

# Installation

- [Installation](#)
- [Installation with Docker](#)
- [Installation from binary](#)
- [Database Preparation](#)

# Installation

Forgejo publishes a stable release every three months and a long term support (LTS) release every year. Patch releases are published more frequently and provide fixes for bugs and security vulnerabilities. Please review the [releases management](#) and also the [Forgejo versioning scheme](#) documentation for further information.

Get started in a few simple steps:

1. Set up Forgejo on your system ...
  - [from binary](#)
  - [with Docker](#)
  - with third-party packaging for various platforms ([see the "Delightful Forgejo" list](#)).  
Unlike the two official installation methods, these resources are not extensively tested to work on every release.
2. Connect to the web interface and complete the configuration and read [our recommendations](#), and,
3. register the first user which will be granted administrative permissions.

If you already have Gitea installed through your package manager, look at the [Gitea migration](#) guide for information on how to install Forgejo, while preserving your data from your Gitea installation.

# Installation with Docker

Forgejo provides [container images](#) for use with Docker or other containerization tools.

```
docker pull codeberg.org/forgejo/forgejo:15
```

If `codeberg.org` cannot be accessed, you can replace every mention of `codeberg.org` with `data.forgejo.org` to use our mirror.

The **15** tag is set to be the latest minor release, starting with **15.0.x**. The **15** tag will then be equal to **15.0.4** when it is released and so on. The **15.0** tag is also set to be the latest patch version release.

Upgrading from **X** to **X+1** (for instance from **12** to **13**) requires a [manual operation and human verification](#). However it is possible to use the **X** tag (for instance **13**) to get the latest minor release automatically.

## Security note for v15 LTS deployments

By default, the setting `security.REVERSE_PROXY_TRUSTED_PROXIES` is set to `*` in versions <16.0.0 of Forgejo container images. Paired with exposed web port and `[service].ENABLE_REVERSE_PROXY_AUTHENTICATION` set to `true` this can have devastating consequences to security.

The value `*` was changed in Forgejo v16.0.0, but as a breaking change it was not changed in v15 LTS.

To keep your deployment secure please make sure that:

- When using reverse proxy, Forgejo web port (`:3000` by default) is not directly accessible from untrusted networks
- `security.REVERSE_PROXY_TRUSTED_PROXIES` is set to a some value other than `*`. For example, `127.0.0.0/8,::1/128`. Additional networks can be added with commas, for example, to include your Docker bridge network

## Docker:

Here is an example [docker-compose](#) file:

```
networks:
  forgejo:
    external: false

services:
  server:
    image: codeberg.org/forgejo/forgejo:15
    container_name: forgejo
    environment:
      - USER_UID=1000
      - USER_GID=1000
    restart: always
    networks:
      - forgejo
    volumes:
      - ./forgejo:/data
      - /etc/localtime:/etc/localtime:ro
    ports:
      - '3000:3000'
      - '222:22'
```

Note that the volume should be owned by the user/group with the UID/GID specified in the config file. If you don't set the volume correct permissions, the container may not start.

## Podman:

Currently tested on the following specifications:

- Podman v5.2.1
- Fedora 40, Alma Linux 9.4, Rocky Linux 9

Save the following files in `/etc/containers/systemd`, as port 222 requires elevated privileges:

```
# forgejo.container
[Container]
ContainerName=forgejo
Environment=USER_UID=1000
Environment=USER_GID=1000
Image=codeberg.org/forgejo/forgejo:15
Network=forgejo.network
```

```
PublishPort=3000:3000
PublishPort=222:22
Volume=forgejo-data:/data
```

```
[Service]
Restart=always
```

```
[Install]
WantedBy=default.target
```

```
---
```

```
# forgejo.network
[Network]
NetworkName=forgejo
```

```
---
```

```
#forgejo-data.volume
[Volume]
VolumeName=forgejo-data
```

Once saved, run:

```
sudo systemctl daemon-reload
sudo systemctl start forgejo
```

On a browser, go to `http://localhost:3000` to initialize Onboarding

Note: SSH will be accessible to the container via port 222 on `localhost`. Once SSH keys have been verified on Forgejo, to confirm functionality, run:

```
ssh -F /dev/null git@<address> -p 222
```

# Configuration

The Forgejo configuration is stored in the `app.ini` file as described in the [Configuration Cheat Sheet](#). When using the Forgejo container image, this file is automatically created if it does not exist already.

It is possible to add and modify configuration values by setting environment variables named in format `FORGEJO__[SECTION]__[KEY]`. The `DEFAULT` section of the configuration should be an empty string. For instance:

```
FORGEJO__APP_NAME=Frogejo []
FORGEJO__repository__ENABLE_PUSH_CREATE_USER=true
```

is equivalent to adding the following to `app.ini`:

```
APP_NAME=Frogejo []

[repository]
ENABLE_PUSH_CREATE_USER = true
```

“ [!note] Note It is not possible to use environment variables to remove an existing value, it must be done by editing the `app.ini` file.

“ [!note] Note In case you are in a SELinux environment check the audit logs if you are having issues with containers.

# Databases

In the following each database is shown as part of a `docker-compose` example file, with a diff-like presentation that highlights additions to the example above.

If no database is configured, it will default to using SQLite.

## MySQL database

```
networks:
  forgejo:
    external: false

services:
  server:
```

```

image: codeberg.org/forgejo/forgejo:15
container_name: forgejo
environment:
  - USER_UID=1000
  - USER_GID=1000
+   - FORGEJO__database__DB_TYPE=mysql
+   - FORGEJO__database__HOST=db:3306
+   - FORGEJO__database__NAME=forgejo
+   - FORGEJO__database__USER=forgejo
+   - FORGEJO__database__PASSWD=forgejo
restart: always
networks:
  - forgejo
volumes:
  - ./forgejo:/data
  - /etc/localtime:/etc/localtime:ro
ports:
  - "3000:3000"
  - "222:22"
+ depends_on:
+   - db
+
+ db:
+   image: mysql:8
+   restart: always
+   environment:
+     - MYSQL_ROOT_PASSWORD=forgejo
+     - MYSQL_USER=forgejo
+     - MYSQL_PASSWORD=forgejo
+     - MYSQL_DATABASE=forgejo
+   networks:
+     - forgejo
+   volumes:
+     - ./mysql:/var/lib/mysql

```

## PostgreSQL database

```
networks:
  forgejo:
    external: false

services:
  server:
    image: codeberg.org/forgejo/forgejo:15
    container_name: forgejo
    environment:
      - USER_UID=1000
      - USER_GID=1000
+     - FORGEJO__database__DB_TYPE=postgres
+     - FORGEJO__database__HOST=db:5432
+     - FORGEJO__database__NAME=forgejo
+     - FORGEJO__database__USER=forgejo
+     - FORGEJO__database__PASSWD=forgejo
    restart: always
    networks:
      - forgejo
    volumes:
      - ./forgejo:/data
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "3000:3000"
      - "222:22"
+   depends_on:
+     - db
+
+   db:
+     image: postgres:14
+     restart: always
+     environment:
+       - POSTGRES_USER=forgejo
+       - POSTGRES_PASSWORD=forgejo
+       - POSTGRES_DB=forgejo
+     networks:
+       - forgejo
+     volumes:
+       - ./postgres:/var/lib/postgresql/data
```

# Using rootless image

Rootless image uses a different path for the data folder for Forgejo.

Also, the correct permissions must be set for bound folders, here, we will keep the user 1000 and groupid from before:

```
mkdir -p ./forgejo
sudo chown -R 1000:1000 ./forgejo
```

Here is an example of a `docker-compose.yaml` adapted from the previous PostgreSQL:

```
networks:
  forgejo:
    external: false

services:
  server:
    - image: codeberg.org/forgejo/forgejo:15
    + image: codeberg.org/forgejo/forgejo:15-rootless
    container_name: forgejo
    + user: 1000:1000
    environment:
      - USER_UID=1000
      - USER_GID=1000
      - FORGEJO__database__DB_TYPE=postgres
      - FORGEJO__database__HOST=db:5432
      - FORGEJO__database__NAME=forgejo
      - FORGEJO__database__USER=forgejo
      - FORGEJO__database__PASSWD=forgejo
    restart: always
    networks:
      - forgejo
    volumes:
    - - ./forgejo:/data
    + - ./forgejo:/var/lib/gitea
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "3000:3000"
```

```

-       - "222:22"
+       - "222:2222"
  depends_on:
    - db

  db:
    image: postgres:14
    restart: always
    environment:
      - POSTGRES_USER=forgejo
      - POSTGRES_PASSWORD=forgejo
      - POSTGRES_DB=forgejo
    networks:
      - forgejo
    volumes:
      - ./postgres:/var/lib/postgresql/data

```

## Using rootless image on raspberry pi

Unfortunately, docker on raspberry pis sometimes has issues with permission remapping when mounting folders, even if that is not enabled within docker. So the configuration provided above will fail because the service will be unable to create files in the `./forgejo` directory. A workaround for this is to use named volumes. Amending the above compose file in the following ways:

```

networks:
  forgejo:
    external: false

+volumes:
+  forgejo-data:
+    driver: local
+    driver_opts:
+      type: none
+      o: bind
+      device: ./forgejo
+  postgres-data:
+    driver: local
+    driver_opts:
+      type: none

```

```
+      o: bind
+      device: ./postgres

services:
  server:
    image: codeberg.org/forgejo/forgejo:15-rootless
    container_name: forgejo
    user: 1000:1000
    environment:
      - USER_UID=1000
      - USER_GID=1000
      - FORGEJO__database__DB_TYPE=postgres
      - FORGEJO__database__HOST=db:5432
      - FORGEJO__database__NAME=forgejo
      - FORGEJO__database__USER=forgejo
      - FORGEJO__database__PASSWD=forgejo
    restart: always
    networks:
      - forgejo
    volumes:
      - ./forgejo:/var/lib/gitea
+     - forgejo-data:/var/lib/gitea
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "3000:3000"
      - "222:2222"
    depends_on:
      - db

  db:
    image: postgres:14
    restart: always
    environment:
      - POSTGRES_USER=forgejo
      - POSTGRES_PASSWORD=forgejo
      - POSTGRES_DB=forgejo
    networks:
      - forgejo
    volumes:
```

```
- - ./postgres:/var/lib/postgresql/data
+ - postgres-data:/var/lib/postgresql/data
```

# Hosting repository data on remote storage systems

You might also mount the data and repository folders on a remote drive such as a network-attached storage system. While there are a multitude of possible solutions, we will focus on a somewhat minimal setup with NFS here and explain what measures have to be taken in general so that the administrators can adapt this to their individual setup.

We begin to describe a possible setup and will try to highlight all important aspects which the administrator will have to consider if a different hosting environment is present. An important assumption for the Forgejo image to make is to own the folders it writes into and reads from. This is naturally an issue since file-system permissions are a machine-local concept and don't translate over the network easily.

We assume that a server with the hostname `server` is accessible which has a folder `/repositories` shared via NFS. Append an entry to your `/etc/exports` like

```
[...]
/repositories    *(rw,sync,all_squash,sec=sys,anonuid=1024,anongid=100)
```

Four aspects to consider:

- The folder is mounted as `rw`, meaning clients can both read and write in the folder.
- The folder is mounted as `sync`. This is NFS-specific but means that transactions block until they are finished. This is not essential but increases the robustness against file corruption
- The `all_squash` setting maps all file accesses to an anonymous user, meaning that both the files of a user with the UID of `1050` and `1051` are mapped to a single `UID` on the server.
- We set these anonymous (G/U)ID to explicit values on the server with `anonuid=1024,anongid=100`. Hence all files will be owned by a user with the UID `1024`, belonging to a group `100`. Make sure the UID is available and a group with that ID is present.

Effectively we are now able to write and create files and folders on the remote share. With the `all_squash` setting, we map all users to one user, hence all data writable by one user is writable by all users, implying all files have a `drwxrwxrwx` setting (abbreviated "`0777` permissions"). We can also "fake-own" data, since all `chown` calls are now mapped to the anonymous user. This is an important behaviour. We now mount this folder on the `client` which will host Forgejo to a folder

```
/mnt/repositories...
```

```
sudo mount -o hard,timeo=10,retry=10,vers=4.1 server:/repositories /mnt/repositories/
```

... and create two folders

```
mkdir data
```

To consider in the NFS client setup is the `hard` setting, blocking all file operations if the share is not available. This prevents state changes in the repository which could potentially corrupt the repository data and is an NFS-specific setting.

We will use the `rootless` image, which hosts the `ssh` server for Forgejo embedded. A possible entry for a `docker-compose` file would look like this (shown as a diff-like view to the example shown [in our initial example](#)):

```
networks:
  forgejo:
    external: false

services:
  server:
-   image: codeberg.org/forgejo/forgejo:15
+   image: codeberg.org/forgejo/forgejo:15-rootless
    container_name: forgejo
+   user: "1024:100"
-   environment:
-     - USER_UID=1000
-     - USER_GID=1000

    restart: always
    networks:
      - forgejo
    volumes:
-     - ./forgejo:/data
+     - /mnt/repositories/data:/var/lib/gitea
      - /etc/localtime:/etc/localtime:ro
    ports:
-     - "3000:3000"
-     - "222:22"
+     - "222:2222"
```

This will write the configuration and all other data into the `data` folder. Make sure that `1024` and `100` match the `anonuid` and `anongid` setting in the NFS server setting here such that the Forgejo user sees files and folders with the same UID and GID in the respective folders and thus identifies itself as the sole owner of the folder structure.

Using the `rootless` image here solves another problem resulting from the file-system ownership issue. If we create ssh keys on the `client` image and save them on the `server`, they too will have `0777` permissions, which is prohibited by `openssh`. It is important for all involved tools that these files not be writable by just anybody with a login, so you would get you an error if you try to use them. Changing permissions will also not succeed through the chosen `all_squash` setup, which was necessary to allow a correct ownership mechanic on the server. To resolve this, we consider the `rootless` image, which embeds the `ssh` server, circumventing the problem entirely.

Note, that this setup is simple and does not necessarily reflect the reality of your network. User mapping and ownership could be streamlined better with Kerberos, but that is out of the scope of this guide.

# Installation from binary

## Install Forgejo and Git, create git user

“ [!note] Note This guide uses git.example.com and x.y.z for illustrative purpose. Replace with your domain and the used Forgejo version as appropriate.

First, download the Forgejo binary for your CPU architecture and verify the GPG signature, as described on [the Forgejo download page](#).

Next, copy the downloaded Forgejo binary to `/usr/local/bin/` (renaming it to "forgejo") and make it executable:

```
sudo cp forgejo-x.y.z-linux-amd64 /usr/local/bin/forgejo
sudo chmod 755 /usr/local/bin/forgejo
```

Make sure `git` and `git-lfs` are installed on your system. On Debian GNU/Linux you can use:

```
sudo apt install git git-lfs
```

Create a user `git` on the system. Forgejo will run as that user, and when accessing git through SSH (which is the default), this user is part of the URL (*for example in `git clone git@git.example.com:YourOrg/YourRepo.git` the `git` before the `@` is the user you'll create now*). On **Debian, Ubuntu** and their derivatives that's done with:

```
sudo adduser --system --shell /bin/bash --gecos 'Git Version Control' \
  --group --disabled-password --home /home/git git
```

On **Linux distributions not based on Debian/Ubuntu** (this should at least work with Red Hat derivatives like Fedora, CentOS etc.), run this instead:

```
sudo groupadd --system git
sudo useradd --system --shell /bin/bash --comment 'Git Version Control' \
  --gid git --home-dir /home/git --create-home git
```

# Create directories Forgejo will use

Now create the directories Forgejo will use and set access permissions appropriately:

```
sudo mkdir /var/lib/forgejo
sudo chown git:git /var/lib/forgejo && sudo chmod 750 /var/lib/forgejo
```

This is the directory Forgejo will store its data in, including your Git repositories.

```
sudo mkdir /etc/forgejo
sudo chown root:git /etc/forgejo && sudo chmod 770 /etc/forgejo
```

This is the directory Forgejo's config, called `app.ini`, is stored in. **Initially it needs to be writable by Forgejo**, but after the installation you can make it read-only for Forgejo because then it shouldn't modify it anymore.

## Optional: Set up database

When using sqlite as Forgejo's database, nothing needs to be done here.

If you need a more powerful database, you can use MySQL/MariaDB or PostgreSQL (apparently sqlite is good enough for at least 10 users, but might even suffice for more).

See [Forgejo's Database Preparation guide](#) for setup instructions.

## Install systemd service for Forgejo

Forgejo provides a [systemd service script](#). Download it to the correct location:

```
sudo wget -O /etc/systemd/system/forgejo.service
https://codeberg.org/forgejo/forgejo/raw/branch/forgejo/contrib/systemd/forgejo.service
```

If you're *not* using sqlite, but MySQL or MariaDB or PostgreSQL, you'll have to edit that file ( `/etc/systemd/system/forgejo.service` ) and uncomment the corresponding `Wants=` and `After=` lines. Otherwise it *should* work as it is.

After adding or modifying a systemd service file, you will need to reload systemd.

```
sudo systemctl daemon-reload
```

Now enable and start the Forgejo service, so you can go on with the installation:

```
sudo systemctl enable forgejo.service
sudo systemctl start forgejo.service
```

# Forgejo's web-based configuration

Access the Forgejo web UI by opening `http://localhost:3000/` in your local web browser. Replace `localhost` with the server's local IP address if necessary.

If it doesn't work:

- Make sure the forgejo service started successfully by checking the output of:

```
sudo systemctl status forgejo.service
```

If that indicates an error but the log lines underneath are too incomplete to tell what caused it,

```
sudo journalctl -n 100 --unit forgejo.service
```

will print the last 100 lines logged by Forgejo.

You should be greeted by Forgejo's "Initial Configuration" screen. The settings should be mostly self-explanatory, some hints:

- Select the correct database (SQLite3, or if you configured something else in the "Set up database" step above, select that and set the corresponding options)
- **Server Domain** should be `git.example.com` (or whatever you're actually using), **Forgejo Base URL** should be `http://git.example.com:3000` (assuming you won't change `HTTP_PORT` to a different value than 3000)
- Check the **Server and Third-Party Service Settings** settings for settings that look relevant for you.
- It may make sense to create the administrator account right now (**Administrator Account Settings**), even more so if you disabled self-registration.

- Most settings can be changed in `/etc/forgejo/app.ini` later, so don't worry about them too much.

Once you're done configuring, click `Install Forgejo` and a few seconds later you should be on the dashboard (if you created an administrator account) or at the login/register screen, where you can create an account to then get to the dashboard.

So far, so good, but we're not quite done yet - some manual configuration in the `app.ini` is needed.

# Further configuration in Forgejo's app.ini

Stop the forgejo service:

```
sudo systemctl stop forgejo.service
```

While at it, make `/etc/forgejo/` and the `app.ini` read-only for the git user (Forgejo doesn't write to it after the initial configuration):

```
sudo chmod 750 /etc/forgejo && sudo chmod 640 /etc/forgejo/app.ini
```

Now, (as root), edit `/etc/forgejo/app.ini`.

“ [!note] Note You'll probably find the [Configuration Cheat Sheet](#) and the [Example app.ini](#) that contains all options incl. descriptions helpful.

The following changes are recommended if dealing with many large files:

- Forgejo allows uploading files to Git repositories through the web interface. By default the **file size for uploads** is limited to 3MB per file, and 5 files at once. To increase it, under the `[repository]` section, add a `[repository.upload]` section with a line like `FILE_MAX_SIZE = 4095` (that would be 4095MB, about 4GB) and `MAX FILES = 20` It'll look somewhat like this:

```
...
[repository]
ROOT = /var/lib/forgejo/data/forgejo-repositories
```

```

[repository.upload]
;; max size for files to the repo via web interface, in MB,
;; defaults to 3 (this sets a limit of about 4GB)
FILE_MAX_SIZE = 4095
;; by default 5 files can be uploaded at once, increase to 20
MAX_FILES = 20

[server]
...

```

Similar restrictions exist for attachments to issues/pull requests, configured in the

`[attachment]` `sections` `MAX_SIZE` (default 4MB) and `MAX_FILES` (default 5) settings.

- By default **LFS data uploads expire** after 20 minutes - this can be too short for big files, slow connections or slow LFS storage (git-lfs seems to automatically restart the upload then - which means that it can take forever and use lots of traffic).. If you're going to use LFS with big uploads, increase this limit, by adding the line `LFS_HTTP_AUTH_EXPIRY = 180m` (for 180 minutes) to the `[server]` section.
- Similarly, there are timeouts for all kinds of git operations, that can be too short. Increasing all those git timeouts by adding a `[git.timeout]` section below the `[server]` section:

```

;; Git Operation timeout in seconds
;; increase the timeouts, so importing big repos (and presumably
;; pushing large files?) hopefully won't fail anymore
[git.timeout]
DEFAULT = 3600 ; Git operations default timeout seconds
MIGRATE = 6000 ; Migrate external repositories timeout seconds
MIRROR  = 3000 ; Mirror external repositories timeout seconds
CLONE   = 3000 ; Git clone from internal repositories timeout seconds
PULL    = 3000 ; Git pull from internal repositories timeout seconds
GC      = 600  ; Git repository GC timeout seconds

```

The example values are increased by a factor of 10 for illustration. Increasing all timeouts, particularly by this magnitude, may not be necessary. Evaluate your specific performance requirements and adjust values accordingly.

- By default LFS files are stored in the filesystem, in `/var/lib/forgejo/data/lfs`. In the `[lfs]` section you can change the `PATH = ...` line to store elsewhere, but you can also configure Forgejo to store the files in an S3-like Object-Storage.
- If you want to use the systemwide sendmail, enable sending E-Mails by changing the `[mailer]` section like this:

```
[mailer]
;; send mail with systemwide "sendmail"
ENABLED = true
PROTOCOL = sendmail
FROM = "Forgejo Git" <noreply@yourdomain.com>
```

- By default, Forgejo will listen on the port 3000, but that can be [changed to 80 with HTTP\\_PORT](#) like this:

```
[server]
HTTP_PORT = 80
```

When you're done editing the app.ini, save it and start the Forgejo service again:

```
sudo systemctl start forgejo.service
```

You can test sending a mail by clicking the user button on the upper right of the Forgejo page ("Profile and Settings"), then `Site Administration`, then `Configuration` and under `Mailer Configuration` type in your mail address and click `Send Testing Email`.

# General hints for using Forgejo

Sometimes you may want/need to use the Forgejo [command line interface](#). Keep in mind that:

- You need to **run it as the `git` user**, for example with:

```
$ sudo -u git forgejo command --argument
```

- You need to specify the **Forgejo work path**, either with the `--work-path /var/lib/forgejo` (or `-w /var/lib/forgejo`) commandline option or by setting the `FORGEJO_WORK_DIR` environment variable before calling `forgejo`:

```
$ export FORGEJO_WORK_DIR=/var/lib/forgejo
```

- You need to specify the path to the config (app.ini) with either `--config /etc/forgejo/app.ini` or `-c /etc/forgejo/app.ini`.

So all in all your command might look like:

```
$ sudo -u git forgejo -w /var/lib/forgejo -c /etc/forgejo/app.ini admin user list
```



**For convenience**, you could create a `/usr/local/bin/forgejo.sh` with the following contents:

```
#!/bin/sh  
sudo -u git forgejo -w /var/lib/forgejo -c /etc/forgejo/app.ini "$@"
```

and make it executable:

```
sudo chmod 755 /usr/local/bin/forgejo.sh
```

Now if you want to call `forgejo` on the commandline (for the default system-wide installation in `/var/lib/forgejo`), instead of the long line shown above, use:

```
$ forgejo.sh admin user list
```

You can always call `forgejo` and its subcommands with `-h` or `--help` to make it output usage information like available options and (sub)commands, for example to show available subcommands to administrate users on the commandline:

```
$ forgejo admin user -h
```

# Database Preparation

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 ( $\geq 10.6$ ), using the InnoDB storage engine
- MySQL ( $\geq 8.4$ ), using the InnoDB storage engine
- PostgreSQL ( $\geq 14$ )
- 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 `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
```

- Restart MySQL/MariaDB to apply the setting.
- 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.

- 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.
- 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](#).