container_name> -e POSTGRESQL_USER=<user_name> -e
POSTGRESQL_PASSWORD=<password> -e
POSTGRESQL_DATABASE=<database_name> -p <host_port_2>:5432
rhel9/postgresql-15
```

このコンテナーイメージを使用する方法の詳細は、Red Hat Ecosystem Catalog を参照してください。

4. コンテナー内で PostgreSQL 16 を実行します。

```
$ podman run -d --name <container_name> -e POSTGRESQL_USER=<user_name> -e
POSTGRESQL_PASSWORD=<password> -e
POSTGRESQL_DATABASE=<database_name> -p <host_port_3>:5432
rhel9/postgresql-16
```

このコンテナーイメージを使用する方法の詳細は、Red Hat Ecosystem Catalog を参照してください。

注記

2 つのデータベースサーバーのコンテナー名とホストポートが異なっている必要があります。

5. クライアントがネットワーク上のデータベースサーバーにアクセスできるように、ファイアウォールでホストポートを開きます。

```
# firewall-cmd --permanent --add-port=
{<host_port_1>/tcp,<host_port_2>/tcp,<host_port_3>/tcp,...}
# firewall-cmd --reload
```

検証手順

1. 実行中のコンテナーに関する情報を表示します。

```
$ podman ps
```

2. データベースサーバーに接続し、root としてログインします。

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

関連情報

- コンテナーの構築、実行、および管理
- Red Hat Ecosystem Catalog でコンテナーを参照する

4.2. POSTGRESQL ユーザーの作成

PostgreSQL ユーザーは以下のタイプのものです。

- **postgres** UNIX システムユーザー: PostgreSQL サーバーおよびクライアントアプリケーション (**pg_dump** など) を実行する場合にのみ使用してください。データベース作成およびユーザー管理などの、PostgreSQL 管理上の対話的な作業には、**postgres** システムユーザーを使用しないでください。

- データベースのスーパーユーザー: デフォルトの **postgres** PostgreSQL スーパーユーザーは、**postgres** システムユーザーとは関係ありません。**pg_hba.conf** ファイルの **postgres** のスーパーユーザーのアクセスを制限することができます。制限しない場合には、その他のパーミッションの制限はありません。他のデータベースのスーパーユーザーを作成することもできます。

- 特定のデータベースアクセスパーミッションを持つロール:

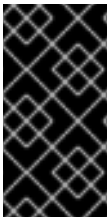
  - データベースユーザー: デフォルトでログインするパーミッションがある。

  - ユーザーのグループ: グループ全体のパーミッションを管理できるようにします。

ロールはデータベースオブジェクト (テーブルや関数など) を所有することができ、SQL コマンドを使用してオブジェクト権限を他のロールに割り当てることができます。

標準のデータベース管理権限には
SELECT、**INSERT**、**UPDATE**、**DELETE**、**TRUNCATE**、**REFERENCES**、**TRIGGER**、**CREATE**、**CONNECT**、**TEMPORARY**、**EXECUTE**、および **USAGE** が含まれます。

ロール属性は、**LOGIN**、**SUPERUSER**、**CREATEDB**、および **CREATEROLE** などの特別な権限です。



重要

Red Hat は、スーパーユーザーではないロールとしてほとんどのタスクを実行することを推奨します。一般的な方法として、**CREATEDB** および **CREATEROLE** の権限を持つロールを作成し、このロールをデータベースおよびロールのすべてのルーチン管理に使用します。

前提条件

- PostgreSQL サーバーがインストールされている
- データベースクラスターが初期化されている

手順

- ユーザーを作成するには、ユーザーのパスワードを設定し、ユーザーに **CREATEROLE** および **CREATEDB** の権限を割り当てます。

  ```
  postgres=# CREATE USER mydbuser WITH PASSWORD 'mypasswd' CREATEROLE CREATEDB;
  ```

  mydbuser をユーザー名に、**mypasswd** をユーザーのパスワードに置き換えます。

関連情報

- PostgreSQL データベースのロール
- PostgreSQL 権限

- PostgreSQL の設定

例4.1 PostgreSQL データベースの初期化、作成、接続

この例では、PostgreSQL データベースを初期化方法、日常的なデータベース管理権限を持つデータベースユーザーの作成方法、および管理権限を持つデータベースユーザーを介して任意のシステムアカウントからアクセスできるデータベースの作成方法を示します。

1. PosgreSQL サーバーをインストールします。

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

2. データベースクラスターを初期化します。

```
# postgresql-setup --initdb
* Initializing database in '/var/lib/pgsql/data'
* Initialized, logs are in /var/lib/pgsql/initdb_postgresql.log
```

3. パスワードハッシュアルゴリズムを **scram-sha-256** に設定します。

   a. **/var/lib/pgsql/data/postgresql.conf** ファイルで、次の行を変更します。

   ```
   #password_encryption = md5              # md5 or scram-sha-256
   ```

   更新後は次のようになります。

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

   b. **/var/lib/pgsql/data/pg_hba.conf** ファイルで、IPv4 ローカル接続用に次の行を変更します。

   ```
   host    all             all             127.0.0.1/32            ident
   ```

   更新後は次のようになります。

   ```
   host    all             all             127.0.0.1/32            scram-sha-256
   ```

4. postgresql サービスを起動します。

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

5. **postgres** という名前のシステムユーザーとしてログインします。

```
# su - postgres
```

6. **PostgreSQL** インタラクティブターミナルを起動します。

```
$ psql
psql (13.7)
Type "help" for help.

postgres=#
```

7. オプション: 現在のデータベース接続に関する情報を取得します。

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

8. **mydbuser** という名前のユーザーを作成し、**mydbuser** のパスワードを設定して、**CREATEROLE** および **CREATEDB** の権限を **mydbuser** に割り当てます。

```
postgres=# CREATE USER mydbuser WITH PASSWORD 'mypasswd' CREATEROLE
CREATEDB;
CREATE ROLE
```

これで、**mydbuser** ユーザーは、日常的なデータベース管理操作 (データベースの作成とユーザーインデックスの管理) を実行できるようになりました。

9. **\q** メタコマンドを使用して、インタラクティブターミナルからログアウトします。

```
postgres=# \q
```

10. **postgres** ユーザーセッションからログアウトします。

```
$ logout
```

11. **mydbuser** として PostgreSQL ターミナルにログインし、ホスト名を指定して、初期化中に作成されたデフォルトの **postgres** データベースに接続します。

```
# psql -U mydbuser -h 127.0.0.1 -d postgres
Password for user mydbuser:
Type the password.
psql (13.7)
Type "help" for help.

postgres=>
```

12. **mydatabase** という名前のデータベースを作成します。

```
postgres=> CREATE DATABASE mydatabase;
CREATE DATABASE
postgres=>
```

13. セッションからログアウトします。

```
postgres=# \q
```

14. **mydbuser** として mydatabase に接続します。

```
# psql -U mydbuser -h 127.0.0.1 -d mydatabase
Password for user mydbuser:
psql (13.7)
Type "help" for help.
mydatabase=>
```

15. オプション: 現在のデータベース接続に関する情報を取得します。

```
mydatabase=> \conninfo
You are connected to database "mydatabase" as user "mydbuser" on host "127.0.0.1" at port "5432".
```

4.3. POSTGRESQL の設定

PostgreSQL データベースでは、データおよび設定ファイルはすべて、データベースクラスターと呼ばれる 1 つのディレクトリーに保存されます。Red Hat は、設定ファイルを含むすべてのデータをデフォルトの **/var/lib/pgsql/data/** ディレクトリーに保存することを推奨しています。

PostgreSQL 設定は、以下のファイルで設定されます。

- **postgresql.conf**: データベースのクラスターパラメーターの設定に使用されます。
- **postgresql.auto.conf**: **postgresql.conf** と同様の基本的な PostgreSQL 設定を保持します。ただし、このファイルはサーバーの制御下にあります。これは、**ALTER SYSTEM** クエリーにより編集され、手動で編集することはできません。
- **pg_ident.conf**: 外部認証メカニズムから PostgreSQL ユーザー ID へのユーザー ID のマッピングに使用されます。
- **pg_hba.conf**: PostgreSQL データベースのクライアント認証の設定に使用されます。

PostgreSQL 設定を変更するには、以下の手順に従います。

手順

1. 各設定ファイル (例: **/var/lib/pgsql/data/postgresql.conf**) を編集します。
2. **postgresql** サービスを再起動して、変更を有効にします。

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

例4.2 PostgreSQL データベースクラスターパラメーターの設定

以下の例では、**/var/lib/pgsql/data/postgresql.conf** ファイルのデータベースクラスターパラメーターの基本設定を示しています。

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

例4.3 PostgreSQL でのクライアント認証の設定

以下の例では、**/var/lib/pgsql/data/pg_hba.conf** ファイルでクライアント認証を設定する方法を説明します。

```
# 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
```

4.4. POSTGRESQL サーバーにおける TLS 暗号化の設定

デフォルトでは、**PostgreSQL** は暗号化されていない接続を使用します。よりセキュアな接続のために、**PostgreSQL** サーバーで Transport Layer Security (TLS) サポートを有効にし、暗号化された接続を確立するようにクライアントを設定できます。

前提条件

- **PostgreSQL** サーバーがインストールされている
- データベースクラスターが初期化されている
- サーバーが RHEL 9.2 以降を実行し、FIPS モードが有効になっている場合、クライアントが Extended Master Secret (EMS) 拡張機能をサポートしているか、TLS 1.3 を使用している必要があります。EMS を使用しない TLS 1.2 接続は失敗します。詳細は、ナレッジベースの記事 TLS extension "Extended Master Secret" enforced を参照してください。

手順

1. OpenSSL ライブラリーをインストールします。

```
# dnf install openssl
```

2. TLS 証明書とキーを生成します。

```
# openssl req -new -x509 -days 365 -nodes -text -out server.crt \
  -keyout server.key -subj "/CN=dbhost.yourdomain.com"
```

dbhost.yourdomain.com を データベースのホストとドメイン名に置き換えます。

3. 署名済み証明書と秘密鍵をデータベースサーバー上の必要なロケーションにコピーします。

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

4. 署名付き証明書と秘密鍵の所有者とグループの所有権を **postgres** ユーザーに変更します。

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

5. 所有者だけが読み取れるように、秘密鍵の権限を制限します。

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

6. **/var/lib/pgsql/data/postgresql.conf** ファイルの次の行を変更して、パスワードハッシュアルゴリズムを **scram-sha-256** に設定します。

```
#password_encryption = md5             # md5 or scram-sha-256
```

更新後は次のようになります。

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

7. **/var/lib/pgsql/data/postgresql.conf** ファイルの次の行を変更して、SSL/TLS を使用するように PostgreSQL を設定します。

```
#ssl = off
```

更新後は次のようになります。

```
ssl=on
```

8. **/var/lib/pgsql/data/pg_hba.conf** ファイルの IPv4 ローカル接続で次の行を変更して、TLS を使用するクライアントからの接続のみを受け入れるように、すべてのデータベースへのアクセスを制限します。

```
host  all  all  127.0.0.1/32  ident
```

更新後は次のようになります。

```
hostssl  all  all  127.0.0.1/32  scram-sha-256
```

または、次の行を新たに追加して、単一のデータベースとユーザーのアクセスを制限できます。

```
hostssl mydatabase mydbuser 127.0.0.1/32  scram-sha-256
```

mydatabase をデータベース名に、**mydbuser** をユーザー名に置き換えます。

9. **postgresql** サービスを再起動して、変更を有効にします。

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

検証

- 接続が暗号化されていることを手動で確認するには、以下を行います。

1. **mydbuser** ユーザーとして **PostgreSQL** データベースに接続し、ホスト名とデータベース名を指定します。

```
$ psql -U mydbuser -h 127.0.0.1 -d mydatabase
Password for user mydbuser:
```

mydatabase をデータベース名に、**mydbuser** をユーザー名に置き換えます。

2. 現在のデータベース接続に関する情報を取得します。

```
mydbuser=> \conninfo
You are connected to database "mydatabase" as user "mydbuser" on host "127.0.0.1" at port "5432".
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
```

–

- PostgreSQL への接続が暗号化されているかどうかを検証する簡単なアプリケーションを作成できます。この例は、**libpq-devel** パッケージで提供される **libpq** クライアントライブラリーを使用する C で記述されたアプリケーションを示しています。

```
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>

int main(int argc, char* argv[])
{
//Create connection
PGconn* connection = PQconnectdb("hostaddr=127.0.0.1 password=mypassword
port=5432 dbname=mydatabase user=mydbuser");

if (PQstatus(connection) ==CONNECTION_BAD)
    {
    printf("Connection error\n");
    PQfinish(connection);
    return -1; //Execution of the program will stop here
    }
    printf("Connection ok\n");
    //Verify TLS
    if (PQsslInUse(connection)){
     printf("TLS in use\n");
     printf("%s\n", PQsslAttribute(connection,"protocol"));
    }
    //End connection
    PQfinish(connection);
    printf("Disconnected\n");
    return 0;
}
```

mypassword をパスワードに、mydatabase をデータベース名に、myduser をユーザー名に置き換えます。

注記

-lpq オプションを使用して、コンパイルのために **pq** ライブラリーをロードする必要があります。たとえば、GCC コンパイラーを使用してアプリケーションをコンパイルするには、次のようにします。

```
$ gcc source_file.c -lpq -o myapplication
```

この source_file.c には上記のサンプルコードが含まれており、myapplication はセキュアな PostgreSQL 接続を検証するためのアプリケーションの名前です。

例4.4 TLS 暗号化を使用した PostgreSQL データベースの初期化、作成、接続

この例では、PostgreSQL データベースの初期化方法、データベースユーザーとデータベースの作成方法、セキュアな接続を使用したデータベースへの接続方法を示します。

1. PosgreSQL サーバーをインストールします。

■

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

2. データベースクラスターを初期化します。

```
# postgresql-setup --initdb
* Initializing database in '/var/lib/pgsql/data'
* Initialized, logs are in /var/lib/pgsql/initdb_postgresql.log
```

3. OpenSSL ライブラリーをインストールします。

```
# dnf install openssl
```

4. TLS 証明書とキーを生成します。

```
# openssl req -new -x509 -days 365 -nodes -text -out server.crt \
  -keyout server.key -subj "/CN=dbhost.yourdomain.com"
```

dbhost.yourdomain.com を データベースのホストとドメイン名に置き換えます。

5. 署名済み証明書と秘密鍵をデータベースサーバー上の必要なロケーションにコピーします。

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

6. 署名付き証明書と秘密鍵の所有者とグループの所有権を **postgres** ユーザーに変更します。

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

7. 所有者だけが読み取れるように、秘密鍵の権限を制限します。

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

8. パスワードハッシュアルゴリズムを **scram-sha-256** に設定します。**/var/lib/pgsql/data/postgresql.conf** ファイルで、次の行を変更します。

```
#password_encryption = md5              # md5 or scram-sha-256
```

更新後は次のようになります。

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

9. SSL/TLS を使用するように PostgreSQL を設定します。**/var/lib/pgsql/data/postgresql.conf** ファイルで、次の行を変更します。

```
#ssl = off
```

更新後は次のようになります。

```
ssl=on
```

10. **postgresql** サービスを開始します。

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

11. **postgres** という名前のシステムユーザーとしてログインします。

```
# su - postgres
```

12. **postgres** ユーザーとして PostgreSQL インタラクティブターミナルを起動します。

```
$ psql -U postgres
psql (13.7)
Type "help" for help.

postgres=#
```

13. **mydbuser** という名前のユーザーを作成し、**mydbuser** のパスワードを設定します。

```
postgres=# CREATE USER mydbuser WITH PASSWORD 'mypasswd';
CREATE ROLE
postgres=#
```

14. **mydatabase** という名前のデータベースを作成します。

```
postgres=# CREATE DATABASE mydatabase;
CREATE DATABASE
postgres=#
```

15. すべての権限を **mydbuser** ユーザーに付与します。

```
postgres=# GRANT ALL PRIVILEGES ON DATABASE mydatabase TO mydbuser;
GRANT
postgres=#
```

16. インタラクティブターミナルからログアウトします。

```
postgres=# \q
```

17. **postgres** ユーザーセッションからログアウトします。

```
$ logout
```

18. **/var/lib/pgsql/data/pg_hba.conf** ファイルの IPv4 ローカル接続で次の行を変更して、TLS を使用するクライアントからの接続のみを受け入れるように、すべてのデータベースへのアクセスを制限します。

```
host  all  all  127.0.0.1/32  ident
```

更新後は次のようになります。

```
hostssl  all  all  127.0.0.1/32  scram-sha-256
```

19. **postgresql** サービスを再起動して、変更を有効にします。

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

20. **mydbuser** ユーザーとして **PostgreSQL** データベースに接続し、ホスト名とデータベース名を指定します。

```
$ psql -U mydbuser -h 127.0.0.1 -d mydatabase
Password for user mydbuser:
psql (13.7)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

mydatabase=>
```

4.5. POSTGRESQL データのバックアップ

PostgreSQL データをバックアップするには、以下のいずれかの方法を使用します。

SQL ダンプ

Backing up with SQL dump を参照してください。

ファイルシステムレベルのバックアップ

File system level backup を参照してください。

継続的アーカイブ

継続的アーカイブ を参照してください。

4.5.1. SQL ダンプを使用した PostgreSQL データのバックアップ

SQL ダンプのメソッドは、SQL コマンドを使用したダンプファイルの生成に基づいています。ダンプがデータベースサーバーにアップロードされると、ダンプ時と同じ状態でデータベースが再作成されます。

SQL ダンプは、以下の **PostgreSQL** クライアントアプリケーションによって保証されます。

- **pg_dump** は、ロールまたはテーブル空間に関するクラスター全体の情報なしに単一のデータベースをダンプします。
- **pg_dumpall** は、指定のクラスターに各データベースをダンプし、ロールやテーブル空間定義などのクラスター全体のデータを保持します。

デフォルトでは、**pg_dump** コマンドおよび **pg_dumpall** コマンドは、結果を標準出力に書き込みます。ダンプをファイルに保存するには、出力を SQL ファイルにリダイレクトします。作成される SQL ファイルは、テキスト形式またはその他の形式のいずれかになります。これにより並列処理が可能になり、オブジェクトの復元をより詳細に制御できます。

データベースにアクセスできる任意のリモートホストから、SQL ダンプを実行できます。

4.5.1.1. SQL ダンプの長所と短所

SQL ダンプには、他の **PostgreSQL** バックアップ方法と比較して、以下の長所があります。

- SQL ダンプは、サーバーのバージョン固有ではない唯一の **PostgreSQL** バックアップメソッドです。**pg_dump** ユーティリティーの出力は、**PostgreSQL** の後続のバージョンに再読み込みできます。これは、ファイルシステムレベルのバックアップ、または継続的なアーカイブにはできません。

- SQL ダンプは、32 ビットサーバーから 64 ビットサーバーへなど、異なるアーキテクチャーにデータベースを転送する際に有効な唯一の方法です。
- SQL ダンプは、内部的に一貫性のあるダンプを提供します。ダンプは、**pg_dump** の実行開始時のデータベースのスナップショットを表します。
- **pg_dump** ユーティリティーは、実行中のデータベースの他の操作をブロックしません。

SQL ダンプの短所は、ファイルシステムレベルのバックアップと比較して時間がかかることです。

4.5.1.2. pg_dump を使用した SQL ダンプの実行

クラスター全体の情報なしに単一のデータベースをダンプするには、**pg_dump** ユーティリティーを使用します。

前提条件

- ダンプするすべてのテーブルへの読み取りアクセスが必要です。データベース全体をダンプするには、**postgres** のスーパーユーザーまたはデータベースの管理者権限を持つユーザーとして、コマンドを実行する必要があります。

手順

- クラスター全体の情報なしでデータベースをダンプします。

```
$ pg_dump dbname > dumpfile
```

pg_dump が接続するデータベースサーバーを指定するには、以下のコマンドラインオプションを使用します。

- ホストを定義する **-h** オプション
  デフォルトのホストは、ローカルホストか、**PGHOST** 環境変数で指定されているものです。
- ポートを定義する **-p** オプション
  デフォルトのポートは、**PGPORT** 環境変数またはコンパイル済みデフォルトで示されます。

4.5.1.3. pg_dumpall を使用した SQL ダンプの実行

特定のデータベースクラスターで各データベースをダンプし、クラスター全体のデータを保持するには、**pg_dumpall** ユーティリティーを使用します。

前提条件

- **postgres** スーパーユーザーまたはデータベースの管理者権限を持つユーザーとして、コマンドを実行する必要があります。

手順

- データベースクラスターのすべてのデータベースをダンプし、クラスター全体のデータを保存します。

```
$ pg_dumpall > dumpfile
```

pg_dumpall が接続するデータベースサーバーを指定するには、以下のコマンドラインオプションを使用します。

- ホストを定義する **-h** オプション
  デフォルトのホストは、ローカルホストか、**PGHOST** 環境変数で指定されているものです。
- ポートを定義する **-p** オプション
  デフォルトのポートは、**PGPORT** 環境変数またはコンパイル済みデフォルトで示されます。
- デフォルトのデータベースを定義する **-l** オプション
  このオプションにより、初期化時に自動的に作成された **postgres** データベースとは異なるデフォルトのデータベースを選択できます。

4.5.1.4. pg_dump を使用したダンプされたデータベースの復元

pg_dump ユーティリティーを使用してダンプした SQL ダンプからデータベースを復元するには、以下の手順に従います。

前提条件

- **postgres** スーパーユーザーまたはデータベースの管理者権限を持つユーザーとして、コマンドを実行する必要があります。

手順

1. 新しいデータベースを作成します。

   ```
   $ createdb dbname
   ```

2. ダンプされたデータベースのオブジェクトを所有するか、オブジェクトに対する権限が許可されたユーザーがすべて存在していることを検証してください。このようなユーザーが存在しない場合、復元は元の所有権と権限でオブジェクトの再作成に失敗します。

3. **psql** ユーティリティーを実行して、**pg_dump** ユーティリティーが作成したテキストファイルのダンプを復元します。

   ```
   $ psql dbname < dumpfile
   ```

   ここでの **dumpfile** は、**pg_dump** コマンドの出力になります。非テキストファイルのダンプを復元するには、代わりに **pg_restore** ユーティリティーを使用します。

   ```
   $ pg_restore non-plain-text-file
   ```

4.5.1.5. pg_dumpall を使用したダンプされたデータベースの復元

pg_dumpall ユーティリティーを使用してダンプしたデータベースクラスターからデータを復元するには、以下の手順に従います。

前提条件

- **postgres** スーパーユーザーまたはデータベースの管理者権限を持つユーザーとして、コマンドを実行する必要があります。

手順

1. ダンプされたデータベースのオブジェクトを所有するか、オブジェクトに対する権限が許可されたユーザーがすべて、すでに存在していることを検証してください。このようなユーザーが存在しない場合、復元は元の所有権と権限でオブジェクトの再作成に失敗します。

2. **psql** ユーティリティーを実行して、**pg_dumpall** ユーティリティーにより作成されたテキストファイルのダンプを復元します。

```
$ psql < dumpfile
```

ここでの **dumpfile** は、**pg_dumpall** コマンドの出力になります。

4.5.1.6. 別のサーバーでのデータベースの SQL ダンプの実行

pg_dump および psql はパイプに対する読み書きが可能であるため、あるサーバーから別のサーバーにデータベースを直接ダンプできます。

手順

- データベースを、サーバーから別のサーバーにダンプするには、以下のコマンドを実行します。

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

4.5.1.7. 復元中の SQL エラーの処理

デフォルトでは、SQL エラーが発生した場合、**psql** は実行を継続します。これにより、データベースの復元は一部のみとなります。

デフォルトの動作を変更するには、ダンプを復元する際に以下のいずれかの方法を使用します。

前提条件

- **postgres** スーパーユーザーまたはデータベースの管理者権限を持つユーザーとして、コマンドを実行する必要があります。

手順

- **ON_ERROR_STOP** 変数を設定して SQL エラーが発生した場合は、終了ステータスが 3 で **psql** を終了します。

```
$ psql --set ON_ERROR_STOP=on dbname < dumpfile
```

- ダンプ全体が単一のトランザクションとして復元されるように指定して、復元が完全に完了するかキャンセルされるようにします。
  - **psql** ユーティリティーを使用してテキストファイルのダンプを復元する場合:

```
$ psql -1
```

  - **pg_restore** ユーティリティーを使用してテキストファイル以外のダンプを復元する場合:

```
$ pg_restore -e
```

-

この方法を使用する場合は、多少のエラーでも、すでに何時間も実行している復元操作をキャンセルできます。

4.5.1.8. 関連情報

- PostgreSQL Documentation - SQL dump

4.5.2. ファイルシステムレベルのバックアップを使用した PostgreSQL データのバックアップ

ファイルシステムレベルのバックアップを実行するには、**PostgreSQL** データベースファイルを別の場所に作成します。たとえば、以下のいずれかの方法を使用できます。

- **tar** ユーティリティーを使用してアーカイブファイルを作成します。
- **rsync** ユーティリティーを使用して、ファイルを別の場所にコピーします。
- データディレクトリーの一貫したスナップショットを作成します。

4.5.2.1. ファイルシステムのバックアップを作成する利点と制限

ファイルシステムレベルのバックアップは、他の **PostgreSQL** バックアップ方法と比較して、以下の長所があります。

- 通常、ファイルシステムレベルのバックアップは、SQL ダンプよりも高速です。

ファイルシステムレベルのバックアップは、他の **PostgreSQL** バックアップ方法と比較して、以下の制限があります。

- このバックアップメソッドは、RHEL 8 から RHEL 9 にアップグレードし、アップグレードしたシステムにデータを移行する場合には適していません。ファイルシステムレベルのバックアップは、アーキテクチャーと RHEL メジャーバージョンに固有のものです。アップグレードに成功しなかった場合は、RHEL 8 システムでデータを復元できますが、RHEL 9 システムではデータを復元できません。
- データをバックアップおよび復元する前に、データベースサーバーをシャットダウンする必要があります。
- 特定のファイルまたはテーブルを個々にバックアップまたは復元することはできません。ファイルシステムのバックアップは、データベースクラスター全体を完全にバックアップおよび復元する場合にのみ機能します。

4.5.2.2. ファイルシステムレベルのバックアップの実行

ファイルシステムレベルのバックアップを実行するには、次の手順を使用します。

手順

1. データベースクラスターの場所を選択し、このクラスターを初期化します。

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

2. postgresql サービスを停止します。

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

3. 任意のメソッドを使用してファイルシステムのバックアップを作成します (例: **tar** アーカイブ)。

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

4. postgresql サービスを起動します。

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

関連情報

- PostgreSQL Documentation - file system level backup

4.5.3. 継続的にアーカイブして PostgreSQL データのバックアップを作成

4.5.3.1. 継続的なアーカイブの概要

PostgreSQL は、データベースのデータファイルに対するすべての変更を、クラスターのデータディレクトリーの **pg_wal/** サブディレクトリーで利用可能なログ先行書き込み (WAL) ファイルに記録します。このログは、主にクラッシュからの復元を目的としています。クラッシュ後、最後のチェックポイント以降に作成されたログエントリーを使用して、データベースの整合性まで復元できます。

オンラインバックアップとも呼ばれる継続的なアーカイブメソッドは、WAL ファイルを、稼働中のサーバーまたはファイルシステムレベルのバックアップで実行されるベースバックアップの形式でデータベースクラスターのコピーと組み合わせます。

データベース復元が必要な場合は、データベースクラスターのコピーからデータベースを復元してから、バックアップを作成した WAL ファイルからログを再生して、システムを現在の状態にすることができます。

継続的なアーカイブメソッドでは、少なくとも最後のベースバックアップの開始時間までさかのぼって、アーカイブされたすべての WAL ファイルの連続したシーケンスを保持する必要があります。そのため、基本バックアップの理想的な頻度は、次の条件により異なります。

- アーカイブされた WAL ファイルで利用可能なストレージボリューム。
- 復元が必要な場合の、データ復元の最大許容期間。最後のバックアップからの期間が長い場合、システムはより多くの WAL セグメントを再生するため、回復に時間がかかります。

注記

pg_dump および **pg_dumpall** SQL ダンプは、継続的にアーカイブするバックアップソリューションの一部として使用することができません。SQL ダンプは論理バックアップを生成しますが、WAL 再生で使用する上で十分な情報は含まれていません。

継続的なアーカイブメソッドを使用してデータベースのバックアップと復元を実行するには、以下の手順に従います。

1. WAL ファイルのアーカイブ手順をセットアップおよびテストします。WAL アーカイブ を参照してください。

2. ベースバックアップを実行します。ベースバックアップ を参照してください。

データを復元するには、連続アーカイブを使用したデータベースの復元 の手順に従います。

4.5.3.2. 継続的なアーカイブの長所と短所

継続的なアーカイブは、**PostgreSQL** のその他のバックアップ方法と比較して、以下の利点があります。

- 継続的なバックアップメソッドでは、バックアップ内の内部不整合がログ再生により修正されるため、整合性が完全に取れないベースバックアップを使用することができます。したがって、実行中の **PostgreSQL** サーバーでベースバックアップを実行できます。
- ファイルシステムのスナップショットは必要ありません。**tar** または同様のアーカイブユーティリティーで十分です。
- 継続的にバックアップを行うには、継続的に WAL ファイルをアーカイブします。これは、ログ再生用の WAL ファイルの順序が無限に長くなる可能性があるためです。これは、特に大規模なデータベースで有用です。
- 継続的バックアップは、特定の時点への復旧 (ポイントインタイムリカバリー) をサポートします。WAL エントリーを最後まで再生する必要はありません。再生はいつでも停止でき、ベースバックアップを作成してから、データベースをいつでもその状態に復元できます。
- 一連の WAL ファイルが同じベースのバックアップファイルで読み込まれた別のマシンが継続的に利用可能である場合は、任意の時点で、データベースで現在に一番近いコピーで、他のマシンを復元できます。

継続的なアーカイブには、その他の **PostgreSQL** バックアップ方法と比較して、以下の短所があります。

- 継続バックアップ方法は、サブセットではなく、データベースクラスター全体の復元のみをサポートします。
- 継続的にバックアップするには、大きなアーカイブストレージが必要です。

4.5.3.3. WAL アーカイブの設定

稼働中の **PostgreSQL** サーバーでは、ログ先行書き込み (WAL) レコードのシーケンスを生成します。サーバーは、このシーケンスを WAL セグメントファイルに物理的に分割します。このファイルには、WAL シーケンスの位置を反映する数値名が与えられます。WAL のアーカイブを使用しない場合は、セグメントファイルは再利用され、より大きなセグメント番号に名前が変更されます。

WAL データをアーカイブする場合、各セグメントファイルの内容がキャプチャーされ、新しい場所に保存されてから、セグメントファイルが再利用されます。別のマシン上の NFS マウントディレクトリー、テープドライブ、または CD など、コンテンツの保存場所には複数のオプションがあります。

WAL レコードには、設定ファイルへの変更が含まれていないことに注意してください。

WAL のアーカイブを有効にするには、以下の手順に従います。

手順

1. **/var/lib/pgsql/data/postgresql.conf** ファイルで以下を行います。
    a. **wal_level** 設定パラメーターを **replica** 以降に設定します。

b. **archive_mode** パラメーターを **on** に設定します。

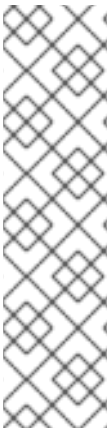
c. **archive_command** 設定パラメーターでシェルコマンドを指定します。**cp** コマンド、別のコマンド、またはシェルスクリプトを使用できます。

2. **postgresql** サービスを再起動して、変更を適用します。

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

3. アーカイブコマンドをテストし、既存のファイルが上書きされないこと、失敗した場合にゼロ以外の終了ステータスが返されることを確認します。

4. データを保護するには、セグメントファイルがグループまたはワールド読み取りアクセスを持たないディレクトリーにアーカイブされていることを確認してください。



注記

archive コマンドは、完了した WAL セグメントでのみ実行されます。WAL トラフィックをほとんど生成しないサーバーでは、トランザクションの完了とアーカイブストレージへの安全な記録の間にかなりの遅延が生じる可能性があります。アーカイブされていないデータの古さを制限するには、以下を行います。

- **archive_timeout** パラメーターを設定して、サーバーが特定の頻度で新しい WAL セグメントファイルに切り替えるように強制します。
- **pg_switch_wal** パラメーターを使用して、セグメント切り替えを強制し、トランザクションが終了後すぐにアーカイブされるようにします。

例4.5 WAL セグメントをアーカイブするためのシェルコマンド

この例は、**archive_command** 設定パラメーターに設定できる簡単なシェルコマンドを示しています。

以下のコマンドは、完了したセグメントファイルを必要な場所にコピーします。

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

%p パラメーターは、アーカイブするファイルの相対パスに置き換えられ、**%f** パラメーターはファイル名に置き換えられます。

このコマンドは、アーカイブ可能な WAL セグメントを **/mnt/server/archivedir/** ディレクトリーにコピーします。**%p** パラメーターおよび **%f** パラメーターを置き換えると、実行されたコマンドは以下のようになります。

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

アーカイブされる新規ファイルごとに同様のコマンドが生成されます。

関連情報

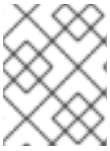
- PostgreSQL 16 ドキュメント

4.5.3.4. ベースバックアップの作成

ベースバックアップは、複数の方法で作成できます。ベースバックアップを実行する最も簡単な方法は、実行中の **PostgreSQL** サーバーで **pg_basebackup** ユーティリティーを使用することです。

ベースバックアッププロセスは、WAL アーカイブ領域に保存され、ベースバックアップに必要な最初の WAL セグメントファイルにちなんで名付けられたバックアップ履歴ファイルを作成します。

バックアップ履歴ファイルは、開始時間および終了時間、およびバックアップの WAL セグメントが含まれる小さなテキストファイルです。ラベル文字列を使用して関連するダンプファイルを特定した場合は、バックアップ履歴ファイルを使用して復元するダンプファイルを判断できます。



注記

データを確実に復元できるようにするために、複数のバックアップセットを維持することを検討してください。

ベースバックアップを実行するには、以下の手順を行います。

前提条件

- **postgres** スーパーユーザー、データベースの管理者権限のあるユーザー、または少なくとも **REPLICATION** パーミッションを持つ別のユーザーとして、コマンドを実行する必要があります。
- ベースバックアップ中およびベースバックアップ後に生成されたすべての WAL セグメントファイルを保持する必要があります。

手順

1. **pg_basebackup** ユーティリティーを使用してベースバックアップを実行します。
   - 個々のファイル (プレーン形式) としてベースバックアップを作成するには、以下を実行します。

     ```
     $ pg_basebackup -D backup_directory -Fp
     ```

     backup_directory は、任意のバックアップの場所に置き換えます。

     テーブル空間を使用し、サーバーと同じホストでベースバックアップを実行する場合は、**--tablespace-mapping** オプションも使用する必要があります。そうしないと、バックアップを同じ場所に書き込もうとすると、バックアップが失敗します。

   - **tar** アーカイブ (**tar** および圧縮形式) としてベースバックアップを作成するには、以下を実行します。

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

     backup_directory は、任意のバックアップの場所に置き換えます。

     このようなデータを復元するには、ファイルを正しい場所に手動で抽出する必要があります。

2. ベースバックアッププロセスが完了すると、バックアップ履歴ファイルで指定されている、データベースクラスターのコピーとバックアップ中に使用された WAL セグメントファイルを安全にアーカイブします。

3. ベースバックアップで使用されている WAL セグメントファイルよりも数値が小さい WAL セグメントを削除します。これらはベースバックアップよりも古く、復元には必要ないためです。

pg_basebackup が接続するデータベースサーバーを指定するには、以下のコマンドラインオプションを使用します。

- ホストを定義する **-h** オプション
  デフォルトのホストは、ローカルホストまたは **PGHOST** 環境変数により指定されたホストです。

- ポートを定義する **-p** オプション
  デフォルトのポートは、**PGPORT** 環境変数またはコンパイル済みデフォルトで示されます。

関連情報

- PostgreSQL ドキュメント - ベースバックアップ
- PostgreSQL ドキュメント - pg_basebackup ユーティリティー

4.5.3.5. 継続的なアーカイブバックアップを使用したデータベースの復元

継続バックアップを使用してデータベースを復元するには、以下の手順を行います。

手順

1. サーバーを停止します。

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

2. 必要なデータを一時的な場所にコピーします。
   必要に応じて、クラスターデータのディレクトリー全体と、すべてのテーブル空間をコピーします。既存データベースのコピーを 2 つ保持するには、システムに十分な空き領域が必要になることに注意してください。

   十分な容量がない場合は、クラスターの **pg_wal** ディレクトリーの内容を保存します。これには、システムがダウンする前にアーカイブされなかったログが含まれます。

3. クラスターデータディレクトリー、および使用しているテーブル空間のルートディレクトリー下の既存ファイルおよびサブディレクトリーをすべて削除します。

4. ベースバックアップからデータベースファイルを復元します。
   以下の点を確認してください。

   - ファイルは、正しい所有権 (**root** ではなくデータベースシステムのユーザー) で復元されます。

   - ファイルは、正しい権限で復元されます。

   - **pg_tblspc/** サブディレクトリーのシンボリックリンクが正しく復元されます。

5. **pg_wal/** サブディレクトリーにあるファイルをすべて削除します。

このファイルは、ベースバックアップから作成されるため、非推奨になりました。**pg_wal/** をアーカイブしていない場合は、適切な権限で再作成します。

6. 手順 2 で保存したアーカイブされていない WAL セグメントファイルを **pg_wal/** にコピーします。

7. クラスターデータディレクトリーの **recovery.conf** リカバリーコマンドファイルを作成し、**restore_command** 設定パラメーターにシェルコマンドを指定します。**cp** コマンド、別のコマンド、またはシェルスクリプトを使用できます。以下に例を示します。

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

8. サーバーを起動します。

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

サーバーは復元モードに入り、引き続き必要なアーカイブファイル (WAL) を読み込みます。

外部エラーにより復元が終了した場合は、サーバーを再起動して復元を続行します。復元プロセスが完了すると、サーバーは **recovery.conf** の名前を **recovery. done** に変更します。これにより、サーバーが通常のデータベース操作を開始した後に、誤ってリカバリーモードに戻るのを防ぐことができます。

9. データベースのコンテンツを確認して、データベースが必要な状態に復元されたことを検証します。
データベースが必要な状態に復元されていない場合は、手順 1 に戻ります。データベースが必要な状態に復元された場合は、**pg_hba.conf** ファイルでクライアント認証設定を復元して接続できるようにします。

継続バックアップを使用した復元の詳細は、PostgreSQL ドキュメント を参照してください。

4.5.3.6. 関連情報

- 継続的アーカイブ方法

4.6. RHEL 9 バージョンの POSTGRESQL への移行

Red Hat Enterprise Linux 8 は、複数のモジュールストリームで **PostgreSQL** を提供します (**PostgreSQL 10** (デフォルトの postgresql ストリーム)、**PostgreSQL 9.6**、**PostgreSQL 12**、**PostgreSQL 13**、**PostgreSQL 15**、および **PostgreSQL 16**)。

RHEL 9 では、**PostgreSQL 13**、**PostgreSQL 15**、および **PostgreSQL 16** を利用できます。

RHEL では、データベースファイルに 2 つの **PostgreSQL** 移行パスを使用できます。

- pg_upgrade ユーティリティーを使用した高速アップグレード
- ダンプおよび復元のアップグレード

高速アップグレードメソッドは、ダンプおよび復元のプロセスよりも速くなります。ただし、場合によっては高速アップグレードが機能せず、たとえばアーキテクチャー間のアップグレードなど、ダンプおよび復元プロセスのみを使用できます。

新しいバージョンの **PostgreSQL** に移行するための前提条件として、すべての **PostgreSQL** データベースをバックアップします。

データベースをダンプし、SQL ファイルのバックアップを実行することは、ダンプおよび復元プロセスで必要であり、高速アップグレードメソッドとして推奨されます。

新しいバージョンの PostgreSQL に移行する前に、移行する PostgreSQL バージョンと、移行元と移行先のバージョンの間にあるすべて PostgreSQL バージョンの アップストリームの互換性ノート を参照してください。

4.6.1. PostgreSQL 15 と PostgreSQL 16 間の主な違い

PostgreSQL 16 では、次の主な変更点が導入されました。

postmasters バイナリーが利用できなくなりました

PostgreSQL が **postmaster** バイナリーとともに配布されなくなりました。提供されている **systemd** ユニットファイル (**systemctl start postgres** コマンド) を使用して **postgresql** サーバーを起動するユーザーは、この変更の影響を受けません。以前に **postmaster** バイナリーを介して **postgresql** サーバーを直接起動していた場合は、今後は代わりに **postgres** バイナリーを使用する必要があります。

ドキュメントがパッケージに同梱されなくなりました

PostgreSQL のパッケージで PDF 形式のドキュメントが提供されなくなりました。代わりに オンラインドキュメント を使用してください。

4.6.2. PostgreSQL 13 と PostgreSQL 15 間の主な違い

PostgreSQL 15 では、以下の後方互換性のない変更が導入されました。

パブリックスキーマのデフォルトパーミッション

パブリックスキーマのデフォルトパーミッションは、PostgreSQL 15 で変更されています。新規に作成されたユーザーは、**GRANT ALL ON SCHEMA public TO myuser;** コマンドを使用して、権限を明示的に付与する必要があります。

次の例は PostgreSQL 13 以前で動作します。

```
postgres=# CREATE USER mydbuser;
postgres=# \c postgres mydbuser
postgres=$ CREATE TABLE mytable (id int);
```

次の例は PostgreSQL 15 以降で動作します。

```
postgres=# CREATE USER mydbuser;
postgres=# GRANT ALL ON SCHEMA public TO mydbuser;
postgres=# \c postgres mydbuser
postgres=$ CREATE TABLE mytable (id int);
```

注記

pg_hba.conf ファイルで **mydbuser** のアクセス権が適切に設定されていることを確認してください。詳細は、PostgreSQL ユーザーの作成 を参照してください。

PQsendQuery() がパイプラインモードでサポートされなくなる

PostgreSQL 15 以降、**libpq PQsendQuery()** 関数はパイプラインモードでサポートされなくなりました。影響を受けるアプリケーションを変更して、代わりに **PQsendQueryParams()** 関数を使用します。

4.6.3. pg_upgrade ユーティリティーを使用した高速アップグレード

システム管理者は、高速アップグレード方法を使用して最新バージョンの PostgreSQL にアップグレードできます。高速アップグレードを実行するには、バイナリーデータファイルを **/var/lib/pgsql/data/** ディレクトリーにコピーし、**pg_upgrade** ユーティリティーを使用します。

この方法を使用すると、次のバージョン間でデータを移行できます。

- RHEL 8 バージョンの **PostgreSQL 12** から RHEL バージョンの **PostgreSQL 13** へ
- RHEL 8 または 9 バージョンの **PostgreSQL 13** から RHEL バージョンの **PostgreSQL 15** へ
- RHEL 8 または 9 バージョンの **PostgreSQL 15** から RHEL バージョンの **PostgreSQL 16** へ

以下の手順では、高速アップグレードメソッドを使用して、RHEL 8 バージョンの **PostgreSQL 12** から、RHEL 9 バージョンの **PostgreSQL 13** への移行方法を説明します。**12** 以外の **postgresql** ストリームからの移行には、以下のいずれかの方法を使用します。

- RHEL 8 で **PostgreSQL** サーバーをバージョン 12 に更新し、**pg_upgrade** ユーティリティーを使用して RHEL 9 バージョンの **PostgreSQL 13** への高速アップグレードを実行します。
- ダンプおよび復元のアップグレードは、RHEL 8 バージョンの **PostgreSQL** と、RHEL 9 の **PostgreSQL** 以降のバージョンとの間に直接使用します。

前提条件

- アップグレードを実行する前に、**PostgreSQL** データベースに保存されているすべてのデータのバックアップを作成します。デフォルトでは、すべてのデータは、RHEL 8 および RHEL 9 システムの両方の **/var/lib/pgsql/data/** ディレクトリーに保存されます。

手順

1. RHEL 9 システムで、**postgresql-server** パッケージおよび **postgresql-upgrade** パッケージをインストールします。

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

   必要に応じて、RHEL 8 で **PostgreSQL** サーバーモジュールを使用していた場合は、それらを 2 つのバージョンで RHEL 9 システムにもインストールし、**PostgreSQL 12** (**postgresql-upgrade** パッケージでインストール) および対象バージョンの **PostgreSQL 13** (**postgresql-server** パッケージでインストール) の両方に対してコンパイルします。サードパーティーの **PostgreSQL** サーバーモジュールをコンパイルする必要がある場合は、**postgresql-devel** パッケージと **postgresql-upgrade-devel** パッケージの両方に対してビルドしてください。

2. 以下の項目を確認します。

   - 基本設定 - RHEL 9 システムで、サーバーがデフォルトの **/var/lib/pgsql/data** ディレクトリーを使用し、データベースが正しく初期化され、有効になっているかどうかを確認します。さらに、データファイルは、**/usr/lib/systemd/system/postgresql.service** ファイルに記載されているパスと同じパスに保存する必要があります。

- PostgreSQL サーバー - システムは複数の PostgreSQL サーバーを実行できます。これらのすべてのサーバーのデータディレクトリーが独立して処理されていることを確認してください。
- PostgreSQL サーバーモジュール - RHEL 8 で使用されていた PostgreSQL サーバーモジュールも、RHEL 9 システムにインストールされていることを確認してください。プラグインは **/usr/lib64/pgsql/** ディレクトリーにインストールされていることに注意してください。

3. データのコピー時に、**postgresql** サービスがソースおよびターゲットのシステムで稼働していないことを確認します。

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

4. データベースファイルをソースの場所から RHEL 9 システムの **/var/lib/pgsql/data/** ディレクトリーにコピーします。

5. PostgreSQL ユーザーで以下のコマンドを実行して、アップグレードプロセスを実行します。

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

これでバックグラウンドで **pg_upgrade** プロセスが開始します。

障害が発生すると、**postgresql-setup** は通知のエラーメッセージを提供します。

6. **/var/lib/pgsql/data-old** から新規クラスターに、以前の設定をコピーします。
高速アップグレードは、新しいデータスタックで以前の設定を再利用せず、設定がゼロから生成されることに注意してください。古い設定と新しい設定を手動で組み合わせたい場合は、データディレクトリーの *.conf ファイルを使用します。

7. 新しい PostgreSQL サーバーを起動します。

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

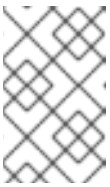
8. 新しいデータベースクラスターを分析します。

- PostgreSQL 13 の場合:

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

- PostgreSQL 15 以降の場合:

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



注記

場合によって、**ALTER COLLATION name REFRESH VERSION** を使用する必要があります。詳細は、アップストリームのドキュメント を参照してください。

9. システムの起動時に、新しい PostgreSQL サーバーを自動的に起動させる場合は、次のコマンドを実行します。

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

4.6.4. ダンプおよび復元のアップグレード

ダンプおよび復元のアップグレードを使用する場合は、すべてのデータベースのコンテンツを SQL ファイルのダンプファイルにダンプする必要があります。ダンプおよび復元のアップグレードは高速なアップグレード方法よりも低速であり、生成された SQL ファイルで手動修正が必要になる場合があります。

この方法を使用して、RHEL 8 バージョンの **PostgreSQL** から、RHEL 9 の PostgreSQL の同等以降のバージョンにデータを移行できます。

RHEL 8 および RHEL 9 システムでは、**PostgreSQL** データは、デフォルトで **/var/lib/pgsql/data/** ディレクトリーに保存されます。

ダンプおよび復元のアップグレードを実行するには、ユーザーを **root** に変更します。

以下の手順では、RHEL 8 デフォルトバージョンの **PostgreSQL 10** から、RHEL 9 バージョンの **PostgreSQL 13** への移行を説明します。

手順

1. RHEL 8 システムで **PostgreSQL 10** サーバーを起動します。

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

2. RHEL 8 システムで、すべてのデータベースのコンテンツを **pgdump_file.sql** ファイルにダンプします。

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

3. データベースが正しくダンプされたことを確認します。

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

   これにより、ダンプされた sql ファイルのパスが **/var/lib/pgsql/pgdump_file.sql** に表示されます。

4. RHEL 9 システムで、**postgresql-server** パッケージをインストールします。

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

   必要に応じて、RHEL 8 で **PostgreSQL** サーバーモジュールを使用していた場合は、RHEL 9 システムにもインストールしてください。サードパーティーの **PostgreSQL** サーバーモジュールをコンパイルする必要がある場合は、**postgresql-devel** パッケージに対してビルドします。

5. RHEL 9 システムで、新しい **PostgreSQL** サーバーのデータディレクトリーを初期化します。

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

6. RHEL 9 システムで、**pgdump_file.sql** を **PostgreSQL** ホームディレクトリーにコピーし、ファイルが正しくコピーされたことを確認します。

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

7. RHEL 8 システムから設定ファイルをコピーします。

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

コピーされる設定ファイルは、以下のとおりです。

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

8. RHEL 9 システムで、新しい **PostgreSQL** サーバーを起動します。

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

9. RHEL 9 システムで、ダンプされた sql ファイルからデータをインポートします。

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