Full docker example Vikunja

Link: https://vikunja.io/docs/full-docker-example/
Em 22/08/2026

Full docker example

This docker compose configuration will run Vikunja with a postgres database. It uses a proxy configuration to make it available under a domain.

For all available configuration options, see configuration.

After registering all your users, you might also want to disable the user registration.

If you intend to run Vikunja with MySQL or MariaDB and/or to use non-latin characters make sure your db is utf-8 compatible. All examples on this page use postgres and do not require additional work.

File permissions#

Vikunja runs as user `1000` and no group by default. You can use Docker's `--user` flag to change that.

You must ensure Vikunja is able to write into the `files` directory. To do this, create the folder and chown it before starting the stack:

```
mkdir $PWD/files
chown 1000 $PWD/files
```

You'll need to do this before running any of the examples on this page.

Vikunja will not try to acquire ownership of the files folder, as that would mean it had to run as root.

On macOS: Docker Desktop handles bind-mount permissions transparently, so the `chown` step is usually not needed — and will fail because the default macOS user has UID `501`, not `1000`. You can safely skip `chown 1000 $PWD/files` on macOS.

Rootless Docker#

If you're running Docker in rootless mode, file permissions work differently than with a regular Docker installation.

Rootless Docker uses user namespace remapping to map UIDs inside the container to different UIDs on the host. This means that even if you set `-u 1000:1000` and the files directory is owned by UID 1000 on the host, the process inside the container will actually run as a different UID on the kernel level. Because file permission checks happen at the kernel level, this causes `permission denied` errors when Vikunja tries to write to its files directory, even though the ownership appears to be correct.

To fix this, run Vikunja as root inside the container by setting the user to `0:0`:

```
services:
  vikunja:
    image: vikunja/vikunja
    user: "0:0"
```

This is safe because rootless Docker maps the container root user (UID 0) to your unprivileged host user. The process will not have any elevated privileges on the host.

You can use `vikunja doctor` to check if your Vikunja instance is affected by this issue. It will detect user namespace problems and suggest the fix.

MySQL / MariaDB#

Vikunja supports postgres, mysql and sqlite as a database backend. The examples on this page use postgres with a postgres container. To use MySQL or MariaDB as a database backend, change the `db` section of the examples to this:

```
db:
  image: mariadb:10
  command: --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci
  environment:
```

```
    MYSQL_ROOT_PASSWORD: supersecret
    MYSQL_USER: vikunja
    MYSQL_PASSWORD: changeme
    MYSQL_DATABASE: vikunja
  volumes:
    - ./db:/var/lib/mysql
  restart: unless-stopped
  healthcheck:
    test: ["CMD-SHELL", "mysqladmin ping -h localhost -u $$MYSQL_USER --
password=$$MYSQL_PASSWORD"]
    interval: 2s
    start_period: 30s
```

You'll also need to change the `VIKUNJA_DATABASE_TYPE` to `mysql` on the vikunja container declaration.

SQLite#

Vikunja supports postgres, mysql and sqlite as a database backend. The examples on this page use postgres with a postgres container. To use sqlite as a database backend, change the `vikunja` section of the examples to this:

```
vikunja:
  image: vikunja/vikunja
  environment:
    VIKUNJA_SERVICE_SECRET: <a super secure random secret>
    VIKUNJA_SERVICE_PUBLICURL: http://<your public frontend url with slash>/
    # The official image already presets this to /db/vikunja.db; it is shown
    # explicitly here so the database file lives in the ./db volume below and
    # state is persisted even if the container is destroyed.
    VIKUNJA_DATABASE_PATH: /db/vikunja.db
  ports:
    - 3456:3456
  volumes:
    - ./files:/app/vikunja/files
    - ./db:/db
  restart: unless-stopped
```

The official Vikunja image presets `VIKUNJA_DATABASE_PATH=/db/vikunja.db`, so the sqlite database file is stored at `/db/vikunja.db` inside the container by default. Mounting a volume at `/db` (as in the

example above) persists it across restarts. Note that because environment variables take precedence over the config file, setting `database.path` in a mounted `config.yml` will **not** move the database — change the `VIKUNJA_DATABASE_PATH` environment variable or the host side of the `/db` bind mount instead.

You'll also need to remove or change the `VIKUNJA_DATABASE_TYPE` to `sqlite` on the container declaration.

You can also remove the db section.

To run the container, you need to create the directories first and make sure they have all required permissions:

```
mkdir $PWD/files $PWD/db
chown 1000 $PWD/files $PWD/db
```

If you'll use your instance with more than a handful of users, we recommend using mysql or postgres.

Example without any proxy#

This example lets you host Vikunja without any reverse proxy in front of it. This is the absolute minimum configuration you need to get something up and running. If you want to make Vikunja available on a domain or need tls termination, check out one of the other examples.

Note that you need to change the `VIKUNJA_SERVICE_PUBLICURL` environment variable to the public ip or hostname including the port (the docker host you're running this on) is reachable at, prefixed with `http://`. Because the browser you'll use to access the Vikunja frontend uses that url to make the requests, it has to be able to reach it from the outside.

You must ensure Vikunja has write permissions on the `files` directory before starting the stack. To do this, check out the related commands here.

```
services:
  vikunja:
    image: vikunja/vikunja
    environment:
      VIKUNJA_SERVICE_PUBLICURL: http://<the public ip or host where Vikunja is reachable>
      VIKUNJA_DATABASE_HOST: db
      VIKUNJA_DATABASE_PASSWORD: changeme
```

```
      VIKUNJA_DATABASE_TYPE: postgres
      VIKUNJA_DATABASE_USER: vikunja
      VIKUNJA_DATABASE_DATABASE: vikunja
      VIKUNJA_SERVICE_SECRET: <a super secure random secret>
    ports:
      - 3456:3456
    volumes:
      - ./files:/app/vikunja/files
    depends_on:
      db:
        condition: service_healthy
    restart: unless-stopped
  db:
    image: postgres:18
    environment:
      POSTGRES_PASSWORD: changeme
      POSTGRES_USER: vikunja
    volumes:
      - ./db:/var/lib/postgresql
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -h localhost -U $$POSTGRES_USER"]
      interval: 2s
      start_period: 30s
```

Example with Traefik 2#

This example assumes traefik version 2 installed and configured to use docker as a configuration provider.

We also make a few assumptions here which you'll most likely need to adjust for your traefik setup:

- Your domain is `vikunja.example.com`
- The entrypoint you want to make Vikunja available from is called `https`
- The tls cert resolver is called `acme`

You must ensure Vikunja has write permissions on the `files` directory before starting the stack. To do this, check out the related commands here.

```
services:
  vikunja:
    image: vikunja/vikunja
    environment:
      VIKUNJA_SERVICE_PUBLICURL: https://<the public url where Vikunja is reachable>
      VIKUNJA_DATABASE_HOST: db
      VIKUNJA_DATABASE_PASSWORD: changeme
      VIKUNJA_DATABASE_TYPE: postgres
      VIKUNJA_DATABASE_USER: vikunja
      VIKUNJA_DATABASE_DATABASE: vikunja
      VIKUNJA_SERVICE_SECRET: <a super secure random secret>
    volumes:
      - ./files:/app/vikunja/files
    networks:
      - web
      - default
    depends_on:
      db:
        condition: service_healthy
    restart: unless-stopped
    labels:
      - "traefik.enable=true"
      - "traefik.docker.network=web"
      - "traefik.http.routers.vikunja.rule=Host(`vikunja.example.com`)"
      - "traefik.http.routers.vikunja.entrypoints=https"
      - "traefik.http.routers.vikunja.tls.certResolver=acme"
  db:
    image: postgres:18
    environment:
      POSTGRES_PASSWORD: changeme
      POSTGRES_USER: vikunja
    volumes:
      - ./db:/var/lib/postgresql
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -h localhost -U $$POSTGRES_USER"]
      interval: 2s
      start_period: 30s
```

```
networks:
  web:
    external: true
```

Example with Caddy v2 as proxy#

You will need the following `Caddyfile` on your host (or elsewhere, but then you'd need to adjust the proxy mount at the bottom of the compose file):

```
vikunja.example.com {
    reverse_proxy vikunja:3456
}
```

Note that you need to change the `VIKUNJA_SERVICE_PUBLICURL` environment variable to the ip (the docker host you're running this on) is reachable at. Because the browser you'll use to access the Vikunja frontend uses that url to make the requests, it has to be able to reach that ip + port from the outside.

You must ensure Vikunja has write permissions on the `files` directory before starting the stack. To do this, check out the related commands here.

Docker Compose config:

```
services:
  vikunja:
    image: vikunja/vikunja
    environment:
      VIKUNJA_SERVICE_PUBLICURL: https://<the public url where Vikunja is reachable>
      VIKUNJA_DATABASE_HOST: db
      VIKUNJA_DATABASE_PASSWORD: changeme
      VIKUNJA_DATABASE_TYPE: postgres
      VIKUNJA_DATABASE_USER: vikunja
      VIKUNJA_DATABASE_DATABASE: vikunja
      VIKUNJA_SERVICE_SECRET: <a super secure random secret>
    ports:
      - 3456:3456
    volumes:
      - ./files:/app/vikunja/files
```

```
    depends_on:
      db:
        condition: service_healthy
    restart: unless-stopped
  db:
    image: postgres:18
    environment:
      POSTGRES_PASSWORD: changeme
      POSTGRES_USER: vikunja
    volumes:
      - ./db:/var/lib/postgresql
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -h localhost -U $$POSTGRES_USER"]
      interval: 2s
      start_period: 30s
  caddy:
    image: caddy
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    depends_on:
      - vikunja
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile:ro
```

Setup on a Synology NAS#

There is a proxy preinstalled in DSM, so if you want to access Vikunja from outside, you need to prepare a proxy rule for the Vikunja Service.

Synology Proxy Settings

You should also add 2 empty folders for PostgreSQL and Vikunja inside Synology's docker main folders:

- Docker
  - vikunja

    - postgres

Synology has its own GUI for managing Docker containers, but it's easier via docker compose.

To do that, you can

- Either activate SSH and paste the adapted compose file in a terminal (using Putty or similar)
- Without activating SSH as a "custom script" (go to Control Panel / Task Scheduler / Create / Scheduled Task / User-defined script)
- Without activating SSH, by using Portainer (you have to install first, check out this tutorial for example):
  1. Go to **Dashboard / Stacks** click the button **"Add Stack"**
  2. Give it the name Vikunja and paste the adapted docker compose file
  3. Deploy the Stack with the "Deploy Stack" button:

Portainer Stack deploy

The docker-compose file we're going to use is exactly the same from the example without any proxy above.

You may want to change the volumes to match the rest of your setup.

After registering all your users, you might also want to disable the user registration.

You must ensure Vikunja has write permissions on the `files` directory before starting the stack. To do this, check out the related commands here.

Redis#

While Vikunja has support to use redis as a caching backend, you'll probably not need it unless you're using Vikunja with more than a handful of users.

To use redis, you'll need to add this to the config examples below:

```
services:
  vikunja:
    image: vikunja/vikunja
    environment:
      VIKUNJA_REDIS_ENABLED: 1
```

```
      VIKUNJA_REDIS_HOST: 'redis:6379'
      VIKUNJA_CACHE_ENABLED: 1
      VIKUNJA_CACHE_TYPE: redis
    volumes:
      - ./files:/app/vikunja/files
  redis:
    image: redis
```

ParadeDB (Full-text Search)#

ParadeDB provides enhanced full-text search capabilities for Vikunja. ParadeDB runs as a PostgreSQL extension, so you don't need a separate service.

To use ParadeDB, simply replace the standard PostgreSQL image with the ParadeDB image:

```
services:
  vikunja:
    image: vikunja/vikunja
    environment:
      VIKUNJA_SERVICE_PUBLICURL: http://<the public ip or host where Vikunja is reachable>
      VIKUNJA_DATABASE_HOST: db
      VIKUNJA_DATABASE_PASSWORD: changeme
      VIKUNJA_DATABASE_TYPE: postgres
      VIKUNJA_DATABASE_USER: vikunja
      VIKUNJA_DATABASE_DATABASE: vikunja
      VIKUNJA_SERVICE_SECRET: <a super secure random secret>
    ports:
      - 3456:3456
    volumes:
      - ./files:/app/vikunja/files
    depends_on:
      db:
        condition: service_healthy
    restart: unless-stopped
  db:
    image: paradedb/paradedb:latest
    environment:
      POSTGRES_PASSWORD: changeme
```

```
    POSTGRES_USER: vikunja
    POSTGRES_DB: vikunja
  volumes:
    - ./db:/var/lib/postgresql
  restart: unless-stopped
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -h localhost -U $$POSTGRES_USER"]
    interval: 2s
    start_period: 30s
```

Vikunja will automatically detect ParadeDB and enable enhanced search - no additional configuration required.

Revision #1
Created 2026-08-22 21:06:23 UTC by Administrador
Updated 2026-08-22 21:08:13 UTC by Administrador