Red Hat Enterprise Linux 9

데이터베이스 서버 구성 및 사용

데이터베이스 서버에 데이터 설치, 구성, 백업 및 마이그레이션

Last Updated: 2024-09-07

Red Hat Enterprise Linux 9 데이터베이스 서버 구성 및 사용

데이터베이스 서버에 데이터 설치, 구성, 백업 및 마이그레이션

법적 공지

초록

Red Hat Enterprise Linux 9에 MariaDB, MySQL 또는 PostgreSQL 데이터베이스 서버를 설치합니다. 선택한 데이터베이스 서버를 구성하고, 데이터를 백업하고, 데이터를 최신 버전의 데이터베이스 서버로 마이그레이션합니다.

차례

RED HAT 문서에 관한 피드백 제공

문서에 대한 피드백에 감사드립니다. 어떻게 개선할 수 있는지 알려주십시오.

Jira를 통해 피드백 제출 (등록 필요)

1. Jira 웹 사이트에 로그인합니다.
2. 상단 탐색 모음에서 **생성** 을 클릭합니다.
3. **Summary** (요약) 필드에 설명 제목을 입력합니다.
4. **Description** (설명) 필드에 개선을 위한 제안을 입력합니다. 문서의 관련 부분에 대한 링크를 포함합니다.
5. 대화 상자 하단에서 **생성** 을 클릭합니다.

1장. 데이터베이스 서버 소개

데이터베이스 서버는 DBMS(데이터베이스 관리 시스템)의 기능을 제공하는 서비스입니다. DBMS는 데이터베이스 관리를 위한 유틸리티를 제공하고 최종 사용자, 애플리케이션 및 데이터베이스와 상호 작용합니다.

Red Hat Enterprise Linux 9는 다음과 같은 데이터베이스 관리 시스템을 제공합니다.

- **MariaDB 10.5**
- **MariaDB 10.11** - RHEL 9.4 이후 사용 가능
- **MySQL 8.0**
- **PostgreSQL 13**
- **PostgreSQL 15** - RHEL 9.2 이후 사용 가능
- **PostgreSQL 16** - RHEL 9.4 이후 사용 가능
- **redis 6**

2장. MARIADB 사용

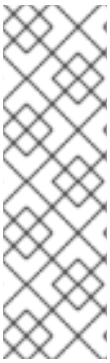
MariaDB 서버는 **MySQL** 기술을 기반으로 하는 오픈 소스 빠르고 강력한 데이터베이스 서버입니다. **MariaDB** 는 데이터를 구조화된 정보로 변환하고 데이터에 액세스하기 위한 SQL 인터페이스를 제공하는 관계형 데이터베이스입니다. 여러 스토리지 엔진 및 플러그인뿐만 아니라 GIS(Gyge Information System) 및 JSON(JavaScript Object Notation) 기능이 포함되어 있습니다.

RHEL 시스템에 **MariaDB** 를 설치하고 구성하는 방법, MariaDB 데이터를 백업하는 방법, 이전 **MariaDB** 버전에서 마이그레이션하는 방법, **MariaDB Galera Cluster** 를 사용하여 데이터베이스를 복제하는 방법을 알아봅니다.

2.1. MARIADB 설치

RHEL 9.0은 RPM 패키지로 쉽게 설치할 수 있는 이 Application Stream의 초기 버전으로 **MariaDB 10.5** 를 제공합니다. 추가 **MariaDB** 버전은 RHEL 9의 마이너 릴리스에서 라이프 사이클이 짧은 모듈로 제공됩니다.

RHEL 9.4에서는 **MariaDB 10.11** 을 **mariadb:10.11** 모듈 스트림으로 도입했습니다.



참고

설계상 동일한 모듈의 두 개 이상 버전(스트림)을 병렬로 설치할 수 없습니다. 따라서 **mariadb** 모듈에서 사용 가능한 스트림 중 하나만 선택해야 합니다. 컨테이너에서 다른 버전의 **MariaDB** 데이터베이스 서버를 사용할 수 있습니다. 컨테이너에서 여러 MariaDB 버전 실행을 참조하십시오.

충돌하는 RPM 패키지로 인해 RHEL 9에서 **MariaDB** 및 **MySQL** 데이터베이스 서버를 병렬로 설치할 수 없습니다. 컨테이너에서 병렬로 **MariaDB** 및 **MySQL** 데이터베이스 서버를 사용할 수 있습니다. 컨테이너에서 여러 MySQL 및 MariaDB 버전 실행을 참조하십시오.

MariaDB 를 설치하려면 다음 절차를 사용합니다.

절차

1. **MariaDB** 서버 패키지를 설치합니다.
   a. **MariaDB 10.5** 의 경우 RPM 패키지의 경우:

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

   b. **mariadb** 모듈에서 스트림(버전) **11** 을 선택하고 서버 프로필을 지정하여 **MariaDB 10.11** 의 경우 다음과 같습니다.

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

2. **mariadb** 서비스를 시작합니다.

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

3. 부팅 시 시작되도록 **mariadb** 서비스를 활성화합니다.

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

2.1.1. 컨테이너에서 여러 MariaDB 버전 실행

동일한 호스트에서 다른 버전의 MariaDB 를 실행하려면 동일한 모듈의 여러 버전(스트림)을 병렬로 설치할 수 없기 때문에 컨테이너에서 실행합니다.

사전 요구 사항

- **container-tools** meta-package가 설치되어 있습니다.

절차

1. Red Hat 고객 포털 계정을 사용하여 **registry.redhat.io** 레지스트리에 인증합니다.

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

   컨테이너 레지스트리에 이미 로그인한 경우 이 단계를 건너뜁니다.

2. 컨테이너에서 MariaDB 10.5 를 실행합니다.

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

   이 컨테이너 이미지 사용에 대한 자세한 내용은 Red Hat Ecosystem Catalog 를 참조하십시오.

3. 컨테이너에서 MariaDB 10.11 을 실행합니다.

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

   이 컨테이너 이미지 사용에 대한 자세한 내용은 Red Hat Ecosystem Catalog 를 참조하십시오.

참고

두 데이터베이스 서버의 컨테이너 이름과 호스트 포트는 달라야 합니다.

4. 클라이언트가 네트워크의 데이터베이스 서버에 액세스할 수 있도록 하려면 방화벽에서 호스트 포트를 엽니다.

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

검증

1. 실행 중인 컨테이너에 대한 정보를 표시합니다.

```
$ podman ps
```

2. 데이터베이스 서버에 연결하고 root로 로그인합니다.

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

추가 리소스

- 컨테이너 빌드, 실행 및 관리
- Red Hat Ecosystem Catalog에서 컨테이너 검색

2.2. MARIADB 구성

네트워킹에 사용할 **MariaDB** 서버를 구성하려면 다음 절차를 사용합니다.

절차

1. **/etc/my.cnf. d/mariadb-server.cnf 파일의 [mysqld]** 섹션을 편집합니다. 다음 구성 지시문을 설정할 수 있습니다.
   - **bind-address** - 서버가 수신 대기하는 주소입니다. 가능한 옵션은 다음과 같습니다.
     - 호스트 이름
     - IPv4 주소
     - IPv6 주소
   - **skip-networking** - 서버가 TCP/IP 연결을 수신하는지 여부를 제어합니다. 가능한 값은 다음과 같습니다.
     - 0 - 모든 클라이언트 수신 대기
     - 1 - 로컬 클라이언트만 수신 대기
   - **포트** - **MariaDB** 가 TCP/IP 연결을 수신 대기하는 포트입니다.
2. **mariadb** 서비스를 다시 시작합니다.

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

2.3. MARIADB 서버에서 TLS 암호화 설정

기본적으로 **MariaDB** 는 암호화되지 않은 연결을 사용합니다. 보안 연결을 위해 **MariaDB** 서버에서 TLS 지원을 활성화하고 암호화된 연결을 설정하도록 클라이언트를 구성합니다.

2.3.1. MariaDB 서버에 CA 인증서, 서버 인증서 및 개인 키 배치

MariaDB 서버에서 TLS 암호화를 활성화하려면 **MariaDB** 서버에 인증 기관(CA) 인증서, 서버 인증서 및 개인 키를 저장할 수 있습니다.

사전 요구 사항

- Privacy Enhanced Mail (PEM) 형식의 다음 파일이 서버에 복사되었습니다.
  - 서버의 개인 키: **server.example.com.key.pem**
  - 서버 인증서: **server.example.com.crt.pem**
  - CA(인증 기관) 인증서: **ca.crt.pem**

개인 키 및 CSR(인증서 서명 요청) 생성 및 CA에서 인증서 요청에 대한 자세한 내용은 CA 문서를 참조하십시오.

절차

1. CA 및 서버 인증서를 **/etc/pki/tls/certs/** 디렉터리에 저장합니다.

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

2. MariaDB 서버가 파일을 읽을 수 있도록 CA 및 서버 인증서에 대한 권한을 설정합니다.

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

인증서는 보안 연결을 설정하기 전에 통신의 일부이므로 모든 클라이언트는 인증 없이 검색할 수 있습니다. 따라서 CA 및 서버 인증서 파일에 대한 엄격한 권한을 설정할 필요가 없습니다.

3. 서버의 개인 키를 **/etc/pki/tls/private/** 디렉터리에 저장합니다.

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

4. 서버의 개인 키에 보안 권한을 설정합니다.

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

권한이 없는 사용자가 개인 키에 액세스할 수 있는 경우 MariaDB 서버에 대한 연결이 더 이상 안전하지 않습니다.

5. SELinux 컨텍스트를 복원하십시오.

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

2.3.2. MariaDB 서버에서 TLS 구성

보안을 개선하려면 MariaDB 서버에서 TLS 지원을 활성화합니다. 결과적으로 클라이언트는 TLS 암호화를 사용하여 서버로 데이터를 전송할 수 있습니다.

사전 요구 사항

- MariaDB 서버를 설치했습니다.
- **mariadb** 서비스가 실행 중입니다.
- Privacy Enhanced Mail (PEM) 형식의 다음 파일은 서버에 있으며 **mysql** 사용자가 읽을 수 있습니다.
  - 서버의 개인 키: **/etc/pki/tls/private/server.example.com.key.pem**
  - 서버 인증서: **/etc/pki/tls/certs/server.example.com.crt.pem**
  - CA(인증 기관) 인증서 **/etc/pki/tls/certs/ca.crt.pem**

- 서버 인증서의 주체 고유 이름(DN) 또는 subject alternatives name(SAN) 필드는 서버의 호스트 이름과 일치합니다.
- 서버가 RHEL 9.2 이상을 실행하고 FIPS 모드가 활성화된 경우 클라이언트는 확장 마스터 시크릿(Extended Master Secret) 확장을 지원하거나 TLS 1.3을 사용해야 합니다. TLS 1.2 연결이 없는 경우 실패합니다. 자세한 내용은 RHEL 9.2 이상에 적용된 Red Hat Knowledgebase 솔루션 TLS 확장 "확장 마스터 시크릿"을 참조하십시오.

절차

1. **/etc/my.cnf.d/mariadb-server-tls.cnf** 파일을 생성합니다.
   a. 다음 콘텐츠를 추가하여 개인 키, 서버 및 CA 인증서에 대한 경로를 구성합니다.

```
[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. CRL(인증서 해지 목록)이 있는 경우 이를 사용하도록 MariaDB 서버를 구성합니다.

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

   c. 선택 사항: 암호화 없이 연결 시도를 거부합니다. 이 기능을 활성화하려면 다음을 추가합니다.

```
require_secure_transport = on
```

   d. 선택 사항: 서버가 지원해야 하는 TLS 버전을 설정합니다. 예를 들어 TLS 1.2 및 TLS 1.3을 지원하려면 다음을 추가합니다.

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

   기본적으로 서버는 TLS 1.1, TLS 1.2 및 TLS 1.3을 지원합니다.

2. **mariadb** 서비스를 다시 시작합니다.

```
# systemctl restart mariadb
```

검증

문제 해결을 간소화하려면 TLS 암호화를 사용하도록 로컬 클라이언트를 구성하기 전에 MariaDB 서버에서 다음 단계를 수행합니다.

1. MariaDB 에 TLS 암호화가 활성화되어 있는지 확인합니다.

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

   have_ssl 변수를 **yes** 로 설정하면 TLS 암호화가 활성화됩니다.

2. 특정 TLS 버전만 지원하도록 MariaDB 서비스를 구성한 경우 **tls_version** 변수를 표시합니다.

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

추가 리소스

- MariaDB 서버에 CA 인증서, 서버 인증서 및 개인 키 배치

2.3.3. 특정 사용자 계정에 대해 TLS 암호화 연결 필요

중요한 데이터에 액세스할 수 있는 사용자는 항상 TLS 암호화 연결을 사용하여 네트워크를 통해 암호화되지 않은 데이터를 전송하지 않도록 해야 합니다.

모든 연결(에서 required_secure_transport =)에 보안 전송이 필요한서버에서 를 구성할 수 없는 경우 TLS 암호화가 필요하도록 개별 사용자 계정을 구성합니다.

사전 요구 사항

- MariaDB 서버에는 TLS 지원이 활성화되어 있습니다.
- 보안 전송을 요구하도록 구성하는 사용자가 있습니다.

절차

1. 관리자로 MariaDB 서버에 연결합니다.

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

   관리자가 서버에 원격으로 액세스할 수 있는 권한이 없는 경우 MariaDB 서버에서 명령을 수행하고 localhost 에 연결합니다.

2. REQUIRE SSL 절을 사용하여 사용자가 TLS 암호화 연결을 사용하여 연결해야 함을 적용합니다.

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

검증

1. TLS 암호화를 사용하여 서버에 예제 사용자로 연결합니다.

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

   오류가 표시되지 않고 대화형 MariaDB 콘솔에 액세스할 수 있는 경우 TLS와의 연결이 성공합니다.

2. TLS가 비활성화된 예제 사용자로 연결을 시도합니다.

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

이 사용자에게는 TLS가 필요하지만 비활성화됨(**--skip-ssl**)이므로 서버에서 로그인 시도를 거부했습니다.

추가 리소스

- MariaDB 서버에서 TLS 암호화 설정

2.4. MARIADB 클라이언트에서 전역적으로 TLS 암호화 활성화

MariaDB 서버가 TLS 암호화를 지원하는 경우 보안 연결만 설정하고 서버 인증서를 확인하도록 클라이언트를 구성합니다. 다음 절차에서는 서버에 있는 모든 사용자에 대해 TLS 지원을 활성화하는 방법을 설명합니다.

2.4.1. 기본적으로 TLS 암호화를 사용하도록 MariaDB 클라이언트 구성

RHEL에서는 **MariaDB** 클라이언트가 TLS 암호화를 사용하도록 구성하고 서버 인증서의 CN(Common Name)이 사용자가 연결하는 호스트 이름과 일치하는지 확인할 수 있습니다. 따라서 메시지 가로채기(man-in-the-middle) 공격을 방지합니다.

사전 요구 사항

- **MariaDB** 서버에는 TLS 지원이 활성화되어 있습니다.
- 서버의 인증서를 발급한 CA(인증 기관)가 RHEL에서 신뢰할 수 없는 경우 CA 인증서가 클라이언트에 복사됩니다.
- MariaDB 서버가 RHEL 9.2 이상을 실행하고 FIPS 모드가 활성화된 경우 이 클라이언트는 확장 마스터 시크릿(Extended Master Secret) 확장을 지원하거나 TLS 1.3을 사용합니다. TLS 1.2 연결이 없는 경우 실패합니다. 자세한 내용은 RHEL 9.2 이상에 적용된 Red Hat Knowledgebase 솔루션 TLS 확장 "확장 마스터 시크릿"을 참조하십시오.

절차

1. RHEL이 서버의 인증서를 발급한 CA를 신뢰하지 않는 경우:
   a. CA 인증서를 **/etc/pki/ca-trust/source/anchors/** 디렉터리에 복사합니다.

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

   b. 모든 사용자가 CA 인증서 파일을 읽을 수 있도록 권한을 설정합니다.

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

   c. CA 신뢰 데이터베이스를 다시 빌드합니다.

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

2. 다음 콘텐츠를 사용하여 **/etc/my.cnf.d/mariadb-client-tls.cnf** 파일을 생성합니다.

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

이러한 설정은 **MariaDB** 클라이언트가 TLS 암호화(**ssl**)를 사용하고 클라이언트가 서버 인증서의 CN과 호스트 이름을 비교하는 것을 정의합니다(**ssl-verify-server-cert**).

검증

- 호스트 이름을 사용하여 서버에 연결하고 서버 상태를 표시합니다.

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

SSL 항목에 **사용 중 암호화가 있는 경우...**, 연결이 암호화됩니다.

이 명령에서 사용하는 사용자에게는 원격으로 인증할 수 있는 권한이 있습니다.

연결하는 호스트 이름이 서버의 TLS 인증서의 호스트 이름과 일치하지 않으면 **ssl-verify-server-cert** 매개변수로 인해 연결이 실패합니다. 예를 들어 **localhost** 에 연결하는 경우 :

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

추가 리소스

- **mysql(1) 도움말** 페이지의 **--ssl*** 매개 변수 설명.

2.5. MARIADB 데이터 백업

Red Hat Enterprise Linux 9의 **MariaDB** 데이터베이스에서 데이터를 백업하는 방법에는 두 가지가 있습니다.

- 논리적 백업
- 물리적 백업

논리적 백업 은 데이터를 복원하는 데 필요한 SQL 문으로 구성됩니다. 이러한 유형의 백업은 일반 텍스트 파일로 정보와 레코드를 내보냅니다.

물리적 백업보다 논리적 백업의 주요 장점은 이식성과 유연성입니다. 데이터는 물리적 백업에서 사용할 수 없는 다른 하드웨어 구성, **MariaDB** 버전 또는 DBMS(데이터베이스 관리 시스템)에서 복원할 수 있습니다.

mariadb.service 가 실행 중인 경우 논리적 백업을 수행할 수 있습니다. 논리적 백업에는 로그 및 구성 파일이 포함되지 않습니다.

물리적 백업 은 콘텐츠를 저장하는 파일과 디렉토리의 복사본으로 구성됩니다.

물리적 백업은 논리적 백업과 비교하여 다음과 같은 이점이 있습니다.

- 출력이 더 작습니다.
- 백업은 크기가 작습니다.

- 백업 및 복원 속도가 빨라집니다.
- 백업에는 로그 및 구성 파일이 포함됩니다.

물리적 백업은 **mariadb.service** 가 실행 중이 아니거나 백업 중 변경을 방지하기 위해 데이터베이스의 모든 테이블이 잠겨 있는 경우 수행해야 합니다.

다음 MariaDB 백업 방법 중 하나를 사용하여 MariaDB 데이터베이스에서 데이터를 백업할 수 있습니다.

- **mariadb-dump**를 사용한 논리 백업
- **Mariabackup** 유틸리티를 사용한 물리적 온라인 백업
- 파일 시스템 백업
- 백업 솔루션으로의 복제

2.5.1. mariadb-dump를 사용하여 논리적 백업 수행

mariadb-dump 클라이언트는 백업 또는 다른 데이터베이스 서버로 전송하기 위해 데이터베이스 컬렉션을 덤프하는 데 사용할 수 있는 백업 유틸리티입니다. mariadb-dump 의 출력은 일반적으로 서버 테이블 구조를 다시 생성하거나, 데이터로 채우거나, 둘 다로 구성되는 SQL 문으로 구성됩니다. mariadb-dump 는 XML 및 CSV와 같은 구분된 텍스트 형식을 포함한 다른 형식의 파일도 생성할 수 있습니다.

mariadb-dump 백업을 수행하려면 다음 옵션 중 하나를 사용할 수 있습니다.

- 하나 이상의 선택된 데이터베이스 백업
- 모든 데이터베이스 백업
- 하나의 데이터베이스에서 테이블의 하위 집합 백업

절차

- 단일 데이터베이스를 덤프하려면 다음을 실행합니다.

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

- 한 번에 여러 데이터베이스를 덤프하려면 다음을 실행합니다.

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

- 모든 데이터베이스를 덤프하려면 다음을 실행합니다.

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

- 하나 이상의 덤프된 전체 데이터베이스를 서버에 다시 로드하려면 다음을 실행합니다.

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

- 데이터베이스를 원격 MariaDB 서버에 로드하려면 다음을 실행합니다.

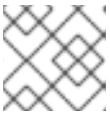
```
# mariadb --host=remote_host < backup-file.sql
```

- 하나의 데이터베이스에서 테이블 서브 세트를 덤프하려면 **mariadb-dump** 명령 끝에 선택한 테이블 목록을 추가합니다.

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

- 하나의 데이터베이스에서 덤프된 테이블의 하위 집합을 로드하려면 다음을 실행합니다.

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



참고

db_name 데이터베이스는 이 시점에 있어야 합니다.

- mariadb-dump 에서 지원하는 옵션 목록을 보려면 다음을 실행합니다.

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

추가 리소스

- MariaDB 문서 - mariadb-dump

2.5.2. Mariabackup 유틸리티를 사용하여 물리적 온라인 백업 수행

mariabackup 은 Percona XtraBackup 기술을 기반으로 하는 유틸리티로서 InnoDB, Aria, MyISAM 테이블의 물리적 온라인 백업을 수행할 수 있습니다. 이 유틸리티는 AppStream 리포지토리에서 **mariadb-backup** 패키지에서 제공합니다.

mariabackup 은 암호화 및 압축된 데이터를 포함하는MariaDB 서버에 대한 전체 백업 기능을 지원합니다.

사전 요구 사항

- **mariadb-backup** 패키지가 시스템에 설치됩니다.

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

- 백업이 실행될 사용자 자격 증명이 포함된 Mariabackup 을 제공해야 합니다. 명령줄 또는 구성 파일을 통해 자격 증명을 제공할 수 있습니다.
- Mariabackup 사용자는 **RELOAD,LOCK TABLES** 및 **REPLICATION CLIENT** 권한이 있어야 합니다.

Mariabackup 을 사용하여 데이터베이스 백업을 만들려면 다음 절차를 따르십시오.

절차

- 명령줄에서 자격 증명을 제공하는 동안 백업을 만들려면 다음을 실행합니다.

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

target-dir 옵션은 백업 파일이 저장되는 디렉터리를 정의합니다. 전체 백업을 수행하려면 대상 디렉터리가 비어 있거나 존재하지 않아야 합니다.

사용자 이름 및 암호 옵션을 사용하여 사용자 이름과 암호를 구성할 수 있습니다.

- 구성 파일에 인증 정보가 설정된 백업을 생성하려면 다음을 수행합니다.

  1. **/etc/my.cnf.d/** 디렉터리에 구성 파일을 만듭니다(예:**/etc/my.cnf.d/mariabackup.cnf**).
  2. 새 파일의 **[xtrabackup]** 또는 **[mysqld]** 섹션에 다음 행을 추가합니다.

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

  3. 백업을 수행합니다.

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

추가 리소스

- Backup을 통한 전체 백업 및 복원

2.5.3. Mariabackup 유틸리티를 사용하여 데이터 복원

백업이 완료되면 다음 옵션 중 하나와 함께 **mariabackup** 명령을 사용하여 백업에서 데이터를 복원할 수 있습니다.

- **--copy-back** 을 사용하면 원본 백업 파일을 유지할 수 있습니다.
- **--move-back** 은 백업 파일을 데이터 디렉터리로 이동하고 원래 백업 파일을 제거합니다.

Mariabackup 유틸리티를 사용하여 데이터를 복원하려면 다음 절차를 따르십시오.

사전 요구 사항

- **mariadb** 서비스가 실행 중이 아닌지 확인합니다.

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

- 데이터 디렉터리가 비어 있는지 확인합니다.
- Mariabackup 사용자는 **RELOAD,LOCK TABLES** 및 **REPLICATION CLIENT** 권한이 있어야 합니다.

절차

1. **mariabackup** 명령을 실행합니다.

   - 데이터를 복원하고 원본 백업 파일을 유지하려면 **--copy-back** 옵션을 사용합니다.

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

   - 데이터를 복원하고 원본 백업 파일을 제거하려면 **--move-back** 옵션을 사용합니다.

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

2. 파일 권한을 수정합니다.
   데이터베이스를 복원할 때 Mariabackup 은 백업의 파일 및 디렉토리 권한을 보존합니다. 그러나 Mariabackup 은 파일을 디스크에 쓰는 사용자 및 그룹으로 데이터베이스를 복원합니다. 백업을 복원한 후 일반적으로 MariaDB 서버의 사용자 및 그룹과 일치하도록 데이터 디렉터리의 소유자(일반적으로 **mysql**)를 조정해야 할 수 있습니다.

   예를 들어 파일의 소유권을 **mysql** 사용자 및 그룹으로 재귀적으로 변경하려면 다음을 수행합니다.

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

3. **mariadb** 서비스를 시작합니다.

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

추가 리소스

- Backup을 통한 전체 백업 및 복원

2.5.4. 파일 시스템 백업 수행

MariaDB 데이터 파일의 파일 시스템 백업을 생성하려면 MariaDB 데이터 디렉터리의 콘텐츠를 백업 위치에 복사합니다.

현재 구성 또는 로그 파일도 백업하려면 다음 절차의 선택적 단계를 사용하십시오.

절차

1. **mariadb** 서비스를 중지합니다.

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

2. 필요한 위치에 데이터 파일을 복사합니다.

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

3. 선택 사항: 구성 파일을 필요한 위치에 복사합니다.

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

4. 선택 사항: 로그 파일을 필요한 위치에 복사합니다.

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

5. **mariadb** 서비스를 시작합니다.

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

6. 백업 위치에서 **/var/lib/mysql** 디렉터리에 백업 데이터를 로드할 때**mysql:mysql** 이 **/var/lib/mysql** 에 있는 모든 데이터의 소유자인지 확인하십시오.

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

2.5.5. 백업 솔루션으로의 복제

복제는 소스 서버를 위한 대체 백업 솔루션입니다. 소스 서버에서 복제본 서버에 복제하는 경우 소스에 영향을 주지 않고 복제본에서 백업을 실행할 수 있습니다. 복제본을 종료하는 동안 계속 소스를 실행하고 복제본에서 데이터를 백업할 수 있습니다.

주의

복제 자체는 충분한 백업 솔루션이 아닙니다. 복제는 하드웨어 장애로부터 소스 서버를 보호하지만 데이터 손실에 대한 보호는 보장하지 않습니다. 이 메서드와 함께 복제본에서 다른 백업 솔루션을 사용하는 것이 좋습니다.

추가 리소스

- Galera를 사용하여 MariaDB 복제
- 백업 솔루션으로 복제

2.6. MARIADB 10.5로 마이그레이션

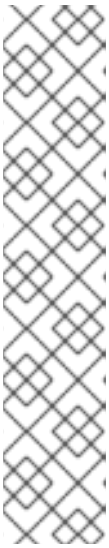
RHEL 8에서는 MariaDB 서버는 각각 별도의 모듈 스트림에서 제공하는 버전 10.3, 10.5, 10.11에서 사용할 수 있습니다. RHEL 9에서는 MariaDB 10.5,MariaDB 10.11 및 MySQL 8.0을 제공합니다.

이 부분에서는 MariaDB 10.3 의 RHEL 8 버전에서 MariaDB 10.5의 RHEL 9 버전으로의 마이그레이션을 설명합니다.

2.6.1. MariaDB 10.3과 MariaDB 10.5의 주요 차이점

MariaDB 10.3과 MariaDB 10.5 간의 중요한 변경 사항은 다음과 같습니다.

- 이제 MariaDB 에서 기본적으로 **unix_socket** 인증 플러그인을 사용합니다. 플러그인을 사용하면 로컬 UNIX 소켓 파일을 통해 MariaDB 에 연결할 때 운영 체제 인증 정보를 사용할 수 있습니다.
- **MariaDB** 는 바이너리라는 **mariadb-*** 와 **mariadb-*** 바이너리를 가리키는 **mysql*** 심볼릭 링크를 추가합니다. 예를 들어 **mysqladmin**,**mysqlaccess**, **mysqlshow** symlinks는 **mariadb-admin**,**mariadb-access**, **mariadb-show** 바이너리를 각각 가리킵니다.
- 각 사용자 역할에 더 잘 맞게 **SUPER** 권한이 여러 권한으로 분할되었습니다. 결과적으로 특정 설명이 필요한 권한을 변경했습니다.
- 병렬 복제에서 **slave_parallel_mode** 는 이제 기본적으로 fake로설정됩니다.
- InnoDB 스토리지 엔진에서 다음 변수의 기본값이 변경되었습니다. **innodb_adaptive_hash_index** 는 **OFF** 로, **innodb_checksum_algorithm**에서 **full_ crc32** 로 변경되었습니다.
- MariaDB 는 이전에 사용된읽기라인 라이브러리 대신 MariaDB 명령 기록(**.mysql_history** 파일)을 관리하는 기본 소프트웨어의 **libedit** 구현을 사용합니다. 이 변경 사항은 **.mysql_history** 파일을 직접 사용하는 사용자에게 영향을 미칩니다. **.mysql_history** 는 MariaDB 또는 MySQL 애플리케이션에서 관리하는 파일이며 사용자는 파일에서 직접 작업해서는 안 됩니다. 사람이 읽을 수 있는 모양은 무의미합니다.



참고

보안을 강화하기 위해 기록 파일을 유지하지 않는 것이 좋습니다. 명령 기록 기록을 비활성화하려면 다음을 수행합니다.

1. 있는 경우 **.mysql_history** 파일을 제거합니다.
2. 다음 방법 중 하나를 사용합니다.
   - **MYSQL_HISTFILE** 변수를 **/dev/null** 로 설정하고 이 설정을 쉘의 시작 파일에 포함합니다.
   - **/dev/null** 로 심볼릭 링크로 **.mysql_history** 파일을 변경합니다.

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

MariaDB Galera Cluster 가 버전 4로 업그레이드되었으며 다음과 같은 주요 변경 사항이 적용됩니다.

- Galera 는 무제한 크기의 복제본 처리를 지원하는 새로운 스트리밍 복제 기능을 추가합니다. 스트리밍 복제를 실행하는 동안 클러스터는 작은 조각으로 트랜잭션을 복제합니다.
- Galera 는 이제 GTID(Global Transaction ID)를 완전히 지원합니다.
- 추가 옵션을 구성하지 않고 최종 사용자가 **wsrep** 복제를 시작하지 못하도록 **/etc/my.cnf.d/galera.cnf** 파일의 **wsrep_on** 옵션의 기본값이 **1** 에서 **0** 으로 변경되었습니다.

MariaDB 10.5 의 PAM 플러그인 변경 사항은 다음과 같습니다.

- MariaDB 10.5 에는 새로운 버전의 PAM(Pluggable Authentication Modules) 플러그인이 추가되었습니다. PAM 플러그인 버전 2.0은 MariaDB 가 추가 PAM 모듈을 사용할 수 있도록 별도의 **setuid** 루트 도우미 바이너리를 사용하여 PAM 인증을 수행합니다.
- 도우미 바이너리는 **mysql** 그룹의 사용자만 실행할 수 있습니다. 기본적으로 그룹에는**mysql** 사용자만 포함되어 있습니다. Red Hat은 이 도우미 유틸리티를 통해 액세스하거나 로깅하지 않고 관리자가 **mysql** 그룹에 사용자를 추가하지 않도록 권장합니다.
- MariaDB 10.5 에서 PAM(Pluggable Authentication Modules) 플러그인 및 관련 파일이 새 패키지 **mariadb-pam** 으로 이동되었습니다. 결과적으로 **MariaDB** 에 PAM 인증을 사용하지 않는 새로운 **setuid root** 바이너리가 시스템에 소개되지 않았습니다.
- **mariadb-pam** 패키지에는 두 PAM 플러그인 버전이 모두 포함되어 있습니다. 버전 2.0은 기본값이며 버전 1.0은 **auth_pam_v1** 공유 오브젝트 라이브러리로 사용할 수 있습니다.
- **mariadb-pam** 패키지는 기본적으로 MariaDB 서버와 함께 설치되지 않습니다.MariaDB 10.5 에서 PAM 인증 플러그인을 사용할 수 있도록 하려면 **mariadb-pam** 패키지를 수동으로 설치합니다.

2.6.2. RHEL 8 버전의 MariaDB 10.3에서 RHEL 9 버전의 MariaDB 10.5로 마이그레이션

이 절차에서는 **mariadb-upgrade** 유틸리티를 사용하여 MariaDB 10.3 에서 MariaDB 10.5 로 마이그레이션하는 방법을 설명합니다.

mariadb-upgrade 유틸리티는 **mariadb-server** 패키지의 종속성으로 설치된 **mariadb-server- utils** 하위 패키지에서 제공합니다.

사전 요구 사항

- 업그레이드를 수행하기 전에 MariaDB 데이터베이스에 저장된 모든 데이터를 백업합니다.

절차

1. **mariadb-server** 패키지가 RHEL 9 시스템에 설치되었는지 확인합니다.

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

2. 데이터를 복사할 때 **mariadb** 서비스가 소스 및 대상 시스템에서 실행되고 있지 않은지 확인합니다.

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

3. 소스 위치의 데이터를 RHEL 9 대상 시스템의 **/var/lib/mysql/** 디렉터리에 복사합니다.

4. 대상 시스템에서 복사된 파일에 대한 적절한 권한 및 SELinux 컨텍스트를 설정합니다.

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

5. **mysql:mysql** 이 **/var/lib/mysql** 디렉터리에 있는 모든 데이터의 소유자인지 확인합니다.

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

6. **/etc/my.cnf.d/** 에 있는 옵션 파일이 MariaDB 10.5 에 유효한 옵션만 포함하도록 구성을 조정합니다. 자세한 내용은 MariaDB 10.4 및 MariaDB 10.5 의 업스트림 문서를 참조하십시오.

7. 대상 시스템에서 MariaDB 서버를 시작합니다.

   - 독립 실행형으로 데이터베이스를 업그레이드하는 경우:

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

   - Galera 클러스터 노드를 업그레이드하는 경우 다음을 수행합니다.

   ```
   # galera_new_cluster
   ```

   mariadb 서비스가 자동으로 시작됩니다.

8. mariadb-upgrade 유틸리티를 실행하여 내부 테이블을 확인하고 복구합니다.

   - 독립 실행형으로 데이터베이스를 업그레이드하는 경우:

   ```
   $ mariadb-upgrade
   ```

   - Galera 클러스터 노드를 업그레이드하는 경우 다음을 수행합니다.

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

중요

즉각적 업그레이드와 관련하여 특정 위험 및 알려진 문제가 있습니다. 예를 들어 일부 쿼리가 작동하지 않거나 업그레이드 전과 다른 순서로 실행될 수 있습니다. 이러한 위험 및 문제에 대한 자세한 내용과 인플레이스 업그레이드에 대한 일반적인 내용은 MariaDB 10.5 릴리스 노트 를 참조하십시오.

2.7. MARIADB 10.5에서 MARIADB 10.11로 업그레이드

이 부분에서는 RHEL 9 내의 MariaDB 10.5 에서 MariaDB 10.11 로의 마이그레이션을 설명합니다.

2.7.1. MariaDB 10.5와 MariaDB 10.11의 주요 차이점

MariaDB 10.5 와 MariaDB 10.11 간의 중요한 변경 사항은 다음과 같습니다.

- 새로운 **sys_schema** 기능은 데이터베이스 사용량에 대한 정보를 제공하는 뷰, 함수 및 프로시저의 컬렉션입니다.
- **CREATE Cryostat**, Cryostat ,**RENAME** Cryostat ,**DROP DATABASE**, DROP DATABASE 및 관련 DDL(Data Definition Language) 문이 원자적입니다. 문을 완전히 완료해야 합니다. 그렇지 않으면 변경 사항이 취소됩니다. **DROP Cryostat**를 사용하여 여러 테이블을 삭제할 때 각 드롭다운만 전체 테이블 목록이 아닌 **atomic**입니다.
- 새로운 **GRANT ... TO PUBLIC** 권한을 사용할 수 있습니다.
- 이제 **SUPER** 및 **READ** 전용 권한이 분리되어 있습니다.
- 이제 새로운 **UUID** 데이터베이스 데이터 유형에 범용 고유 식별자를 저장할 수 있습니다.
- **MariaDB** 는 이제 **SSL(Secure Socket Layer)** 프로토콜 버전 **3**을 지원합니다.
- 이제 **MariaDB** 서버를 시작하려면 올바르게 구성된 **SSL**이 필요합니다. 이전에는 **MariaDB** 에서 **SSL**을 자동으로 비활성화하고 **SSL**을 잘못 구성하는 경우 비보안 연결을 사용했습니다.
- **MariaDB** 는 이제 **natural_sort_key()** 함수를 통해 자연 정렬 순서를 지원합니다.
- 이제 임의의 텍스트 형식에 새로운 **SFORMAT** 함수를 사용할 수 있습니다.
- **utf8** 문자 세트(및 관련 데이터 정렬)는 기본적으로 **utf8mb3** 의 별칭입니다.
- **MariaDB** 는 **UCA(Unicode Collation Algorithm) 14** 데이터 정렬을 지원합니다.
- **MariaDB** 의 **systemd** 소켓 활성화 파일은 이제 **/usr/share/** 디렉토리에서 사용할 수 있습니

다. 이는 업스트림과 달리 RHEL의 기본 구성의 일부가 아닙니다.

- 오류 메시지에 MySQL 대신 MariaDB 문자열이 포함됩니다.
- 이제 중국어로 오류 메시지를 사용할 수 있습니다.
- 기본 logrotate 파일이 크게 변경되었습니다. MariaDB 10.11 로 마이그레이션하기 전에 구성을 검토합니다.
- MariaDB 및 MySQL 클라이언트의 경우 명령줄에 지정된 연결 속성(예: --port=3306)은 이제 tcp,소켓,파이프 또는 메모리 와 같은 클라이언트와 서버 간의 프로토콜 통신 유형을 강제 적용합니다. 이전에는 MariaDB 클라이언트가 UNIX 소켓을 통해 연결된 경우 지정된 포트가 무시되었습니다.

2.7.2. MariaDB 10.5의 RHEL 9 버전에서 MariaDB 10.11로 업그레이드

다음 절차에서는 dnf 및 mariadb-upgrade 유틸리티를 사용하여 mariadb-server RPM 패키지에서 mariadb:10.11 모듈 스트림으로 제공되는 MariaDB 10.5 에서 업그레이드하는 방법을 설명합니다.

mariadb-upgrade 유틸리티는 mariadb-server 패키지의 종속성으로 설치된 mariadb-server- utils 하위 패키지에서 제공합니다.

사전 요구 사항

- 업그레이드를 수행하기 전에 MariaDB 데이터베이스에 저장된 모든 데이터를 백업합니다.

절차

1. MariaDB 서버를 중지합니다.

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

2. 변조되지 않은 MariaDB 10.5 에서 모듈식 MariaDB 10.11 로 전환합니다.

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

3. /etc/my.cnf.d/ 에 있는 옵션 파일이 MariaDB 10.11 에 유효한 옵션만 포함하도록 구성을 조정합니다. 자세한 내용은 MariaDB 10.6 및 MariaDB 10.11 에 대한 업스트림 문서를 참조하십시오.

4. MariaDB 서버를 시작합니다.

   - 독립 실행형으로 데이터베이스를 업그레이드하는 경우:

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

   - Galera 클러스터 노드를 업그레이드하는 경우 다음을 수행합니다.

     ```
     # galera_new_cluster
     ```

     mariadb 서비스가 자동으로 시작됩니다.

5. mariadb-upgrade 유틸리티를 실행하여 내부 테이블을 확인하고 복구합니다.

   - 독립 실행형으로 데이터베이스를 업그레이드하는 경우:

     ```
     # mariadb-upgrade
     ```

   - Galera 클러스터 노드를 업그레이드하는 경우 다음을 수행합니다.

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

중요

즉각적 업그레이드와 관련하여 특정 위험 및 알려진 문제가 있습니다. 예를 들어 일부 쿼리가 작동하지 않거나 업그레이드 전과 다른 순서로 실행될 수 있습니다. 이러한 위험 및 문제에 대한 자세한 내용과 인플레이스 업그레이드에 대한 일반적인 정보는 MariaDB 10.11 릴리스 노트 를 참조하십시오.

2.8. GALERA를 사용하여 MARIADB 복제

Red Hat Enterprise Linux 9에서 Galera 솔루션을 사용하여 MariaDB 데이터베이스를 복제할 수 있습니다.

2.8.1. MariaDB Galera 클러스터 소개

Galera 복제는 여러 MariaDB 서버로 구성된 동기 다중 소스 MariaDB Galera 클러스터 생성을 기반으로 합니다. 일반적으로 복제본이 읽기 전용인 기존의 기본/복제 설정과 달리 MariaDB Galera Cluster의 노드는 모두 쓸 수 있습니다.

Galera 복제와 MariaDB 데이터베이스 간의 인터페이스는 쓰기 세트 복제 API(wsrep API)로 정의됩니다.

MariaDB Galera 클러스터의 주요 기능은 다음과 같습니다.

- 동기 복제
- 액티브-액티브 멀티 소스 토폴로지
- 클러스터 노드에 읽기 및 쓰기
- 자동 멤버십 제어, 클러스터에서 실패한 노드 삭제
- 자동 노드 결합
- 행 수준에서 병렬 복제
- 직접 클라이언트 연결: 사용자가 클러스터 노드에 로그온하고 복제가 실행되는 동안 노드에서 직접 작업할 수 있습니다.

동기 복제는 서버와 관련된 쓰기 집합을 클러스터의 모든 노드에 브로드캐스트하여 커밋 시 트랜잭션을 복제함을 의미합니다. 클라이언트(사용자 애플리케이션)는 기본 MariaDB 와 유사한 DBMS(데이터베이스 관리 시스템)에 직접 연결됩니다.

동기 복제를 사용하면 클러스터의 한 노드에서 동시에 발생한 변경 사항이 클러스터의 다른 노드에서 동시에 수행됩니다.

따라서 동기 복제는 비동기 복제에 비해 다음과 같은 이점이 있습니다.

- 특정 클러스터 노드 간의 변경 사항을 전파하는 데 지연 없음
- 모든 클러스터 노드는 항상 일관성이 유지됩니다.
- 클러스터 노드 중 하나가 충돌하면 최신 변경 사항이 손실되지 않습니다.
- 모든 클러스터 노드의 트랜잭션이 병렬로 실행됩니다.
- 전체 클러스터의 원인

추가 리소스

- Galera 복제 정보
- MariaDB Galera Cluster란
- MariaDB Galera 클러스터 시작하기

2.8.2. MariaDB Galera Cluster를 빌드하기 위한 구성 요소

MariaDB Galera 클러스터를 빌드하려면 시스템에 다음 패키지를 설치해야 합니다.

- mariadb-server-galera - MariaDB Galera 클러스터에 대한 지원 파일 및 스크립트를 포함합니다.
- mariadb-server - 쓰기 세트 복제 API(wsrep API)를 포함하도록 MariaDB 업스트림에서 패

치합니다. 이 API는 Galera 복제와 MariaDB 간의 인터페이스를 제공합니다.

- Galera - MariaDB를 완벽하게 지원하기 위해 MariaDB 업스트림에서 패치했습니다. galera 패키지에는 다음이 포함됩니다.
  - Galera Replication Library 는 전체 복제 기능을 제공합니다.
  - Galera Arbitrator 유틸리티를 분리 장애 시나리오의 투표에 참여하는 클러스터 멤버로 사용할 수 있습니다. 그러나 Galera Arbitrator 는 실제 복제에 참여할 수 없습니다.
  - Galera Systemd 서비스 및 Galera 래퍼 스크립트 (Galera), Galera Arbitrator 유틸리티를 배포하는 데 사용됩니다. RHEL 9는 /usr/lib/systemd/system/garbd.service 및 /usr/sbin/garb-systemd 에 있는 이러한 파일의 업스트림 버전을 제공합니다.

추가 리소스

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

2.8.3. MariaDB Galera 클러스터 배포

MariaDB Galera Cluster 패키지를 배포하고 구성을 업데이트할 수 있습니다. 새 클러스터를 생성하려면 클러스터의 첫 번째 노드를 부트스트랩해야 합니다.

사전 요구 사항

- MariaDB Galera Cluster 패키지를 설치합니다.

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

결과적으로 다음 패키지가 종속 항목과 함께 설치됩니다.

- mariadb-server-galera
- mariadb-server
- galera

MariaDB Galera 클러스터 빌드를 위해 이러한 패키지에 대한 자세한 내용은 MariaDB 클러스터 를 빌드하는 구성 요소를 참조하십시오.

- MariaDB 서버 복제 구성은 시스템을 클러스터에 처음 추가하기 전에 업데이트해야 합니다.

  기본 구성은 /etc/my.cnf.d/galera.cnf 파일에 배포됩니다.

MariaDB Galera 클러스터를 배포하기 전에 /etc/my.cnf.d/galera.cnf 파일에서 다음 문자열로 시작하도록 wsrep_cluster_address 옵션을 설정합니다.

```
gcomm://
```

- 초기 노드의 경우 wsrep_cluster_address 를 빈 목록으로 설정할 수 있습니다.

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

- 다른 모든 노드의 경우 실행 중인 클러스터의 일부인 모든 노드의 주소를 포함하도록 wsrep_cluster_address 를 설정합니다. 예를 들면 다음과 같습니다.

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

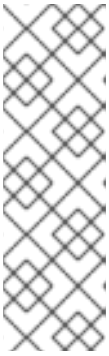
  Galera 클러스터 주소를 설정하는 방법에 대한 자세한 내용은 Galera Cluster Address 를 참조하십시오.

절차

1. 해당 노드에서 다음 래퍼를 실행하여 새 클러스터의 첫 번째 노드를 부트스트랩합니다.

   ```
   # galera_new_cluster
   ```

이 래퍼를 사용하면 **MariaDB** 서버 데몬(**mariadbd**)이 **--wsrep-new-cluster** 옵션과 함께 실행됩니다. 이 옵션은 연결할 기존 클러스터가 없는 정보를 제공합니다. 따라서 노드는 새 클러스터를 식별하기 위해 새 **UUID**를 만듭니다.



참고

mariadb 서비스는 여러 **MariaDB** 서버 프로세스와 상호 작용하는 **systemd** 방법을 지원합니다. 따라서 **MariaDB** 서버가 여러 개 있는 경우 인스턴스 이름을 접미사로 지정하여 특정 인스턴스를 부트스트랩할 수 있습니다.

```
# galera_new_cluster mariadb@node1
```

2. 각 노드에서 다음 명령을 실행하여 다른 노드를 클러스터에 연결합니다.

```
# systemctl start mariadb
```

결과적으로 노드가 클러스터에 연결되고 클러스터 상태와 자체 동기화됩니다.

추가 리소스

- MariaDB Galera 클러스터 시작하기

2.8.4. MariaDB Galera 클러스터에 새 노드 추가

MariaDB Galera 클러스터에 새 노드를 추가하려면 다음 절차를 사용합니다.

이 절차를 사용하여 기존 노드를 다시 연결할 수도 있습니다.

절차

- 특정 노드에서 **/etc/my.cnf.d/galera.cnf** 구성 파일의 **[mariadb]** 섹션에 있는 **wsrep_cluster_address** 옵션에 있는 하나 이상의 기존 클러스터 구성원에게 주소를 제공하십시오.

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

새 노드가 기존 클러스터 노드 중 하나에 연결되면 클러스터의 모든 노드를 볼 수 있습니다.

그러나 클러스터의 모든 노드를 **wsrep_cluster_address** 에 나열하는 것이 좋습니다.

결과적으로 하나 이상의 클러스터 노드가 중단된 경우에도 모든 노드가 다른 클러스터 노드에 연결하여 클러스터에 참여할 수 있습니다. 모든 멤버가 멤버십에 동의하면 클러스터 상태가 변경됩니다. 새 노드의 상태가 클러스터 상태와 다른 경우 새 노드는 증분 상태 전송(**IST**) 또는 **SST**(상태 스냅샷 전송)를 요청하여 다른 노드와 일관성을 보장합니다.

추가 리소스

- **MariaDB Galera** 클러스터 시작하기
- 상태 스냅샷 전송 소개

2.8.5. MariaDB Galera 클러스터 재시작

동시에 모든 노드를 종료하면 클러스터를 중지하고 실행 중인 클러스터가 더 이상 존재하지 않습니다. 그러나 클러스터의 데이터는 여전히 존재합니다.

클러스터를 다시 시작하려면 **MariaDB Galera** 클러스터 구성에 설명된 대로 첫 번째 노드를 부트스트랩합니다.

주의

클러스터가 부트 스트랩되지 않고 첫 번째 노드에서 **mariadbd** 를 **systemctl start mariadb** 명령으로 시작하는 경우 노드는 **/etc/my.cnf.d/galera.cnf** 파일의 **wsrep_cluster_address** 옵션에 나열된 노드 중 하나 이상에 연결을 시도합니다. 현재 실행 중인 노드가 없으면 재시작이 실패합니다.

추가 리소스

- **MariaDB Galera** 클러스터 시작.

2.9. MARIADB 클라이언트 애플리케이션 개발

Red Hat은 MariaDB 클라이언트 라이브러리에 대해 MariaDB 클라이언트 애플리케이션을 개발하는 것이 좋습니다.

MariaDB 클라이언트 라이브러리에 대해 애플리케이션을 빌드하는 데 필요한 개발 파일 및 프로그램은 mariadb-connector-c-devel 패키지에서 제공합니다.

직접 라이브러리 이름을 사용하는 대신 mariadb-connector-c-devel 패키지에 배포되는 mariadb_config 프로그램을 사용합니다. 이 프로그램을 통해 올바른 빌드 플래그가 반환됩니다.

3장. MYSQL 사용

MySQL 서버는 빠르고 강력한 오픈 소스 데이터베이스 서버입니다. MySQL 은 데이터를 구조화된 정보로 변환하고 데이터에 액세스하기 위한 SQL 인터페이스를 제공하는 관계형 데이터베이스입니다. 여러 스토리지 엔진 및 플러그인뿐만 아니라 GIS(Gyge Information System) 및 JSON(JavaScript Object Notation) 기능이 포함되어 있습니다.

RHEL 시스템에 MySQL 을 설치하고 구성하는 방법, MySQL 데이터를 백업하는 방법, 이전 MySQL 버전에서 마이그레이션하는 방법, MySQL 을 복제하는 방법을 알아봅니다.

3.1. MYSQL 설치

RHEL 9.0는 이 애플리케이션 스트림의 초기 버전으로 MySQL 8.0 을 제공하며 RPM 패키지로 쉽게 설치할 수 있습니다.

참고

충돌하는 RPM 패키지로 인해 RHEL 9에서 MySQL 및 MariaDB 데이터베이스 서버를 병렬로 설치할 수 없습니다. 컨테이너에서 병렬로 MySQL 및 MariaDB 데이터베이스 서버를 사용할 수 있습니다. 컨테이너에서 여러 MySQL 및 MariaDB 버전 실행을 참조하십시오.

MySQL 을 설치하려면 다음 절차를 사용하십시오.

절차

1. MySQL 서버 패키지를 설치합니다.

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

2. mysqld 서비스를 시작합니다.

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

3. 부팅 시 mysqld 서비스가 시작되도록 활성화합니다.

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

4. *권장 사항:* **MySQL** 을 설치할 때 보안을 강화하려면 다음 명령을 실행합니다.

```
$ mysql_secure_installation
```

명령은 완전히 대화형 스크립트를 시작하여 프로세스의 각 단계에 대한 메시지를 표시합니다. 이 스크립트를 사용하면 다음과 같은 방법으로 보안을 강화할 수 있습니다.

- **root** 계정의 암호 설정
- 익명 사용자 제거
- 원격 **root** 로그인 허용 (로컬 호스트 내부)

3.1.1. 컨테이너에서 여러 MySQL 및 MariaDB 버전 실행

동일한 호스트에서 **MySQL** 과 **MariaDB** 를 모두 실행하려면 충돌하는 **RPM** 패키지로 인해 이러한 데이터베이스 서버를 병렬로 설치할 수 없기 때문에 컨테이너에서 실행합니다.

이 절차에서는 예를 들어 **MySQL 8.0** 및 **MariaDB 10.5** 가 포함되어 있지만 **Red Hat Ecosystem Catalog**에서 사용할 수 있는 **MySQL** 또는 **MariaDB** 컨테이너 버전을 사용할 수 있습니다.

사전 요구 사항

- **container-tools meta-package**가 설치되어 있습니다.

절차

1. **Red Hat** 고객 포털 계정을 사용하여 **registry.redhat.io** 레지스트리에 인증합니다.

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

컨테이너 레지스트리에 이미 로그인한 경우 이 단계를 건너뜁니다.

2. 컨테이너에서 **MySQL 8.0** 을 실행합니다.

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

이 컨테이너 이미지 사용에 대한 자세한 내용은 Red Hat Ecosystem Catalog 를 참조하십시오.

3. 컨테이너에서 **MariaDB 10.5** 를 실행합니다.

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

이 컨테이너 이미지 사용에 대한 자세한 내용은 Red Hat Ecosystem Catalog 를 참조하십시오.

4. 컨테이너에서 **MariaDB 10.11** 을 실행합니다.

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

이 컨테이너 이미지 사용에 대한 자세한 내용은 Red Hat Ecosystem Catalog 를 참조하십시오.

참고

두 데이터베이스 서버의 컨테이너 이름과 호스트 포트는 달라야 합니다.

5. 클라이언트가 네트워크의 데이터베이스 서버에 액세스할 수 있도록 하려면 방화벽에서 호스트 포트를 엽니다.

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

검증

1. 실행 중인 컨테이너에 대한 정보를 표시합니다.

```
$ podman ps
```

2. 데이터베이스 서버에 연결하고 **root**로 로그인합니다.

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

추가 리소스

- 컨테이너 빌드, 실행 및 관리
- Red Hat Ecosystem Catalog에서 컨테이너 검색

3.2. MYSQL 구성

네트워킹을 위해 **MySQL** 서버를 구성하려면 다음 절차를 사용하십시오.

절차

1. **/etc/my.cnf.d/mysql-server.cnf** 파일의 **[mysqld]** 섹션을 편집합니다. 다음 구성 지시문을 설정할 수 있습니다.
   - **bind-address -** 서버가 수신 대기하는 주소입니다. 가능한 옵션은 다음과 같습니다.
     - 호스트 이름
     - **IPv4** 주소
     - **IPv6** 주소
   - **skip-networking -** 서버가 **TCP/IP** 연결을 수신하는지 여부를 제어합니다. 가능한 값은

다음과 같습니다.

- 0 - 모든 클라이언트 수신 대기
- 1 - 로컬 클라이언트만 수신 대기

- 포트 - MySQL 이 TCP/IP 연결을 수신 대기하는 포트입니다.

2. mysqld 서비스를 다시 시작합니다.

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

3.3. MYSQL 서버에서 TLS 암호화 설정

기본적으로 MySQL 은 암호화되지 않은 연결을 사용합니다. 보안 연결의 경우 MySQL 서버에서 TLS 지원을 활성화하고 암호화된 연결을 설정하도록 클라이언트를 구성합니다.

3.3.1. MySQL 서버에 CA 인증서, 서버 인증서 및 개인 키 배치

MySQL 서버에서 TLS 암호화를 활성화하려면 먼저 CA(인증 기관) 인증서, 서버 인증서 및 MySQL 서버에 개인 키를 저장합니다.

사전 요구 사항

- Privacy Enhanced Mail (PEM) 형식의 다음 파일이 서버에 복사되었습니다.
  - 서버의 개인 키: server.example.com.key.pem
  - 서버 인증서: server.example.com.crt.pem
  - CA(인증 기관) 인증서: ca.crt.pem

개인 키 및 CSR(인증서 서명 요청) 생성 및 CA에서 인증서 요청에 대한 자세한 내용은 CA

문서를 참조하십시오.

절차

1. CA 및 서버 인증서를 /etc/pki/tls/certs/ 디렉터리에 저장합니다.

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

2. MySQL 서버가 파일을 읽을 수 있도록 하는 CA 및 서버 인증서에 대한 권한을 설정합니다.

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

인증서는 보안 연결을 설정하기 전에 통신의 일부이므로 모든 클라이언트는 인증 없이 검색할 수 있습니다. 따라서 CA 및 서버 인증서 파일에 대한 엄격한 권한을 설정할 필요가 없습니다.

3. 서버의 개인 키를 /etc/pki/tls/private/ 디렉터리에 저장합니다.

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

4. 서버의 개인 키에 보안 권한을 설정합니다.

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

권한이 없는 사용자가 개인 키에 액세스할 수 있는 경우 MySQL 서버에 대한 연결이 더 이상 안전하지 않습니다.

5. SELinux 컨텍스트를 복원하십시오.

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

3.3.2. MySQL 서버에서 TLS 구성

보안을 강화하려면 MySQL 서버에서 TLS 지원을 활성화합니다. 결과적으로 클라이언트는 TLS 암호화를 사용하여 서버로 데이터를 전송할 수 있습니다.

사전 요구 사항

- MySQL 서버를 설치했습니다.
- mysqld 서비스가 실행 중입니다.
- Privacy Enhanced Mail (PEM) 형식의 다음 파일은 서버에 있으며 mysql 사용자가 읽을 수 있습니다.
  - 서버의 개인 키: /etc/pki/tls/private/server.example.com.key.pem
  - 서버 인증서: /etc/pki/tls/certs/server.example.com.crt.pem
  - CA(인증 기관) 인증서 /etc/pki/tls/certs/ca.crt.pem
- 서버 인증서의 주체 고유 이름(DN) 또는 subject alternatives name(SAN) 필드는 서버의 호스트 이름과 일치합니다.

절차

1. /etc/my.cnf.d/mysql-server-tls.cnf 파일을 만듭니다.
   a. 다음 콘텐츠를 추가하여 개인 키, 서버 및 CA 인증서에 대한 경로를 구성합니다.

```
[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. CRL(Certificate Revocation List)이 있는 경우 이를 사용하도록 MySQL 서버를 구성합니다.

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

   c. 선택 사항: 암호화 없이 연결 시도를 거부합니다. 이 기능을 활성화하려면 다음을 추가

합니다.

```
require_secure_transport = on
```

d. 선택 사항: 서버가 지원해야 하는 TLS 버전을 설정합니다. 예를 들어 TLS 1.2 및 TLS 1.3을 지원하려면 다음을 추가합니다.

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

기본적으로 서버는 TLS 1.1, TLS 1.2 및 TLS 1.3을 지원합니다.

2. mysqld 서비스를 다시 시작합니다.

```
# systemctl restart mysqld
```

검증

문제 해결을 간소화하려면 TLS 암호화를 사용하도록 로컬 클라이언트를 구성하기 전에 MySQL 서버에서 다음 단계를 수행합니다.

1. 이제 MySQL 에 TLS 암호화가 활성화되어 있는지 확인합니다.

```
# 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. 특정 TLS 버전만 지원하도록 MySQL 서버를 구성한 경우 tls_version 변수를 표시합니다.

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

3. 서버가 올바른 CA 인증서, 서버 인증서 및 개인 키 파일을 사용하는지 확인합니다.

```
# 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 |
+---------------+-------------------------------------------+
```

추가 리소스

- MySQL 서버에 CA 인증서, 서버 인증서 및 개인 키 배치

3.3.3. 특정 사용자 계정에 대해 TLS 암호화 연결 필요

중요한 데이터에 액세스할 수 있는 사용자는 항상 TLS 암호화 연결을 사용하여 네트워크를 통해 암호화되지 않은 데이터를 전송하지 않도록 해야 합니다.

모든 연결(에서 required_secure_transport =)에 보안 전송이 필요한서버에서 를 구성할 수 없는 경우 TLS 암호화가 필요하도록 개별 사용자 계정을 구성합니다.

사전 요구 사항

- MySQL 서버에는 TLS 지원이 활성화되어 있습니다.
- 보안 전송을 요구하도록 구성하는 사용자가 있습니다.
- CA 인증서는 클라이언트에 저장됩니다.

절차

1. 관리 사용자로 MySQL 서버에 연결합니다.

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

관리 사용자에게 원격으로 서버에 액세스할 수 있는 권한이 없는 경우 MySQL 서버에서 명령을 수행하고 localhost 에 연결합니다.

2. **REQUIRE SSL** 절을 사용하여 사용자가 **TLS** 암호화 연결을 사용하여 연결해야 함을 적용합니다.

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

검증

1. **TLS** 암호화를 사용하여 서버에 예제 사용자로 연결합니다.

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

오류가 표시되지 않고 대화형 **MySQL** 콘솔에 액세스할 수 있는 경우 **TLS**와의 연결에 성공합니다.

기본적으로 클라이언트는 서버에서 제공하는 경우 **TLS** 암호화를 자동으로 사용합니다. 따라서 **--ssl-ca=ca.crt.pem** 및 **--ssl-mode=VERIFY_IDENTITY** 옵션은 필요하지 않지만 이러한 옵션을 사용하여 클라이언트는 서버 **ID**를 확인하므로 보안을 개선합니다.

2. **TLS**가 비활성화된 예제 사용자로 연결을 시도합니다.

```
# 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)
```

이 사용자에게는 **TLS**가 필요하지만 비활성화됨(**--ssl-mode=DISABLED**)이므로 서버에서 로그인 시도를 거부했습니다.

추가 리소스

- **MySQL** 서버에서 **TLS** 구성

3.4. MYSQL 클라이언트에서 CA 인증서 검증으로 TLS 암호화 글로벌 활성화

MySQL 서버가 **TLS** 암호화를 지원하는 경우 보안 연결만 설정하고 서버 인증서를 확인하도록 클라이언트를 구성합니다. 다음 절차에서는 서버에 있는 모든 사용자에 대해 **TLS** 지원을 활성화하는 방법을 설

명합니다.

3.4.1. 기본적으로 TLS 암호화를 사용하도록 MySQL 클라이언트 구성

RHEL에서 MySQL 클라이언트가 TLS 암호화를 사용하도록 전역적으로 구성하고 서버 인증서의 CN(일반 이름)이 사용자가 연결하는 호스트 이름과 일치하는지 확인할 수 있습니다. 따라서 메시지 가로채기(man-in-the-middle) 공격을 방지합니다.

사전 요구 사항

- MySQL 서버에는 TLS 지원이 활성화되어 있습니다.
- CA 인증서는 클라이언트의 /etc/pki/tls/certs/ca.crt.pem 파일에 저장됩니다.

절차

- 다음 콘텐츠를 사용하여 /etc/my.cnf.d/mysql-client-tls.cnf 파일을 만듭니다.

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

이러한 설정은 MySQL 클라이언트가 TLS 암호화를 사용하고 클라이언트가 서버 인증서의 CN(ssl-mode=VERIFY_IDENTITY)과 호스트 이름을 비교함을 정의합니다. 또한 CA 인증서의 경로를 지정합니다(ssl-ca).

검증

- 호스트 이름을 사용하여 서버에 연결하고 서버 상태를 표시합니다.

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

SSL 항목에 사용 중 암호화가 있는 경우..., 연결이 암호화됩니다.

이 명령에서 사용하는 사용자에게는 원격으로 인증할 수 있는 권한이 있습니다.

연결하는 호스트 이름이 서버의 **TLS** 인증서의 호스트 이름과 일치하지 않으면 **ssl-mode=VERIFY_IDENTITY** 매개변수로 인해 연결이 실패합니다. 예를 들어 **localhost** 에 연결하는 경우 :

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

추가 리소스

- **mysql(1)** 도움말 페이지의 **--ssl*** 매개 변수 설명.

3.5. MYSQL 데이터 백업

Red Hat Enterprise Linux 9의 **MySQL** 데이터베이스에서 데이터를 백업하는 방법에는 두 가지가 있습니다.

- 논리적 백업
- 물리적 백업

논리적 백업 은 데이터를 복원하는 데 필요한 **SQL** 문으로 구성됩니다. 이러한 유형의 백업은 일반 텍스트 파일로 정보와 레코드를 내보냅니다.

물리적 백업보다 논리적 백업의 주요 장점은 이식성과 유연성입니다. 데이터는 물리적 백업에서 사용할 수 없는 다른 하드웨어 구성, **MySQL** 버전 또는 **DBMS**(데이터베이스 관리 시스템)에서 복원할 수 있습니다.

mysqld.service 가 실행 중인 경우 논리적 백업을 수행할 수 있습니다. 논리적 백업에는 로그 및 구성 파일이 포함되지 않습니다.

물리적 백업 은 콘텐츠를 저장하는 파일과 디렉토리의 복사본으로 구성됩니다.

물리적 백업은 논리적 백업과 비교하여 다음과 같은 이점이 있습니다.

- 출력이 더 작습니다.
- 백업은 크기가 작습니다.
- 백업 및 복원 속도가 빨라집니다.
- 백업에는 로그 및 구성 파일이 포함됩니다.

mysqld.service 가 실행 중이 아니거나 백업 중 변경되지 않도록 데이터베이스의 모든 테이블이 잠길 때 물리적 백업을 수행해야 합니다.

다음 **MySQL** 백업 접근법 중 하나를 사용하여 **MySQL** 데이터베이스에서 데이터를 백업할 수 있습니다.

- **mysqldump**를 사용한 논리적 백업
- 파일 시스템 백업
- 백업 솔루션으로의 복제

3.5.1. mysqldump를 사용하여 논리적 백업 수행

mysqldump 클라이언트는 백업 또는 다른 데이터베이스 서버로 전송하기 위해 데이터베이스 또는 데이터베이스 컬렉션을 덤프하는 데 사용할 수 있는 백업 유틸리티입니다. **mysqldump** 의 출력은 일반적으로 서버 테이블 구조를 다시 만들거나, 데이터로 채우거나, 둘 다입니다. **mysqldump** 는 **XML** 및 **CSV**와 같은 구분된 텍스트 형식을 포함한 다른 형식으로 파일을 생성할 수도 있습니다.

mysqldump 백업을 수행하려면 다음 옵션 중 하나를 사용할 수 있습니다.

- 하나 이상의 선택된 데이터베이스 백업

- 모든 데이터베이스 백업
- 하나의 데이터베이스에서 테이블의 하위 집합 백업

절차

- 단일 데이터베이스를 덤프하려면 다음을 실행합니다.

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

- 한 번에 여러 데이터베이스를 덤프하려면 다음을 실행합니다.

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

- 모든 데이터베이스를 덤프하려면 다음을 실행합니다.

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

- 하나 이상의 덤프된 전체 데이터베이스를 서버에 다시 로드하려면 다음을 실행합니다.

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

- 데이터베이스를 원격 MySQL 서버에 로드하려면 다음을 실행합니다.

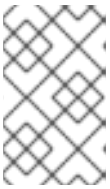
```
# mysql --host=remote_host < backup-file.sql
```

- 하나의 데이터베이스에서 리터럴 하위 집합을 덤프하려면 mysqldump 명령 끝에 선택한 테이블 목록을 추가합니다.

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

- 하나의 데이터베이스에서 덤프된 테이블의 리터럴 하위 집합을 로드하려면 다음을 실행합니다.

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



참고

db_name 데이터베이스는 이 시점에 있어야 합니다.

- **mysqldump** 에서 지원하는 옵션 목록을 보려면 다음을 실행합니다.

```
$ mysqldump --help
```

추가 리소스

- mysqldump를 사용한 논리적 백업

3.5.2. 파일 시스템 백업 수행

MySQL 데이터 파일의 파일 시스템 백업을 생성하려면 **MySQL** 데이터 디렉터리의 내용을 백업 위치에 복사합니다.

현재 구성 또는 로그 파일도 백업하려면 다음 절차의 선택적 단계를 사용하십시오.

절차

1. **mysqld** 서비스를 중지합니다.

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

2. 필요한 위치에 데이터 파일을 복사합니다.

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

3. 선택 사항: 구성 파일을 필요한 위치에 복사합니다.

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

4. 선택 사항: 로그 파일을 필요한 위치에 복사합니다.

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

5. **mysqld** 서비스를 시작합니다.

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

6. 백업 위치에서 **/var/lib/mysql** 디렉터리에 백업 데이터를 로드할 때 **mysql:mysql** 이 **/var/lib/mysql** 에 있는 모든 데이터의 소유자인지 확인하십시오.

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

3.5.3. 백업 솔루션으로의 복제

복제는 소스 서버를 위한 대체 백업 솔루션입니다. 소스 서버에서 복제본 서버에 복제하는 경우 소스에 영향을 주지 않고 복제본에서 백업을 실행할 수 있습니다. 복제본을 종료하는 동안 계속 소스를 실행하고 복제본에서 데이터를 백업할 수 있습니다.

MySQL 데이터베이스를 복제하는 방법에 대한 지침은 **MySQL** 복제를 참조하십시오.

주의

복제 자체는 충분한 백업 솔루션이 아닙니다. 복제는 하드웨어 장애로부터 소스 서버를 보호하지만 데이터 손실에 대한 보호는 보장하지 않습니다. 이 메서드와 함께 복제본에서 다른 백업 솔루션을 사용하는 것이 좋습니다.

추가 리소스

- **MySQL** 복제 문서

3.6. RHEL 9 버전의 MYSQL 8.0으로 마이그레이션

RHEL 8에는 **MySQL** 데이터베이스 제품군에서 서버의 **MySQL 8.0,MariaDB 10.3** 및 **MariaDB 10.5** 구현이 포함되어 있습니다. **RHEL 9**는 **MySQL 8.0** 및 **MariaDB 10.5** 를 제공합니다.

다음 절차에서는 **mysql_upgrade** 유틸리티를 사용하여 **RHEL 8** 버전의 **MySQL 8.0** 에서 **RHEL 9** 버전의 **MySQL 8.0** 으로 마이그레이션하는 방법을 설명합니다. **mysql_upgrade** 유틸리티는 **mysql-server** 패키지에서 제공합니다.

사전 요구 사항

- 업그레이드를 수행하기 전에 **MySQL** 데이터베이스에 저장된 모든 데이터를 백업합니다. **MySQL 데이터 백업을** 참조하십시오.

절차

1. **mysql-server** 패키지가 **RHEL 9** 시스템에 설치되었는지 확인합니다.

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

2. 데이터를 복사할 때 **mysqld** 서비스가 소스 및 대상 시스템에서 실행되고 있지 않은지 확인합니다.

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

3. 소스 위치의 데이터를 **RHEL 9** 대상 시스템의 **/var/lib/mysql/** 디렉터리에 복사합니다.

4. 대상 시스템에서 복사된 파일에 대한 적절한 권한 및 **SELinux** 컨텍스트를 설정합니다.

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

5. **mysql:mysql** 이 **/var/lib/mysql** 디렉터리에 있는 모든 데이터의 소유자인지 확인합니다.

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

6. 대상 시스템에서 **MySQL** 서버를 시작합니다.

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

참고: 이전 버전의 **MySQL** 에서는 내부 테이블을 확인하고 복구하는 데 **mysql_upgrade** 명령이 필요했습니다. 이제 서버를 시작할 때 자동으로 수행됩니다.

3.7. MYSQL 복제

MySQL 은 기본에서 고급까지 복제를 위한 다양한 구성 옵션을 제공합니다. 이 섹션에서는 GTID(Global transaction Identifiers)를 사용하여 새로 설치된 MySQL 서버에서 MySQL에서 복제하는 트랜잭션 기반 방법에 대해 설명합니다. GTID를 사용하면 트랜잭션 식별 및 일관성 확인이 간소화됩니다.

MySQL 에서 복제를 설정하려면 다음을 수행해야 합니다.

- 소스 서버 구성
- 복제본 서버 구성
- 소스 서버에서 복제 사용자 생성
- 소스 서버에 복제본 서버 연결

중요

복제에 기존 MySQL 서버를 사용하려면 먼저 데이터를 동기화해야 합니다. 자세한 내용은 업스트림 문서 를 참조하십시오.

3.7.1. MySQL 소스 서버 구성

MySQL 소스 서버에 필요한 구성 옵션을 설정하여 데이터베이스 서버에서 수행된 모든 변경 사항을 올바르게 실행하고 복제할 수 있습니다.

사전 요구 사항

- 소스 서버가 설치되어 있어야 합니다.

절차

1. [mysqld] 섹션의 /etc/my.cnf.d/mysql-server.cnf 파일에 다음 옵션을 포함합니다.

- **bind-address=*source_ip_adress***

  이 옵션은 복제본에서 소스로 구성된 연결에 필요합니다.

- **server-id=*id***

  id 는 고유해야 합니다.

- **log_bin=*path_to_source_server_log***

  이 옵션은 **MySQL** 소스 서버의 바이너리 로그 파일의 경로를 정의합니다. 예: **log_bin=/var/log/mysql/mysql-bin.log.**

- **gtid_mode=ON**

  이 옵션을 사용하면 서버에서 글로벌 트랜잭션 식별자(**GTID**)를 사용할 수 있습니다.

- **enforce-gtid-consistency=ON**

  서버는 **GTID**를 사용하여 안전하게 로깅할 수 있는 문만 실행할 수 있도록 하여 **GTID** 일관성을 적용합니다.

- ***Optional:*** **binlog_do_db=*db_name***

  선택한 데이터베이스만 복제하려면 이 옵션을 사용합니다. 두 개 이상의 선택된 데이터베이스를 복제하려면 각 데이터베이스를 별도로 지정합니다.

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

- ***Optional:*** **binlog_ignore_db=*db_name***

  복제에서 특정 데이터베이스를 제외하려면 이 옵션을 사용합니다.

2. **mysqld** 서비스를 다시 시작합니다.

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

3.7.2. MySQL 복제본 서버 구성

MySQL 복제본 서버에 필요한 구성 옵션을 설정하여 복제를 성공적으로 수행할 수 있습니다.

사전 요구 사항

- 복제본 서버가 설치되어 있어야 합니다.

절차

1. **[mysqld]** 섹션의 **/etc/my.cnf.d/mysql-server.cnf** 파일에 다음 옵션을 포함합니다.

   - **server-id=*id***

     id 는 고유해야 합니다.

   - **relay-log=*path_to_replica_server_log***

     릴레이 로그는 복제 중에 **MySQL** 복제본 서버에서 생성한 로그 파일 집합입니다.

   - **log_bin=*path_to_replica_sever_log***

     이 옵션은 **MySQL** 복제본 서버의 바이너리 로그 파일에 대한 경로를 정의합니다. 예: **log_bin=/var/log/mysql/mysql-bin.log**.

     이 옵션은 복제본에 필요하지 않지만 강력히 권장됩니다.

   - **gtid_mode=ON**

이 옵션을 사용하면 서버에서 글로벌 트랜잭션 식별자(GTID)를 사용할 수 있습니다.

- **enforce-gtid-consistency=ON**

서버는 GTID를 사용하여 안전하게 로깅할 수 있는 문만 실행할 수 있도록 하여 GTID 일관성을 적용합니다.

- **log-replica-updates=ON**

이 옵션을 사용하면 소스 서버에서 수신한 업데이트가 복제본의 바이너리 로그에 기록됩니다.

- **skip-replica-start=ON**

이 옵션을 사용하면 복제본 서버가 복제 서버가 시작될 때 복제 스레드를 시작하지 않습니다.

- ***Optional:*** **binlog_do_db=*db_name***

특정 데이터베이스만 복제하려면 이 옵션을 사용합니다. 두 개 이상의 데이터베이스를 복제하려면 각 데이터베이스를 별도로 지정합니다.

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

- ***Optional:*** **binlog_ignore_db=*db_name***

복제에서 특정 데이터베이스를 제외하려면 이 옵션을 사용합니다.

2. **mysqld** 서비스를 다시 시작합니다.

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

3.7.3. MySQL 소스 서버에서 복제 사용자 생성

복제 사용자를 생성하고 복제 트래픽에 필요한 이 사용자 권한을 부여해야 합니다. 다음 절차에서는 적절한 권한이 있는 복제 사용자를 만드는 방법을 설명합니다. 소스 서버에서만 이 단계를 실행합니다.

사전 요구 사항

- MySQL 소스 서버 구성에 설명된 대로 소스 서버가 설치되고 구성됩니다.

절차

1. 복제 사용자를 생성합니다.

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

2. 사용자 복제 권한을 부여합니다.

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

3. MySQL 데이터베이스에서 부여 테이블을 다시 로드합니다.

```
mysql> FLUSH PRIVILEGES;
```

4. 소스 서버를 읽기 전용 상태로 설정합니다.

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

3.7.4. 소스 서버에 복제본 서버 연결

MySQL 복제본 서버에서는 소스 서버의 자격 증명과 주소를 구성해야 합니다. 다음 절차에 따라 복제본 서버를 구현합니다.

사전 요구 사항

- MySQL 소스 서버 구성에 설명된 대로 소스 서버가 설치되고 구성됩니다.

- MySQL 복제본 서버 구성에 설명된 대로 복제본 서버가 설치되고 구성됩니다.
- 복제 사용자를 생성했습니다. MySQL 소스 서버에서 복제 사용자 생성을 참조하십시오.

절차

1. 복제본 서버를 읽기 전용 상태로 설정합니다.

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

2. 복제 소스를 구성합니다.

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

3. MySQL 복제본 서버에서 복제본 스레드를 시작합니다.

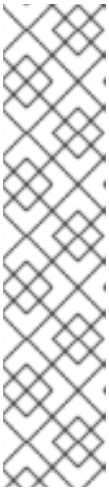
```
mysql> START REPLICA;
```

4. 소스 및 복제본 서버 둘 다에서 읽기 전용 상태를 설정 해제합니다.

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

5. 선택 사항: 디버깅 목적으로 복제본 서버의 상태를 검사합니다.

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



참고

복제본 서버가 시작하거나 연결하는 데 실패하면 **SHOW MASTER STATUS** 명령의 출력에 표시된 바이너리 로그 파일 위치에 따라 특정 이벤트 수를 건너뛸 수 있습니다. 예를 들어 정의된 위치에서 첫 번째 이벤트를 건너뛰십시오.

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

복제 서버를 다시 시작합니다.

6. 선택 사항: 복제본 서버에서 복제본 스레드를 중지합니다.

```
mysql> STOP REPLICA;
```

3.7.5. 검증

1. 소스 서버에서 예제 데이터베이스를 생성합니다.

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

2. ***test_db_name*** 데이터베이스가 복제본 서버에 복제되는지 확인합니다.

3. 소스 또는 복제본 서버에서 다음 명령을 실행하여 **MySQL** 서버의 바이너리 로그 파일에 대한 상태 정보를 표시합니다.

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

소스에서 실행되는 트랜잭션에 대한 **GTID** 집합을 표시하는 **Executed_Gtid_Set** 열은 비어 있지 않아야 합니다.

참고

복제본 서버에서 **SHOW SLAVE STATUS** 를 사용할 때 동일한 **GTID** 세트가 **Executed_Gtid_Set** 행에 표시됩니다.

3.7.6. 추가 리소스

- MySQL 복제 문서
- MySQL에서 복제 설정 방법
- 글로벌 트랜잭션 식별자로 복제

3.8. MYSQL 클라이언트 애플리케이션 개발

Red Hat은 MariaDB 클라이언트 라이브러리에 대해 MySQL 클라이언트 애플리케이션을 개발하는 것이 좋습니다. 클라이언트와 서버 간의 통신 프로토콜은 MariaDB 와 MySQL 간에 호환됩니다. MariaDB 클라이언트 라이브러리는 MySQL 구현과 관련된 제한된 수의 기능을 제외하고 대부분의 일반적인 MySQL 시나리오에서 작동합니다.

MariaDB 클라이언트 라이브러리에 대해 애플리케이션을 빌드하는 데 필요한 개발 파일 및 프로그램은 mariadb-connector-c-devel 패키지에서 제공합니다.

직접 라이브러리 이름을 사용하는 대신 mariadb-connector-c-devel 패키지에 배포되는 mariadb_config 프로그램을 사용합니다. 이 프로그램을 통해 올바른 빌드 플래그가 반환됩니다.

4장. POSTGRESQL 사용

PostgreSQL 서버는 SQL 언어를 기반으로 하는 강력하고 확장성이 뛰어난 오픈 소스 데이터베이스 서버입니다. PostgreSQL 서버는 광범위한 데이터 세트 및 다수의 동시 사용자를 관리할 수 있는 개체 관계형 데이터베이스 시스템을 제공합니다. 이러한 이유로 클러스터에서 PostgreSQL 서버를 사용하여 대량의 데이터를 관리할 수 있습니다.

PostgreSQL 서버에는 데이터 무결성을 보장하기 위한 기능이 포함되어 있어 내결함성 환경 및 애플리케이션을 구축할 수 있습니다. PostgreSQL 서버를 사용하면 데이터베이스를 다시 컴파일하지 않고도 자체 데이터 유형, 사용자 지정 함수 또는 다른 프로그래밍 언어의 코드를 사용하여 데이터베이스를 확장할 수 있습니다.

RHEL 시스템에 PostgreSQL 을 설치하고 구성하는 방법, PostgreSQL 데이터를 백업하는 방법 및 이전 PostgreSQL 버전에서 마이그레이션하는 방법을 알아봅니다.

4.1. POSTGRESQL 설치

RHEL 9에서는 PostgreSQL 13 을 이 Application Stream의 초기 버전으로 제공하며 RPM 패키지로 쉽게 설치할 수 있습니다.

추가 PostgreSQL 버전은 RHEL 9의 마이너 릴리스에서 라이프 사이클이 짧은 모듈로 제공됩니다.

- RHEL 9.2에서 PostgreSQL 15 를 postgresql:15 모듈 스트림으로 도입
- RHEL 9.4에서는 PostgreSQL 16 을 postgresql:16 모듈 스트림으로 도입

PostgreSQL 을 설치하려면 다음 절차를 사용합니다.

참고

설계상 동일한 모듈의 두 개 이상 버전(스트림)을 병렬로 설치할 수 없습니다. 따라서 postgresql 모듈에서 사용 가능한 스트림 중 하나만 선택해야 합니다. 컨테이너에서 다른 버전의 PostgreSQL 데이터베이스 서버를 사용할 수 있습니다. 컨테이너에서 여러 PostgreSQL 버전 실행을 참조하십시오.

절차

1. PostgreSQL 서버 패키지를 설치합니다.

   - RPM 패키지의 PostgreSQL 13 의 경우:

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

   - postgresql 모듈에서 스트림(버전) 15 또는 16을 선택하고 서버 프로필을 지정하여 PostgreSQL 15 또는 PostgreSQL 16 의 경우 다음과 같습니다.

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

   postgres 수퍼유저가 자동으로 생성됩니다.

2. 데이터베이스 클러스터를 초기화합니다.

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

   Red Hat은 기본 /var/lib/pgsql/data 디렉터리에 데이터를 저장하는 것이 좋습니다.

3. postgresql 서비스를 시작합니다.

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

4. 부팅 시 시작되도록 postgresql 서비스를 활성화합니다.

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

중요

RHEL 9에서 이전 postgresql 스트림에서 업그레이드하려면 이후 스트림으로 전환하고 RHEL 9 버전의 PostgreSQL으로 마이그레이션에 설명된 두 가지 절차를 따르십시오.

4.1.1. 컨테이너에서 여러 PostgreSQL 버전 실행

동일한 호스트에서 다른 버전의 PostgreSQL 을 실행하려면 동일한 모듈의 여러 버전(스트림)을 병렬

로 설치할 수 없기 때문에 컨테이너에서 실행합니다.

이 절차에는 **PostgreSQL 13** 및 **PostgreSQL 15** 가 포함되어 있지만 **Red Hat Ecosystem Catalog**에서 사용할 수 있는 **PostgreSQL** 컨테이너 버전을 사용할 수 있습니다.

사전 요구 사항

- **container-tools meta-package**가 설치되어 있습니다.

절차

1. **Red Hat** 고객 포털 계정을 사용하여 **registry.redhat.io** 레지스트리에 인증합니다.

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

컨테이너 레지스트리에 이미 로그인한 경우 이 단계를 건너뜁니다.

2. 컨테이너에서 **PostgreSQL 13** 을 실행합니다.

```
$ 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
```

이 컨테이너 이미지 사용에 대한 자세한 내용은 Red Hat Ecosystem Catalog 를 참조하십시오.

3. 컨테이너에서 **PostgreSQL 15** 를 실행합니다.

```
$ 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
```

이 컨테이너 이미지 사용에 대한 자세한 내용은 Red Hat Ecosystem Catalog 를 참조하십시오.

4. 컨테이너에서 **PostgreSQL 16** 을 실행합니다.

```
$ 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
```

이 컨테이너 이미지 사용에 대한 자세한 내용은 Red Hat Ecosystem Catalog 를 참조하십시오.

참고

두 데이터베이스 서버의 컨테이너 이름과 호스트 포트는 달라야 합니다.

5. 클라이언트가 네트워크의 데이터베이스 서버에 액세스할 수 있도록 하려면 방화벽에서 호스트 포트를 엽니다.

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

검증

1. 실행 중인 컨테이너에 대한 정보를 표시합니다.

```
$ podman ps
```

2. 데이터베이스 서버에 연결하고 **root**로 로그인합니다.

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

추가 리소스

- 컨테이너 빌드, 실행 및 관리
- Red Hat Ecosystem Catalog에서 컨테이너 검색

4.2. POSTGRESQL 사용자 생성

PostgreSQL 사용자는 다음과 같은 유형입니다.

- postgres UNIX 시스템 사용자 - pg_dump 와 같은 PostgreSQL 서버 및 클라이언트 애플리케이션을 실행하는 데만 사용해야 합니다. 데이터베이스 생성 및 사용자 관리와 같은 PostgreSQL 관리에서는 postgres 시스템 사용자를 사용하지 마십시오.

- 데이터베이스 슈퍼유저 - 기본 postgres PostgreSQL 슈퍼유저는 postgres 시스템 사용자와 관련이 없습니다. pg_hba.conf 파일에서 postgres 슈퍼 유저에 대한 액세스를 제한할 수 있고, 그렇지 않으면 다른 권한 제한은 없습니다. 다른 데이터베이스 수퍼유저도 만들 수 있습니다.

- 특정 데이터베이스 액세스 권한이 있는 역할:

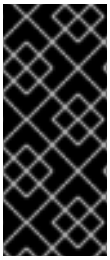
  - 데이터베이스 사용자 - 기본적으로 로그인할 수 있는 권한이 있습니다.

  - 사용자 그룹 - 그룹 전체에 대한 권한 관리를 활성화합니다.

역할은 데이터베이스 개체(예: 테이블 및 함수)를 소유하고 SQL 명령을 사용하여 다른 역할에 개체 권한을 할당할 수 있습니다.

표준 데이터베이스 관리 권한에는 SELECT,INSERT,UPDATE,DELETE,TRUNCATE,REFERENCES,TRIGGER,CREATE,CONNECT,TEMPORARY, EXECUTE, USAGE 등이 있습니다.

역할 속성은 LOGIN,SUPERUSER,CREATEDB 및 CREATE ROLE 과 같은 특수 권한입니다.



중요

대부분의 작업을 수퍼유저가 아닌 역할로 수행하는 것이 좋습니다. 일반적인 방법은 CREATEDB 및 CREATE ROLE 권한이 있는 역할을 생성하고 이 역할을 데이터베이스 및 역할의 모든 일상적인 관리에 사용하는 것입니다.

사전 요구 사항

- PostgreSQL 서버가 설치되어 있어야 합니다.
- 데이터베이스 클러스터가 초기화됩니다.

절차

- 사용자를 만들려면 사용자의 암호를 설정하고 CREATEROLE 및 CREATEDB 권한을 사용자에게 할당합니다.

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

mydbuser 를 사용자 이름으로 바꾸고 *mypasswd* 를 사용자 암호로 교체합니다.

추가 리소스

- PostgreSQL 데이터베이스 역할
- PostgreSQL 권한
- PostgreSQL 구성

예 4.1. PostgreSQL 데이터베이스 초기화, 생성 및 연결

이 예제에서는 PostgreSQL 데이터베이스를 초기화하고 일상적인 데이터베이스 관리 권한이 있는 데이터베이스 사용자를 만드는 방법, 관리 권한이 있는 데이터베이스 사용자를 통해 모든 시스템 계정에서 액세스할 수 있는 데이터베이스를 만드는 방법을 보여줍니다.

1. PosgreSQL 서버를 설치합니다.

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

2. 데이터베이스 클러스터를 초기화합니다.

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

3. 암호 해시 알고리즘을 **scram-sha-256** 로 설정합니다.

   a. **/var/lib/pgsql/data/postgresql.conf** 파일에서 다음 행을 변경합니다.

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

   다음으로 변경합니다.

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

   b. **/var/lib/pgsql/data/pg_hba.conf** 파일에서 **IPv4** 로컬 연결에 대해 다음 행을 변경합니다.

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

   다음으로 변경합니다.

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

4. **postgresql** 서비스를 시작합니다.

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

5. **postgres** 라는 시스템 사용자로 로그인합니다.

```
# su - postgres
```

6. **PostgreSQL** 대화형 터미널을 시작합니다.

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

```
postgres=#
```

7. 선택 사항: 현재 데이터베이스 연결에 대한 정보를 가져옵니다.

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

8. mydbuser 라는 사용자를 만들고, mydbuser 의 암호를 설정하고, mydbuser 에 CREATEROLE 및 CREATEDB 권한을 할당합니다.

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

이제 mydbuser 사용자는 일상적인 데이터베이스 관리 작업을 수행할 수 있습니다: 데이터베이스를 생성하고 사용자 인덱스를 관리합니다.

9. \q meta 명령을 사용하여 대화형 터미널에서 로그아웃합니다.

```
postgres=# \q
```

10. postgres 사용자 세션에서 로그아웃합니다.

```
$ logout
```

11. PostgreSQL 터미널에 mydbuser 로 로그인하고 호스트 이름을 지정하고 초기화 중에 생성된 기본 postgres 데이터베이스에 연결합니다.

```
# 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 라는 데이터베이스를 만듭니다.

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

13. 세션에서 로그아웃합니다.

```
postgres=# \q
```

14. mydbuser 로 mydatabase에 연결합니다.

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

15. 선택 사항: 현재 데이터베이스 연결에 대한 정보를 가져옵니다.

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

4.3. POSTGRESQL 구성

PostgreSQL 데이터베이스에서 모든 데이터 및 구성 파일은 데이터베이스 클러스터라는 단일 디렉터리에 저장됩니다. Red Hat은 기본 /var/lib/pgsql/data/ 디렉토리에 설정 파일을 포함한 모든 데이터를 저장할 것을 권장합니다.

PostgreSQL 구성은 다음 파일로 구성됩니다.

- PostgreSQL.conf - 데이터베이스 클러스터 매개 변수를 설정하는 데 사용됩니다.
- PostgreSQL .auto.conf - postgresql. conf 와 유사한 기본 PostgreSQL 설정을 보유합니다. 그러나 이 파일은 서버 제어 아래에 있습니다. 이 파일은SYSTEM 쿼리에 의해 편집되며 수동으로 편집할 수 없습니다.
- pg_ident.conf - 외부 인증 메커니즘의 사용자 ID를 PostgreSQL 사용자 ID로 매핑하는 데 사

용됩니다.

- pg_hba.conf - PostgreSQL 데이터베이스의 클라이언트 인증을 구성하는 데 사용됩니다.

PostgreSQL 구성을 변경하려면 다음 절차를 사용하십시오.

절차

1. 해당 구성 파일(예: /var/lib/pgsql/data/postgresql.conf)을 편집합니다.
2. 변경 사항이 적용되도록 postgresql 서비스를 다시 시작합니다.

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

예 4.2. PostgreSQL 데이터베이스 클러스터 매개변수 구성

이 예제에서는 /var/lib/pgsql/data/postgresql.conf 파일에서 데이터베이스 클러스터 매개변수의 기본 설정을 보여줍니다.

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

예 4.3. PostgreSQL에서 클라이언트 인증 설정

이 예제에서는 /var/lib/pgsql/data/pg_hba.conf 파일에서 클라이언트 인증을 설정하는 방법을 보여줍니다.

```
# 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 서버에서 TLS 암호화 구성

기본적으로 PostgreSQL 은 암호화되지 않은 연결을 사용합니다. 보다 안전한 연결의 경우 PostgreSQL 서버에서 TLS(Transport Layer Security) 지원을 활성화하고 암호화된 연결을 설정하도록 클라이언트를 구성할 수 있습니다.

사전 요구 사항

- PostgreSQL 서버가 설치되어 있어야 합니다.
- 데이터베이스 클러스터가 초기화됩니다.
- 서버가 RHEL 9.2 이상을 실행하고 FIPS 모드가 활성화된 경우 클라이언트는 확장 마스터 시크릿(Extended Master Secret) 확장을 지원하거나 TLS 1.3을 사용해야 합니다. TLS 1.2 연결이 없는 경우 실패합니다. 자세한 내용은 RHEL 9.2 이상에 적용된 Red Hat Knowledgebase 솔루션 TLS 확장 "확장 마스터 시크릿"을 참조하십시오.

절차

1. OpenSSL 라이브러리를 설치합니다.

```
# dnf install openssl
```

2. TLS 인증서 및 키를 생성합니다.

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

*dbhost.yourdomain.com*을 데이터베이스 호스트 및 도메인 이름으로 교체합니다.

3. 서명된 인증서와 개인 키를 데이터베이스 서버의 필수 위치에 복사합니다.

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

4. 서명된 인증서와 개인 키의 소유자 및 그룹 소유권을 postgres 사용자로 변경합니다.

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

5. 소유자만 읽을 수 있도록 개인 키에 대한 권한을 제한합니다.

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

6. /var/lib/pgsql/data/postgresql.conf 파일에서 다음 행을 변경하여 암호 해시 알고리즘을 scram-sha-256 으로 설정합니다.

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

다음으로 변경합니다.

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

7. /var/lib/pgsql/data/postgresql.conf 파일에서 다음 행을 변경하여 SSL/TLS를 사용하도록 PostgreSQL을 구성합니다.

```
#ssl = off
```

다음으로 변경합니다.

```
ssl=on
```

8. /var/lib/pgsql/data/pg_hba.conf 파일에서 IPv4 로컬 연결에 대해 다음 행을 변경하여 TLS를 사용하는 클라이언트의 연결만 허용하도록 모든 데이터베이스에 대한 액세스를 제한합니다.

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

다음으로 변경합니다.

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

또는 다음 새 행을 추가하여 단일 데이터베이스 및 사용자에 대한 액세스를 제한할 수 있습니다.

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

mydatabase 를 데이터베이스 이름으로 바꾸고 *mydbuser* 를 사용자 이름으로 바꿉니다.

9. **postgresql** 서비스를 다시 시작하여 변경 사항을 적용합니다.

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

검증

- 연결이 암호화되었는지 수동으로 확인하려면 다음을 수행하십시오.

1. **PostgreSQL** 데이터베이스에 *mydbuser* 사용자로 연결하고 호스트 이름과 데이터베이스 이름을 지정합니다.

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

mydatabase 를 데이터베이스 이름으로 바꾸고 *mydbuser* 를 사용자 이름으로 바꿉니다.

2. 현재 데이터베이스 연결에 대한 정보를 가져옵니다.

```
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** 에 대한 연결이 암호화되었는지 여부를 확인하는 간단한 애플리케이션을 작성할 수 있습니다. 이 예제에서는 **libpq-devel** 패키지에서 제공하는 **libpq** 클라이언트 라이브러리를 사용하는 **C**로 작성된 애플리케이션을 보여줍니다.

```
#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 를 암호, ***mydatabase*** 를 데이터베이스 이름으로, ***mydbuser*** 를 사용자 이름으로 바꿉니다.

참고

-l pq 옵션을 사용하여 컴파일할 **pq** 라이브러리를 로드해야 합니다. 예를 들어 **GCC** 컴파일러를 사용하여 애플리케이션을 컴파일하려면 다음을 수행합니다.

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

여기서 ***source_file.c*** 에는 위의 예제 코드가 포함되어 있으며 ***myapplication*** 은 보안 **PostgreSQL** 연결을 확인하는 애플리케이션의 이름입니다.

예 4.4. TLS 암호화를 사용하여 PostgreSQL 데이터베이스 초기화, 생성 및 연결

이 예제에서는 **PostgreSQL** 데이터베이스를 초기화하고, 데이터베이스 사용자와 데이터베이스를 생성하며, 보안 연결을 사용하여 데이터베이스에 연결하는 방법을 보여줍니다.

1. **PosgreSQL** 서버를 설치합니다.

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

2. 데이터베이스 클러스터를 초기화합니다.

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

3. OpenSSL 라이브러리를 설치합니다.

```
# dnf install openssl
```

4. TLS 인증서 및 키를 생성합니다.

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

*dbhost.yourdomain.com*을 데이터베이스 호스트 및 도메인 이름으로 교체합니다.

5. 서명된 인증서와 개인 키를 데이터베이스 서버의 필수 위치에 복사합니다.

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

6. 서명된 인증서와 개인 키의 소유자 및 그룹 소유권을 **postgres** 사용자로 변경합니다.

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

7. 소유자만 읽을 수 있도록 개인 키에 대한 권한을 제한합니다.

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

8. 암호 해시 알고리즘을 **scram-sha-256** 로 설정합니다.
/var/lib/pgsql/data/postgresql.conf 파일에서 다음 행을 변경합니다.

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

다음으로 변경합니다.

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

9.

SSL/TLS를 사용하도록 PostgreSQL을 구성합니다.
/var/lib/pgsql/data/postgresql.conf 파일에서 다음 행을 변경합니다.

```
#ssl = off
```

다음으로 변경합니다.

```
ssl=on
```

10. postgresql 서비스를 시작합니다.

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

11. postgres 라는 시스템 사용자로 로그인합니다.

```
# su - postgres
```

12. postgres 사용자로 PostgreSQL 대화형 터미널을 시작합니다.

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

postgres=#
```

13. mydbuser 라는 사용자를 생성하고 mydbuser 의 암호를 설정합니다.

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

14. mydatabase 라는 데이터베이스를 만듭니다.

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

15. mydbuser 사용자에게 모든 권한을 부여합니다.

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

16. 대화형 터미널에서 로그아웃합니다.

```
postgres=# \q
```

17. postgres 사용자 세션에서 로그아웃합니다.

```
$ logout
```

18. /var/lib/pgsql/data/pg_hba.conf 파일에서 IPv4 로컬 연결에 대해 다음 행을 변경하여 TLS를 사용하는 클라이언트의 연결만 허용하도록 모든 데이터베이스에 대한 액세스를 제한합니다.

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

다음으로 변경합니다.

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

19. postgresql 서비스를 다시 시작하여 변경 사항을 적용합니다.

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

20. PostgreSQL 데이터베이스에 mydbuser 사용자로 연결하고 호스트 이름과 데이터베이스 이름을 지정합니다.

```
$ 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 데이터 백업

PostgreSQL 데이터를 백업하려면 다음 방법 중 하나를 사용합니다.

SQL dump

SQL 덤프로 백업 을 참조하십시오.

파일 시스템 수준 백업

파일 시스템 수준 백업을 참조하십시오.

연속 보관

연속 보관을 참조하십시오.

4.5.1. SQL 덤프를 사용하여 PostgreSQL 데이터 백업

SQL 덤프 방법은 SQL 명령으로 덤프 파일을 생성하는 방법을 기반으로 합니다. 덤프가 데이터베이스 서버에 다시 업로드되면 덤프 시와 동일한 상태로 데이터베이스를 다시 생성합니다.

SQL 덤프는 다음 PostgreSQL 클라이언트 애플리케이션에서 확인합니다.

- pg_dump 는 역할 또는 테이블 공간에 대한 클러스터 전체 정보 없이 단일 데이터베이스를 덤프합니다.
- pg_dumpall 은 지정된 클러스터의 각 데이터베이스를 덤프하고 역할 및 테이블 공간 정의와 같은 클러스터 전체 데이터를 유지합니다.

기본적으로 pg_dump 및 pg_dumpall 명령은 결과를 표준 출력에 씁니다. 덤프를 파일에 저장하려면 출력을 SQL 파일로 리디렉션합니다. 결과 SQL 파일은 병렬 처리를 허용하는 텍스트 형식 또는 다른 형식으로 되어 있고 개체 복원을 보다 자세히 제어할 수 있습니다.

데이터베이스에 액세스할 수 있는 모든 원격 호스트에서 SQL 덤프를 수행할 수 있습니다.

4.5.1.1. SQL 덤프의 장점 및 단점

SQL 덤프에는 다른 PostgreSQL 백업 방법과 비교하여 다음과 같은 이점이 있습니다.

- SQL 덤프는 서버 버전별이 아닌 유일한 PostgreSQL 백업 방법입니다. pg_dump 유틸리티의 출력은 파일 시스템 수준 백업 또는 연속 아카이브에서는 사용할 수 없는 최신 버전의 PostgreSQL 로 다시 로드할 수 있습니다.

- SQL 덤프는 32비트에서 64비트 서버로 이동하는 등 데이터베이스를 다른 시스템 아키텍처로 전송할 때 작동하는 유일한 방법입니다.

- SQL 덤프는 내부적으로 일관된 덤프를 제공합니다. 덤프는 pg_dump 가 실행을 시작한 시점에 데이터베이스의 스냅샷을 나타냅니다.

- pg_dump 유틸리티는 실행 시 데이터베이스의 다른 작업을 차단하지 않습니다.

SQL 덤프의 단점은 파일 시스템 수준 백업에 비해 더 많은 시간이 필요하다는 것입니다.

4.5.1.2. pg_dump를 사용하여 SQL 덤프 수행

클러스터 전체 정보 없이 단일 데이터베이스를 덤프하려면 pg_dump 유틸리티를 사용합니다.

사전 요구 사항

- 덤프하려는 모든 테이블에 대한 읽기 액세스 권한이 있어야 합니다. 전체 데이터베이스를 덤프하려면 postgres superuser 또는 데이터베이스 관리자 권한이 있는 사용자로 명령을 실행해야 합니다.

절차

- 클러스터 전체 정보가 없는 데이터베이스를 덤프합니다.

```
$ pg_dump dbname > dumpfile
```

연결할 데이터베이스 서버 pg_dump 를 지정하려면 다음 명령줄 옵션을 사용합니다.

- 호스트를 정의하는 -h 옵션.

기본 호스트는 로컬 호스트이거나 **PPP HOST** 환경 변수에 의해 지정된 항목입니다.

- 포트를 정의하는 **-p** 옵션.

  기본 포트는 **PPP PORT** 환경 변수 또는 컴파일된 기본값에 의해 표시됩니다.

4.5.1.3. pg_dumpall을 사용하여 SQL 덤프 수행

지정된 데이터베이스 클러스터에서 각 데이터베이스를 덤프하고 클러스터 전체 데이터를 보존하려면 **pg_dumpall** 유틸리티를 사용합니다.

사전 요구 사항

- **postgres** 수퍼유저 또는 데이터베이스 관리자 권한이 있는 사용자로 명령을 실행해야 합니다.

절차

- 데이터베이스 클러스터의 모든 데이터베이스를 덤프하고 클러스터 전체 데이터를 보존합니다.

```
$ pg_dumpall > dumpfile
```

연결할 데이터베이스 서버 **pg_dumpall** 을 지정하려면 다음 명령줄 옵션을 사용합니다.

- 호스트를 정의하는 **-h** 옵션.

  기본 호스트는 로컬 호스트이거나 **PPP HOST** 환경 변수에 의해 지정된 항목입니다.

- 포트를 정의하는 **-p** 옵션.

  기본 포트는 **PPP PORT** 환경 변수 또는 컴파일된 기본값에 의해 표시됩니다.

- 기본 데이터베이스를 정의하는 **-l** 옵션.

이 옵션을 사용하면 초기화 중에 자동으로 생성된 **postgres** 데이터베이스와 다른 기본 데이터베이스를 선택할 수 있습니다.

4.5.1.4. pg_dump를 사용하여 덤프된 데이터베이스 복원

pg_dump 유틸리티를 사용하여 덤프한 **SQL** 덤프에서 데이터베이스를 복원하려면 아래 단계를 수행하십시오.

사전 요구 사항

- **postgres** 수퍼유저 또는 데이터베이스 관리자 권한이 있는 사용자로 명령을 실행해야 합니다.

절차

1. 새 데이터베이스를 생성합니다.

```
$ createdb dbname
```

2. 덤프된 데이터베이스의 오브젝트에 대한 권한을 소유하고 있거나 권한이 부여된 모든 사용자에게 이미 있는지 확인합니다. 이러한 사용자가 없는 경우 복원에서 원래 소유권 및 권한을 가진 오브젝트를 다시 생성할 수 없습니다.

3. **psql** 유틸리티를 실행하여 **pg_dump** 유틸리티에서 생성한 텍스트 파일 덤프를 복원합니다.

```
$ psql dbname < dumpfile
```

여기에서 **dumpfile** 은 **pg_dump** 명령의 출력입니다. 텍스트가 아닌 파일 덤프를 복원하려면 대신 **pg_restore** 유틸리티를 사용합니다.

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

4.5.1.5. pg_dumpall을 사용하여 덤프된 데이터베이스 복원

pg_dumpall 유틸리티를 사용하여 덤프한 데이터베이스 클러스터에서 데이터를 복원하려면 아래 단계를 따르십시오.

사전 요구 사항

- postgres 수퍼유저 또는 데이터베이스 관리자 권한이 있는 사용자로 명령을 실행해야 합니다.

절차

1. 덤프된 데이터베이스의 개체에 대한 권한을 소유하고 있거나 권한이 부여된 모든 사용자가 이미 있는지 확인합니다. 이러한 사용자가 없는 경우 복원에서 원래 소유권 및 권한을 가진 오브젝트를 다시 생성할 수 없습니다.

2. psql 유틸리티를 실행하여 pg_dumpall 유틸리티로 생성된 텍스트 파일 덤프를 복원합니다.

```
$ psql < dumpfile
```

여기에서 dumpfile 은 pg_dumpall 명령의 출력입니다.

4.5.1.6. 다른 서버에서 데이터베이스의 SQL 덤프 수행

pg_dump 및 psql 은 파이프로 쓰고 읽을 수 있기 때문에 한 서버에서 다른 서버로 직접 데이터베이스를 덤프할 수 있습니다.

절차

- 한 서버에서 다른 서버로 데이터베이스를 덤프하려면 다음을 실행합니다.

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

4.5.1.7. 복원 중 SQL 오류 처리

기본적으로 SQL 오류가 발생하는 경우 psql 이 계속 실행되므로 데이터베이스가 부분적으로만 복원됩니다.

기본 동작을 변경하려면 덤프를 복원할 때 다음 방법 중 하나를 사용합니다.

사전 요구 사항

-

postgres 수퍼유저 또는 데이터베이스 관리자 권한이 있는 사용자로 명령을 실행해야 합니다.

절차

- ON_ERROR_STOP 변수를 설정하여 SQL 오류가 발생하는 경우 psql 종료 상태로 3을 종료합니다.

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

- 복원이 완전히 완료되거나 취소되도록 전체 덤프가 단일 트랜잭션으로 복원되도록 지정합니다.
  - psql 유틸리티를 사용하여 텍스트 파일 덤프를 복원하는 경우:

```
$ psql -1
```

  - pg_restore 유틸리티를 사용하여 텍스트가 아닌 파일 덤프를 복원하는 경우:

```
$ pg_restore -e
```

    이 방법을 사용할 때 사소한 오류도 여러 시간 동안 이미 실행된 복원 작업을 취소할 수 있습니다.

4.5.1.8. 추가 리소스

- PostgreSQL 문서 - SQL 덤프

4.5.2. 파일 시스템 수준 백업을 사용하여 PostgreSQL 데이터 백업

파일 시스템 수준 백업을 생성하려면 PostgreSQL 데이터베이스 파일을 다른 위치에 복사합니다. 예를 들어 다음 방법을 사용할 수 있습니다.

- tar 유틸리티를 사용하여 아카이브 파일을 만듭니다.

- rsync 유틸리티를 사용하여 파일을 다른 위치에 복사합니다.
- 데이터 디렉터리의 일관된 스냅샷을 생성합니다.

4.5.2.1. 파일 시스템 백업의 이점 및 제한 사항

파일 시스템 수준 백업은 다른 PostgreSQL 백업 방법에 비해 다음과 같은 이점이 있습니다.

- 일반적으로 백업하는 파일 시스템 수준은 SQL 덤프보다 빠릅니다.

파일 시스템 수준 백업에는 다른 PostgreSQL 백업 방법에 비해 다음과 같은 제한 사항이 있습니다.

- 이 백업 방법은 RHEL 8에서 RHEL 9로 업그레이드하고 업그레이드된 시스템으로 데이터를 마이그레이션하려는 경우 적합하지 않습니다. 파일 시스템 수준 백업은 아키텍처 및 RHEL 주요 버전에 따라 다릅니다. 업그레이드에 성공적이지 않지만 RHEL 9 시스템의 데이터를 복원할 수 없는 경우 RHEL 8 시스템의 데이터를 복원할 수 있습니다.
- 데이터를 백업하고 복원하기 전에 데이터베이스 서버를 종료해야 합니다.
- 특정 개별 파일 또는 테이블을 백업하고 복원하는 것은 불가능합니다. 파일 시스템을 백업하는 것은 전체 데이터베이스 클러스터를 완전히 백업하고 복원하는 경우에만 작동합니다.

4.5.2.2. 파일 시스템 백업 수행

파일 시스템 수준 백업을 수행하려면 다음 절차를 사용하십시오.

절차

1. 데이터베이스 클러스터의 위치를 선택하고 이 클러스터를 초기화합니다.

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

2. postgresql 서비스를 중지합니다.

-

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

3. 모든 방법을 사용하여 파일 시스템 백업을 생성합니다(예: tar 아카이브).

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

4. postgresql 서비스를 시작합니다.

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

추가 리소스

- PostgreSQL 문서 - 파일 시스템 수준 백업

4.5.3. 연속 아카이빙을 통한 PostgreSQL 데이터 백업

4.5.3.1. 지속적인 아카이브 소개

PostgreSQL 은 데이터베이스의 데이터 파일의 모든 변경 사항을 클러스터 데이터 디렉터리의 pg_wal/ 하위 디렉터리에서 사용할 수 있는 쓰기 전 로그(WAL) 파일에 기록합니다. 이 로그는 주로 크래시 복구용으로 사용됩니다. 충돌이 발생한 후 마지막 checkpoint 이후 발생한 로그 항목을 일관성으로 복원하는 데 사용할 수 있습니다.

온라인 백업이라고도 하는 지속적인 아카이브 방법은 실행 중인 서버 또는 파일 시스템 수준 백업에서 수행되는 기본 백업 형태의 데이터베이스 클러스터 사본과 WAL 파일을 결합합니다.

데이터베이스 복구가 필요한 경우 데이터베이스 클러스터 복사본에서 데이터베이스를 복원한 다음 백업된 WAL 파일의 로그를 재생하여 시스템을 현재 상태로 가져올 수 있습니다.

지속적인 아카이브 방법을 사용하면 최소한 마지막 기본 백업 시작 시간으로 확장되는 모든 아카이브 WAL 파일의 연속 시퀀스를 유지해야 합니다. 따라서 기본 백업의 이상적인 빈도는 다음과 같이 달라집니다.

- 아카이브된 WAL 파일에 사용 가능한 스토리지 볼륨입니다.
- 복구가 필요한 상황에서 가능한 최대 데이터 복구 기간입니다. 마지막 백업 이후 기간이 긴 경우 시스템에서 더 많은 WAL 세그먼트를 재생하고 복구에 시간이 더 걸립니다.

참고

지속적인 아카이브 백업 솔루션의 일부로 **pg_dump** 및 **pg_dumpall SQL** 덤프를 사용할 수 없습니다. **SQL** 덤프는 논리적 백업을 생성하고 **WAL** 재생에서 사용할 수 있는 충분한 정보를 포함하지 않습니다.

연속 보관 방법을 사용하여 데이터베이스 백업 및 복원을 수행하려면 다음 지침을 따르십시오.

1. **WAL** 파일 보관에 대한 절차를 설정 및 테스트 - **WAL** 보관을 참조하십시오.

2. 기본 백업 수행 - 기본 백업을 참조하십시오.

데이터를 복원하려면 연속 보관을 사용하여 데이터베이스 복원 의 지침을 따릅니다.

4.5.3.2. 지속적인 아카이빙의 장점과 단점

연속 아카이브는 다른 **PostgreSQL** 백업 방법과 비교하여 다음과 같은 이점이 있습니다.

- 연속 백업 방법을 사용하면 백업의 내부 불일치가 로그 재생에 의해 수정되므로 완전히 일치하지 않는 기본 백업을 사용할 수 있습니다. 따라서 실행 중인 **PostgreSQL** 서버에서 기본 백업을 수행할 수 있습니다.

- 파일 시스템 스냅샷이 필요하지 않습니다. **tar** 또는 유사한 아카이브 유틸리티로 충분합니다.

- 연속 백업은 로그 재생에 대한 **WAL** 파일의 시퀀스가 무기한 길어질 수 있기 때문에 **WAL** 파일을 계속 보관하여 달성할 수 있습니다. 이는 대규모 데이터베이스에 특히 유용합니다.

- 연속 백업은 특정 시점 복구를 지원합니다. **WAL** 항목을 끝까지 재생하지 않아도 됩니다. 언제든지 재생을 중지할 수 있으며 기본 백업 이후 언제든지 데이터베이스를 해당 상태로 복원할 수 있습니다.

-

일련의 **WAL** 파일을 동일한 기본 백업 파일과 함께 로드한 다른 시스템에서 지속적으로 사용할 수 있는 경우 언제든지 거의 최신 데이터베이스 복사본으로 다른 시스템을 복원할 수 있습니다.

연속 아카이브에는 다른 **PostgreSQL** 백업 방법에 비해 다음과 같은 단점이 있습니다.

- 연속 백업 방법은 하위 집합이 아닌 전체 데이터베이스 클러스터의 복원만 지원합니다.
- 연속 백업에는 광범위한 보관 스토리지가 필요합니다.

4.5.3.3. WAL 아카이브 설정

실행 중인 **PostgreSQL** 서버는 일련의 **WAL(Write ahead log)** 레코드를 생성합니다. 서버는 이 시퀀스를 **WAL** 세그먼트 파일로 분할하며, **WAL** 시퀀스에서 해당 위치를 반영하는 숫자 이름이 지정됩니다. **WAL** 아카이브가 없으면 세그먼트 파일이 재사용되고 더 높은 세그먼트 번호로 이름이 변경됩니다.

WAL 데이터를 보관하면 세그먼트 파일을 재사용하기 전에 각 세그먼트 파일의 내용을 새 위치에 캡처하여 저장합니다. 다른 시스템의 **NFS** 마운트된 디렉토리, 테이프 드라이브 또는 **CD**와 같은 콘텐츠를 저장할 수 있는 여러 옵션이 있습니다.

WAL 레코드에는 구성 파일에 대한 변경 사항이 포함되지 않습니다.

WAL 아카이브를 활성화하려면 다음 절차를 사용합니다.

절차

1. **/var/lib/pgsql/data/postgresql.conf** 파일에서 다음을 수행합니다.
   a. **the wal_level** 구성 매개 변수를 **replica** 이상으로 설정합니다.
   b. **archive_mode** 매개 변수를 **on** 으로 설정합니다.
   c. **archive_command** 구성 매개 변수에 **shell** 명령을 지정합니다. **cp** 명령, 다른 명령 또는 쉘 스크립트를 사용할 수 있습니다.

2. **postgresql** 서비스를 다시 시작하여 변경 사항을 활성화합니다.

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

3. **archive** 명령을 테스트하고 기존 파일을 덮어쓰지 않고 실패하는 경우 **0**이 아닌 종료 상태를 반환하는지 확인합니다.

4. 데이터를 보호하려면 세그먼트 파일이 그룹 또는 세계 읽기 액세스 권한이 없는 디렉터리에 보관되었는지 확인합니다.

참고

archive 명령은 완료된 **WAL** 세그먼트에서만 실행됩니다. 작은 **WAL** 트래픽을 생성하는 서버는 트랜잭션 완료와 아카이브 스토리지의 안전한 기록 사이에 상당한 지연이 발생할 수 있습니다. 아카이브되지 않은 데이터가 얼마나 오래된지 제한하려면 다음을 수행할 수 있습니다.

- 지정된 빈도로 서버가 새 **WAL** 세그먼트 파일로 전환되도록 **archive_timeout** 매개 변수를 설정합니다.
- **pg_switch_wal** 매개 변수를 사용하여 세그먼트 스위치를 강제 적용하여 트랜잭션이 완료된 후 즉시 보관되도록 합니다.

예 4.5. WAL 세그먼트를 보관하기 위한 쉘 명령

이 예에서는 **archive_command** 구성 매개 변수에 설정할 수 있는 간단한 쉘 명령을 보여줍니다.

다음 명령은 완료된 세그먼트 파일을 필요한 위치로 복사합니다.

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

여기서 **%p** 매개 변수는 아카이브할 파일의 상대 경로로 바뀌고 **%f** 매개 변수는 파일 이름으로 교체됩니다.

이 명령은 보관 가능한 WAL 세그먼트를 /mnt/server/archivedir/ 디렉터리에 복사합니다. %p 및 % f 매개변수를 교체한 후 실행된 명령은 다음과 같습니다.

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

아카이브된 각 새 파일에 대해 유사한 명령이 생성됩니다.

추가 리소스

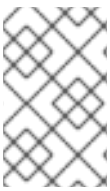
- PostgreSQL 16 문서

4.5.3.4. 기본 백업 만들기

몇 가지 방법으로 기본 백업을 만들 수 있습니다. 기본 백업을 수행하는 가장 간단한 방법은 실행 중인 PostgreSQL 서버에서 pg_basebackup 유틸리티를 사용하는 것입니다.

기본 백업 프로세스는 WAL 아카이브 영역에 저장된 백업 기록 파일을 만들고 기본 백업에 필요한 첫 번째 WAL 세그먼트 파일 다음에 이름이 지정됩니다.

백업 기록 파일은 시작 및 끝 시간과 백업의 WAL 세그먼트를 포함하는 작은 텍스트 파일입니다. 레이블 문자열을 사용하여 연결된 덤프 파일을 식별한 경우 백업 기록 파일을 사용하여 복원할 덤프 파일을 확인할 수 있습니다.



참고

데이터를 복구할 수 있도록 몇 가지 백업 세트를 확실하게 유지하는 것이 좋습니다.

기본 백업을 수행하려면 다음 절차를 사용합니다.

사전 요구 사항

- postgres 수퍼유저, 데이터베이스 관리자 권한이 있는 사용자 또는 최소 REPLICATION 권한이 있는 다른 사용자로 명령을 실행해야 합니다.

- 기본 백업 중 및 이후에 생성된 모든 **WAL** 세그먼트 파일을 보관해야 합니다.

절차

1. **pg_basebackup** 유틸리티를 사용하여 기본 백업을 수행합니다.

   - 개별 파일(일반 형식)으로 기본 백업을 생성하려면 다음을 수행합니다.

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

     backup_directory 를 선택한 백업 위치로 바꿉니다.

     테이블 공간을 사용하고 서버와 동일한 호스트에서 기본 백업을 수행하는 경우 **--tablespace-mapping** 옵션도 사용해야 합니다. 그렇지 않으면 백업을 동일한 위치에 쓰기를 시도할 때 백업이 실패합니다.

   - **tar** 아카이브(**tar** 및 압축된 형식)로 기본 백업을 생성하려면 다음을 수행합니다.

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

     backup_directory 를 선택한 백업 위치로 바꿉니다.

     이러한 데이터를 복원하려면 올바른 위치에 있는 파일을 수동으로 추출해야 합니다.

2. 기본 백업 프로세스가 완료되면 백업 기록 파일에 지정된 데이터베이스 클러스터의 복사본과 백업 중에 사용되는 **WAL** 세그먼트 파일을 안전하게 보관합니다.

3. **WAL** 세그먼트는 기본 백업보다 오래되어 더 이상 복원이 필요하지 않기 때문에 기본 백업에 사용되는 **WAL** 세그먼트보다 숫자적으로 낮은 값을 삭제합니다.

연결할 데이터베이스 서버 **pg_basebackup** 을 지정하려면 다음 명령줄 옵션을 사용합니다.

- 호스트를 정의하는 **-h** 옵션.

  기본 호스트는 **DASD HOST** 환경 변수에서 지정한 로컬 호스트 또는 호스트입니다.

- 포트를 정의하는 **-p** 옵션.

  기본 포트는 **PPP PORT** 환경 변수 또는 컴파일된 기본값에 의해 표시됩니다.

추가 리소스

- PostgreSQL 문서 - 기본 백업
- PostgreSQL 문서 - pg_basebackup 유틸리티

4.5.3.5. 연속 아카이브 백업을 사용하여 데이터베이스 복원

연속 백업을 사용하여 데이터베이스를 복원하려면 다음 절차를 따릅니다.

절차

1. 서버를 중지합니다.

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

2. 필요한 데이터를 임시 위치에 복사합니다.

   전체 클러스터 데이터 디렉터리 및 모든 테이블 공간을 복사하는 것이 좋습니다. 이를 위해서는 기존 데이터베이스의 두 사본을 저장할 수 있는 충분한 여유 공간이 필요합니다.

   공간이 충분하지 않은 경우 시스템이 중단되기 전에 보관되지 않은 로그가 포함될 수 있는 클러스터의 **pg_wal** 디렉터리 콘텐츠를 저장합니다.

3. 클러스터 데이터 디렉토리 아래에 있는 기존 파일과 하위 디렉토리를 사용하고 있는 테이블 공간의 루트 디렉토리 아래에 모두 제거합니다.

4. 기본 백업에서 데이터베이스 파일을 복원합니다.

   다음을 확인하십시오.

   - 파일이 올바른 소유권(root이 아닌 데이터베이스 시스템 사용자)을 사용하여 복원됩니다 .
   - 올바른 권한으로 파일이 복원됩니다.
   - pg_tblspc/ 하위 디렉터리의 심볼릭 링크가 올바르게 복원됩니다.

5. pg_wal/ 하위 디렉터리에 있는 파일을 모두 제거합니다.

   이러한 파일은 기본 백업에서 발생하므로 더 이상 사용되지 않습니다. pg_wal/ 을 보관하지 않은 경우 적절한 권한으로 다시 만듭니다.

6. 2단계에 저장된 아카이브되지 않은 WAL 세그먼트 파일을 pg_wal/ 에 복사합니다.

7. 클러스터 데이터 디렉터리에 recovery.conf 복구 명령 파일을 만들고 restore_command 구성 매개 변수에 shell 명령을 지정합니다. cp 명령, 다른 명령 또는 쉘 스크립트를 사용할 수 있습니다. 예를 들면 다음과 같습니다.

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

8. 서버를 시작합니다.

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

   서버는 복구 모드로 전환하고 필요한 아카이브된 WAL 파일을 읽습니다.

   외부 오류로 인해 복구가 종료되면 서버를 다시 시작할 수 있으며 복구가 계속됩니다. 복구 프로세스가 완료되면 서버의 recovery .conf의 이름이 recovery. done으로 변경됩니다. 이렇게 하면 서버가 정상적인 데이터베이스 작업을 시작한 후 실수로 복구 모드로 전환되지 않습니다.

9. 데이터베이스 내용을 확인하여 데이터베이스가 필수 상태로 복구되었는지 확인합니다.

데이터베이스가 필수 상태로 복구되지 않은 경우 1단계로 돌아갑니다. 데이터베이스가 필수 상태로 복구된 경우 사용자가 pg_hba.conf 파일에서 클라이언트 인증 구성을 복원하여 연결하도록 허용합니다.

연속 백업을 사용하여 복원하는 방법에 대한 자세한 내용은 PostgreSQL 문서를 참조하십시오.

4.5.3.6. 추가 리소스

- 연속 보관 방법

4.6. RHEL 9 버전의 POSTGRESQL으로 마이그레이션

Red Hat Enterprise Linux 8은 여러 모듈 스트림에 PostgreSQL 을 제공합니다. PostgreSQL 10 (기본 postgresql 스트림), PostgreSQL 9.6,PostgreSQL 12,PostgreSQL 13,PostgreSQL 15, PostgreSQL 16.

RHEL 9에서는 PostgreSQL 13,PostgreSQL 15 및 PostgreSQL 16 을 사용할 수 있습니다.

RHEL에서는 데이터베이스 파일에 대해 두 가지 PostgreSQL 마이그레이션 경로를 사용할 수 있습니다.

- pg_upgrade 유틸리티를 사용한 빠른 업그레이드
- 업그레이드 덤프 및 복원

빠른 업그레이드 방법은 dump 및 restore 프로세스보다 빠릅니다. 그러나 특정 경우 빠른 업그레이드가 작동하지 않으며 교차 아키텍처 업그레이드의 경우와 같이 덤프 및 복원 프로세스만 사용할 수 있습니다.

최신 버전의 PostgreSQL으로 마이그레이션하기 위한 전제 조건으로 모든 PostgreSQL 데이터베이스를 백업 합니다.

덤프 및 복원 프로세스에는 데이터베이스를 덤프하고 SQL 파일의 백업을 수행해야 하며 빠른 업그레이드 방법에 권장됩니다.

최신 버전의 PostgreSQL 으로 마이그레이션하기 전에 마이그레이션하려는 PostgreSQL 버전과 대상 버전 및 대상 버전 간의 모든 건너뛰는 PostgreSQL 버전에 대한 업스트림 호환성 노트 를 참조하십시오.

4.6.1. PostgreSQL 15와 PostgreSQL 16의 주요 차이점

PostgreSQL 16 에서는 다음과 같은 주요 변경 사항이 추가되었습니다.

postmasters 바이너리는 더 이상 사용할 수 없습니다.

PostgreSQL 은 더 이상 postmaster 바이너리와 함께 배포되지 않습니다. 제공된 systemd 장치 파일(systemctl start postgres 명령)을 사용하여 postgresql 서버를 시작하는 사용자는 이 변경의 영향을 받지 않습니다. 이전에 postmaster 바이너리를 통해 postgresql 서버를 직접 시작한 경우 대신 postgres 바이너리를 사용해야 합니다.

문서가 더 이상 패키지되지 않음

PostgreSQL 은 더 이상 패키지 내에서 PDF 형식으로 문서를 제공하지 않습니다. 대신 온라인 문서를 사용하십시오.

4.6.2. PostgreSQL 13과 PostgreSQL 15의 주요 차이점

PostgreSQL 15 에서는 이전 버전과 호환되지 않는 다음과 같은 변경 사항이 도입되었습니다.

공용 스키마의 기본 권한

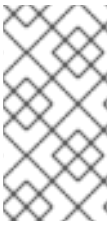
공용 스키마의 기본 권한이 PostgreSQL 15 에서 수정되었습니다. 새로 생성된 사용자는 GRANT ALL ON SCHEMA public TO myuser; 명령을 사용하여 권한을 명시적으로 부여해야 합니다.

다음 예제는 PostgreSQL 13 및 이전 버전에서 작동합니다.

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

다음 예제는 PostgreSQL 15 이상에서 작동합니다.

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



참고

mydbuser 액세스가 pg_hba.conf 파일에 적절하게 구성되었는지 확인합니다. 자세한 내용은 PostgreSQL 사용자 생성을 참조하십시오.

PQsendQuery() 가 더 이상 파이프라인 모드에서 지원되지 않음

PostgreSQL 15 이후 libpq PQsendQuery() 기능은 더 이상 파이프라인 모드에서 지원되지 않습니다. 대신 PQsendQueryParams() 함수를 사용하도록 영향을 받는 애플리케이션을 수정합니다.

4.6.3. pg_upgrade 유틸리티를 사용한 빠른 업그레이드

시스템 관리자는 빠른 업그레이드 방법을 사용하여 PostgreSQL 의 최신 버전으로 업그레이드할 수 있습니다. 빠른 업그레이드를 수행하려면 바이너리 데이터 파일을 /var/lib/pgsql/data/ 디렉터리에 복사하고 pg_upgrade 유틸리티를 사용합니다.

이 방법을 사용하여 데이터를 마이그레이션할 수 있습니다.

- RHEL 8 버전의 PostgreSQL 12 에서 RHEL 버전의 PostgreSQL 13으로
- RHEL 8 또는 9 버전의 PostgreSQL 13 에서 RHEL 버전의 PostgreSQL 15로
- RHEL 8 또는 9 버전의 PostgreSQL 15 에서 RHEL 버전의 PostgreSQL 16으로

다음 절차에서는 빠른 업그레이드 방법을 사용하여 RHEL 8 버전의 PostgreSQL 12 에서 RHEL 9 버전의 PostgreSQL 13 으로의 마이그레이션을 설명합니다. 12 가 아닌 postgresql 스트림에서 마이그레이션하는 경우 다음 방법 중 하나를 사용합니다.

- RHEL 8에서 PostgreSQL 서버를 버전 12로 업데이트한 다음 pg_upgrade 유틸리티를 사용하여 RHEL 9 버전의 PostgreSQL 13 으로 빠르게 업그레이드합니다.
-

덤프를 사용하고 **RHEL 9**의 **RHEL 8** 버전과 동일하거나 이후의 **PostgreSQL** 버전 간에 직접 업그레이드하십시오.

사전 요구 사항

- 업그레이드를 수행하기 전에 **PostgreSQL** 데이터베이스에 저장된 모든 데이터를 백업합니다. 기본적으로 모든 데이터는 **RHEL 8** 및 **RHEL 9** 시스템의 **/var/lib/pgsql/data/** 디렉터리에 저장됩니다.

절차

1. **RHEL 9** 시스템에서 **postgresql-server** 및 **postgresql-upgrade** 패키지를 설치합니다.

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

필요한 경우 **RHEL 8**에서 **PostgreSQL** 서버 모듈을 사용한 경우 두 버전의 **RHEL 9** 시스템에도 설치합니다. **PostgreSQL 12 (postgresql-upgrade** 패키지로 설치됨)와 대상 버전의 **PostgreSQL 13 (postgresql-server** 패키지로 설치됨)으로 컴파일됩니다. 타사 **PostgreSQL** 서버 모듈을 컴파일해야 하는 경우 **postgresql- devel** 및 **postgresql-upgrade-devel** 패키지에 대해 모두 빌드합니다.

2. 다음 항목을 확인합니다.

- 기본 설정**: RHEL 9** 시스템에서 서버가 기본 **/var/lib/pgsql/data** 디렉토리를 사용하는지와 데이터베이스가 올바르게 초기화되고 활성화되어 있는지 확인합니다. 또한 데이터 파일은 **/usr/lib/systemd/system/postgresql.service** 파일에 언급된 것과 동일한 경로에 저장해야 합니다.

- **PostgreSQL** 서버**:** 시스템에서 여러 **PostgreSQL** 서버를 실행할 수 있습니다. 이러한 모든 서버의 데이터 디렉터리가 독립적으로 처리되었는지 확인합니다.

- **PostgreSQL** 서버 모듈**: RHEL 8**에서 사용한 **PostgreSQL** 서버 모듈도 **RHEL 9** 시스템에 설치되어 있는지 확인합니다. 플러그인은 **/usr/lib64/pgsql/** 디렉터리에 설치됩니다.

3. **postgresql** 서비스가 데이터를 복사할 때 소스 및 대상 시스템에서 실행되지 않는지 확인합니다.

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

4. 소스 위치의 데이터베이스 파일을 **RHEL 9** 시스템의 **/var/lib/pgsql/data/** 디렉터리에 복사합니다.

5. **PostgreSQL** 사용자로 다음 명령을 실행하여 업그레이드 프로세스를 수행합니다.

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

그러면 백그라운드에서 **pg_upgrade** 프로세스가 시작됩니다.

실패하는 경우 **postgresql-setup** 은 정보적인 오류 메시지를 제공합니다.

6. **/var/lib/pgsql/data-old** 의 이전 구성을 새 클러스터로 복사합니다.

빠른 업그레이드는 최신 데이터 스택에서 이전 구성을 재사용하지 않으며 구성이 처음부터 생성됩니다. 이전 구성과 새 구성을 수동으로 결합하려면 데이터 디렉터리에서 ***.conf** 파일을 사용합니다.

7. 새 **PostgreSQL** 서버를 시작합니다.

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

8. 새 데이터베이스 클러스터를 분석합니다.

- **PostgreSQL 13** 의 경우:

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

- **PostgreSQL 15** 이상의 경우:

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

참고

REFRESH VERSION을 사용해야 할 수도 있습니다. 자세한 내용은 업스트림 문서를 참조하십시오.

9. 부팅 시 새 **PostgreSQL** 서버가 자동으로 시작되도록 하려면 다음을 실행합니다.

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

4.6.4. 업그레이드 덤프 및 복원

덤프 및 복원 업그레이드를 사용하는 경우 모든 데이터베이스의 콘텐츠를 **SQL** 파일 덤프 파일에 덤프해야 합니다. 덤프 및 복원 업그레이드는 빠른 업그레이드 방법보다 느리고 생성된 **SQL** 파일에 수동 수정이 필요할 수 있습니다.

이 방법은 **RHEL 8** 버전의 **PostgreSQL**에서 **RHEL 9**의 **PostgreSQL** 과 동일하거나 최신 버전으로 데이터를 마이그레이션하는 데 사용할 수 있습니다.

RHEL 8 및 **RHEL 9** 시스템에서 **PostgreSQL** 데이터는 기본적으로 **/var/lib/pgsql/data/** 디렉터리에 저장됩니다.

덤프를 수행하고 업그레이드를 복원하려면 사용자를 **root** 로 변경합니다.

다음 절차에서는 **RHEL 8** 기본 버전의 **PostgreSQL 10** 에서 **RHEL 9** 버전의 **PostgreSQL 13** 으로의 마이그레이션을 설명합니다.

절차

1. **RHEL 8** 시스템에서 **PostgreSQL 10** 서버를 시작합니다.

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

2. **RHEL 8** 시스템에서 모든 데이터베이스 내용을 **pgdump_file.sql** 파일에 덤프합니다.

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

3. 데이터베이스가 올바르게 덤프되었는지 확인합니다.

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

결과적으로 덤프된 **sql** 파일의 경로가 **/var/lib/pgsql/pgdump_file.sql** 이 표시됩니다.

4. **RHEL 9** 시스템에서 **postgresql-server** 패키지를 설치합니다.

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

필요한 경우 **RHEL 8**에서 **PostgreSQL** 서버 모듈을 사용한 경우 **RHEL 9** 시스템에도 설치합니다. 타사 **PostgreSQL** 서버 모듈을 컴파일해야 하는 경우 **postgresql-devel** 패키지에 대해 빌드합니다.

5. **RHEL 9** 시스템에서 새 **PostgreSQL** 서버의 데이터 디렉터리를 초기화합니다.

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

6. **RHEL 9** 시스템에서 **pgdump_file.sql** 을 **PostgreSQL** 홈 디렉터리에 복사하고 파일이 올바르게 복사되었는지 확인합니다.

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

7. **RHEL 8** 시스템에서 구성 파일을 복사합니다.

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

복사할 구성 파일은 다음과 같습니다.

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

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

8. **RHEL 9 시스템에서 새 PostgreSQL 서버를 시작합니다.**

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

9. **RHEL 9 시스템에서 덤프된 sql 파일에서 데이터를 가져옵니다.**

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