

Database Preparation



- [SQLite](#)
 - [MySQL/MariaDB](#)
 - [PostgreSQL](#)
 - [Database Connection over TLS](#)
 - [Prerequisites](#)
 - [PostgreSQL](#)
 - [MySQL/MariaDB](#)
 - [Dedicated Primary/Replica Connection Pools](#)
-

Forgejo uses a database to store relational data, however not every relational database can be used by Forgejo. Only the following databases are supported:

- MariaDB (≥ 10.6), using the InnoDB storage engine
- MySQL (≥ 8.4), using the InnoDB storage engine
- PostgreSQL (≥ 14)
- SQLite

For a recommendation on which database to use and fits your needs, [read the recommendation page](#).

The remainder of this page will be a guide how to prepare each database type. Also take a look at the [database section of the config cheat sheet](#) for a detailed list of options in Forgejo.

SQLite

The binary distributed by the Forgejo project includes SQLite support, meaning you don't have to install additional dependencies on your system.

“ [!note] Note If you build Forgejo from source, build with `make TAGS="sqlite sqlite_unlock_notify" build` to include SQLite support.

Choosing SQLite only requires setting the database type and optionally the path to a database file:

```
[database]
DB_TYPE = sqlite3
# optional if you want to specify another location
# by default, the database file will be stored relative to other data
PATH = data/forgejo.db
```

No maintenance is expected by using SQLite; it might be needed to run the `VACUUM` command if the database size grows too big. This reclaims space that was freed after content was deleted.

MySQL/MariaDB

[Install MariaDB](#) or [install MySQL](#) on the machine where you want to host the database, this can be on the same machine. It is also perfectly fine to use an existing database installation.

We need to perform several administrative actions on the database. In order to accomplish that you need to get into the database instance as root. If you just installed the database, then that would be done as followed:

```
mysql -u root -p
```

Enter the password as prompted.

We will create a new database user that Forgejo will use to connect to the instance. **Please, generate a secure password**; for this example we use `passw0rd` as password and `forgejo` as username.

In the case the instance is on the same machine (if relevant, in the same container):

```
CREATE USER 'forgejo'@'localhost' IDENTIFIED BY 'passw0rd';
```

In all other cases you have to figure out from which IP address the Forgejo instance will be communicating from to the database instance. In this example we assume that to be `192.0.2.10`.

```
CREATE USER 'forgejo'@'192.0.2.10' IDENTIFIED BY 'passw0rd';
```

Now we will create the database that Forgejo will use. Forgejo expects a case-sensitive collation, so we must account for this. `utf8mb4_bin` is a common collation for MariaDB and MySQL. When Forgejo starts, it will try to find a better collation (`utf8mb4_0900_as_cs` or `uca1400_as_cs`) and alter the database if it is possible. If you would like to use another collation, you can set `[database].CHARSET_COLLATION` in the `app.ini` file.

In this example we use `utf8mb4_bin` as collation and `forgejo` as the name to create the database:

```
CREATE DATABASE forgejo CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_bin';
```

Last step is to grant privileges to the `forgejo` user for the `forgejo` database.

In the case the instance is on the same machine (if relevant, in the same container):

```
GRANT ALTER, CREATE, DELETE, DROP, INDEX, INSERT, REFERENCES, SELECT, UPDATE ON forgejo.* TO  
'forgejo'@'localhost';  
FLUSH PRIVILEGES;
```

In all other cases you have to figure out from which IP address the Forgejo instance will be communicating from to the database instance. In this example we assume that to be `192.0.2.10`.

```
GRANT ALTER, CREATE, DELETE, DROP, INDEX, INSERT, REFERENCES, SELECT, UPDATE ON forgejo.* TO  
'forgejo'@'192.0.2.10';  
FLUSH PRIVILEGES;
```

You can now quit the CLI, by executing the `exit` command.

PostgreSQL

1. Install the PostgreSQL server component on the system you would like to store the database on.
2. Protect the `root` user with a secure password or disable the login.
3. On the database server, login to the database console as superuser:

```
su -c "psql" - postgres
```

4. Create database user (role in PostgreSQL terms) with login privilege and password.

Please use a secure, strong password instead of `'passw0rd'` below:

```
CREATE ROLE forgejo WITH LOGIN PASSWORD 'passw0rd';
```

Replace username and password as appropriate.

5. Create database with UTF-8 charset and owned by the database user created earlier. Any `libc` collations can be specified with `LC_COLLATE` and `LC_CTYPE` parameter, depending on expected content:

```
CREATE DATABASE forgejodb WITH OWNER forgejo TEMPLATE template0 ENCODING UTF8  
LC_COLLATE 'en_US.UTF-8' LC_CTYPE 'en_US.UTF-8';
```

Replace database name as appropriate.

6. Allow the database user to access the database created above by adding the following authentication rules to `pg_hba.conf`.

For local database:

```
local    forgejodb    forgejo                scram-sha-256
host     forgejodb    forgejo    127.0.0.1/32    scram-sha-256 # IPv4 local
connections
host     forgejodb    forgejo    ::1/128         scram-sha-256 # IPv6 local
connections
```

For remote database:

```
host     forgejodb    forgejo    192.0.2.10/32    scram-sha-256
```

Replace database name, user, and IP address of Forgejo instance with your own.

Note: rules on `pg_hba.conf` are evaluated sequentially, that is the first matching rule will be used for authentication. Your PostgreSQL installation may come with generic authentication rules that match all users and databases. You may need to place the rules presented here above such generic rules if it is the case.

Restart PostgreSQL to apply new authentication rules.

7. Optional: Test connection to the database from the Forgejo container.

For local database:

```
psql -U forgejo -d forgejodb -h localhost
```

For remote database:

```
psql "postgres://forgejo@203.0.113.3/forgejodb"
```

where `forgejo` is database user, `forgejodb` is database name, and `203.0.113.3` is IP address of your database instance.

You should be prompted to enter password for the database user, and connected to the database.

Database Connection over TLS

If the communication between Forgejo and your database instance is performed through a private network, or if Forgejo and the database are running on the same server, this section can be omitted since the security between Forgejo and the database instance is not critically exposed. If instead the database instance is on a public network, use TLS to encrypt the connection to the database, as it is possible for third-parties to intercept the traffic data.

Prerequisites

- You need two valid TLS certificates, one for the database instance (database server) and one for the Forgejo instance (database client). Both certificates must be signed by a trusted CA.
- The database certificate must contain `TLS Web Server Authentication` in the `X509v3 Extended Key Usage` extension attribute, while the client certificate needs `TLS Web Client Authentication` in the corresponding attribute.
- On the database server certificate, one of `Subject Alternative Name` or `Common Name` entries must be the fully-qualified domain name (FQDN) of the database instance (e.g. `db.example.com`). On the database client certificate, one of the entries mentioned above must contain the database username that Forgejo will be using to connect.
- You need domain name mappings of both Forgejo and database servers to their respective IP addresses. Either set up DNS records for them or add local mappings to `/etc/hosts` (`%WINDIR%\System32\drivers\etc\hosts` in Windows) on each system. This allows the database connections to be performed by domain name instead of IP address. See documentation of your system for details.

PostgreSQL

The PostgreSQL driver used by Forgejo supports two-way TLS. In two-way TLS, both database client and server authenticate each other by sending their respective certificates to their respective opposite for validation. In other words, the server verifies client certificate, and the client verifies server certificate.

1. On the server with the database instance, place the following credentials:

- `/path/to/postgresql.crt`: Database instance certificate
- `/path/to/postgresql.key`: Database instance private key
- `/path/to/root.crt`: CA certificate chain to validate client certificates

2. Add following options to `postgresql.conf`:

```
ssl = on
ssl_ca_file = '/path/to/root.crt'
ssl_cert_file = '/path/to/postgresql.crt'
ssl_key_file = '/path/to/postgresql.key'
ssl_min_protocol_version = 'TLSv1.2'
```

3. Adjust credentials ownership and permission, as required by PostgreSQL:

```
chown postgres:postgres /path/to/root.crt /path/to/postgresql.crt
/path/to/postgresql.key
chmod 0600 /path/to/root.crt /path/to/postgresql.crt /path/to/postgresql.key
```

4. Edit `pg_hba.conf` rule to only allow Forgejo database user to connect over SSL, and to require client certificate verification.

```
hostssl    forgejodb    forgejo    192.0.2.10/32    scram-sha-256
clientcert=verify-full
```

Replace database name, user, and IP address of Forgejo instance as appropriate.

5. Restart PostgreSQL to apply configurations above.
6. On the server running the Forgejo instance, place the following credentials under the home directory of the user who runs Forgejo (e.g. `git`):

- `~/.postgresql/postgresql.crt` : Database client certificate
- `~/.postgresql/postgresql.key` : Database client private key
- `~/.postgresql/root.crt` : CA certificate chain to validate server certificate

Note: Those file names above are hardcoded in PostgreSQL and it is not possible to change them.

7. Adjust credentials, ownership and permission as required:

```
chown git:git ~/.postgresql/postgresql.crt ~/.postgresql/postgresql.key
~/.postgresql/root.crt
chown 0600 ~/.postgresql/postgresql.crt ~/.postgresql/postgresql.key
~/.postgresql/root.crt
```

8. Test the connection to the database:

```
psql "postgres://forgejo@example.db/forgejodb?sslmode=verify-full"
```

You should be prompted to enter password for the database user, and then be connected to the database.

MySQL/MariaDB

While the MySQL/MariaDB driver used by Forgejo also supports two-way TLS, Forgejo currently supports only one-way TLS. See the ["Add TLS File Path Options for MySQL/MariaDB Database Connection"](#) issue for details.

In one-way TLS, the database client verifies the certificate sent from server during the connection handshake, and the server assumes that the connected client is legitimate, since client certificate verification doesn't take place.

1. On the database instance, place the following credentials:
 - `/path/to/mysql.crt` : Database instance certificate
 - `/path/to/mysql.key` : Database instance key
 - `/path/to/ca.crt` : CA certificate chain. This file isn't used on one-way TLS, but is used to validate client certificates on two-way TLS.

2. Add following options to `my.cnf`:

```
[mysqld]
ssl-ca = /path/to/ca.crt
ssl-cert = /path/to/mysql.crt
ssl-key = /path/to/mysql.key
tls-version = TLSv1.2,TLSv1.3
```

3. Adjust credentials ownership and permission:

```
chown mysql:mysql /path/to/ca.crt /path/to/mysql.crt /path/to/mysql.key
chmod 0600 /path/to/ca.crt /path/to/mysql.crt /path/to/mysql.key
```

4. Restart MySQL/MariaDB to apply the setting.
5. The database user for Forgejo may have been created earlier, but it would authenticate only against the IP addresses of the server running Forgejo. To authenticate against its domain name, recreate the user, and this time also set it to require TLS for connecting to the database:

```
DROP USER 'forgejo'@'192.0.2.10';
CREATE USER 'forgejo'@'example.forgejo' IDENTIFIED BY 'passw0rd' REQUIRE SSL;
GRANT ALL PRIVILEGES ON forgejodb.* TO 'forgejo'@'example.forgejo';
FLUSH PRIVILEGES;
```

Replace database user name, password, and Forgejo instance domain as appropriate.

6. Make sure that the CA certificate chain required to validate the database server certificate is on the system certificate store of both the database and Forgejo servers. Consult your system documentation for instructions on adding a CA certificate to the certificate store.
7. On the server running Forgejo, test connection to the database:

```
mysql -u forgejo -h example.db -p --ssl
```

You should be connected to the database.

Dedicated Primary/Replica Connection Pools

Starting with v12, Forgejo supports connecting to a dedicated primary/replica database setup. This allows scaling the database horizontally by adding more replicas which process all read-only queries, effectively offloading read operations from the primary database. This setup is particularly useful for high-traffic instances where read operations can significantly impact performance. It might not be needed for low-traffic/small instances.

The implementation uses the `xorm.EngineGroup` logic to manage the connection pool and load balancing between the primary and replica databases. Furthermore it allows defining a [load balancing strategy](#) for the replica databases.

The default, `Random`, distributes read-only queries across the replica databases in a random manner. Other policies can be used to distribute read-only queries in a round-robin (`RoundRobin`) manner, based on the load on each replica (`LeastConn`), or by using custom weights (`WeightRandom` and `WeightRoundRobin`).

Using a load balancing strategy is only in effect when the number of replica databases in the `EngineGroup` is greater than 1. An alternative to the `xorm` load balancing strategy is to use a dedicated load balancer in front of the replica database instances, such as HAProxy, or more advanced, DB-specific solutions like [pgcat](#) or [pgdog](#).

📄 #3

📄 cocobook-sync 📄 2026-10-28 16:40:56 UTC

📄 cocobook-sync 📄 2026-10-29 01:09:33 UTC