Vikunja Docker Walkthrough

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

Docker Walkthrough

This tutorial assumes basic knowledge of Docker. It is aimed at beginners and should get you up and running quickly.

We'll use docker compose to make handling the bunch of containers easier.

> If you have any issues setting up Vikunja, please don't hesitate to reach out to us via matrix, the community forum or even email.

Getting a server to host Vikunja on

If you want the easy experience of just using Vikunja, we recommend our own hosted offering Vikunja Cloud. With Vikunja Cloud, we take care of everything and make it very easy for you, to sign up and use it, hassle-free.

If you want more control, you need to install and run Vikunja on a server. Since it's designed as a web application, it is not really possible to host it standalone on a desktop device only. If you know what you're doing, you can get it working but that's out of scope for this guide.

To get a server, we recommend one of these hosting providers (affiliate links):

- Hetzner - German provider, we're using them to host a bunch of Vikunja-related infrastructure. Sign up with our link to get 20 € in hosting credits.
- Digital Ocean - Well known cloud provider with a lot of options to choose from. Sign up with our link to get $ 200 USD in hosting credits.

All of these are affiliate links. If you sign up for one of these providers, we get a commission which helps support the project.

Preparations (optional)#

Create a directory for the project where all data and the compose file will live in.

Create all necessary files#

Create a `docker-compose.yml` file with the following contents in your directory:

```
services:
    vikunja:
        image: vikunja/vikunja
        environment:
            VIKUNJA_SERVICE_PUBLICURL: http://<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
```

This defines two services, each with their own container:

- A Vikunja service which runs the Vikunja API and hosts its frontend.
- A database container which will store all projects, tasks, etc. We're using PostgreSQL here, but you're free to use MySQL or MariaDB if you want.

If you already have a proxy on your host, you may want to check out the reverse proxy examples to use that. By default, Vikunja will be exposed on port 3456 on the host.

To change to something different, you'll need to change the `ports` section in the service definition. The number before the colon is the host port - This is where you can reach Vikunja from the outside once all is up and running.

You'll need to change the value of the `VIKUNJA_SERVICE_PUBLICURL` environment variable to the public URL where Vikunja is reachable. If you're accessing Vikunja on a non-standard port (for example via `http://localhost:3456/`), make sure the port is included in the URL — otherwise you'll get an "unauthorized" error when creating your user account.

Ensure adequate file permissions#

Vikunja runs as user `1000` and no group by default.

To be able to upload task attachments or change the background of a project, Vikunja must be 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
```

If you're using rootless Docker, see the dedicated section about file permissions in that setup.

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.

Vikunja's Docker image is based on a `scratch` image. This means:

There is no shell inside the container. Commands like `docker exec -it vikunja sh` will not work. Use `docker logs vikunja` to check for errors instead.

`PUID` and `PGID` environment variables are not supported. These are conventions from other images (like LinuxServer.io) but have no effect on the Vikunja container. To change the user the container runs as, use Docker's `user:` directive in your compose file.

The `/tmp` directory must be writable for data exports to work. If you use a custom user, ensure `/tmp` is accessible or mount a writable volume at `/tmp`.

Run it#

Run `sudo docker compose up` in your directory and take a look at the output you get. When first started, Vikunja will set up the database and run all migrations etc. Once it is ready, you should see a message like this one in your console:

```
vikunja_1       | 2024-02-09T14:44:06.990677157+01:00: INFO      ▶ cmd/func29 05d Vikunja
version 0.23.0
vikunja_1       | ⇨ http server started on [::]:3456
```

This indicates all setup has been successful. If you get any errors, see below:

Troubleshooting#

Vikunja might not run on the first try. There are a few potential issues that could be causing this.

No connection to the database#

Indicated by an error message like this one from the api container:

```
2020/05/23 15:37:59 Config File "config" Not Found in "[/app/vikunja /etc/vikunja
/app/vikunja/.config/vikunja]"
2020/05/23 15:37:59 Using default config.
```

```
2020-05-23T15:37:59.974435725Z: CRITICAL▶ migration/Migrate 002 Migration failed: dial tcp
172.19.0.2:3306: connect: connection refused
```

Especially when using PostgreSQL, this can happen on first start, because the PostgreSQL database container will take a few seconds to start. Vikunja does not know the container is not ready, therefore it will just try to connect to the db, fail since it is not ready and exit.

If you're using the docker compose example from above, you may notice the `restart: unless-stopped` option at the api service. This tells docker to restart the api container if it exits, unless you explicitly stop it. Therefore, it should "magically fix itself" by automatically restarting the container.

After a few seconds (or minutes) you should see a log message like this one from the PostgreSQL container:

```
2024-01-01 00:00:00.000 UTC [1] LOG:  database system is ready to accept connections
```

The next restart of Vikunja should be successful. If not, there might be a different error or a bug with Vikunja, please reach out to us in that case.

(If you have an idea about how we could improve this, we'd like to hear it!)

Migration failed: commands out of sync#

If you get an error like this one:

```
2020/05/23 15:53:38 Config File "config" Not Found in "[/app/vikunja /etc/vikunja
/app/vikunja/.config/vikunja]"
2020/05/23 15:53:38 Using default config.
2020-05-23T15:53:38.762747276Z: CRITICAL▶ migration/Migrate 002 Migration failed: commands out
of sync. Did you run multiple statements at once?
```

This is a MySQL issue. Currently, we don't have a better solution than to completely wipe the database files and start over. To do this, first stop everything by running `sudo docker compose down`, then remove the `db/` folder in your current folder with `sudo rm -rf db` and start the whole stack again with `sudo docker compose up -d`.

Permission denied on file upload#

If you see an error like:

```
open /app/vikunja/files/1: permission denied
```

This means the `files` directory is not writable by the Vikunja process. Make sure you followed the file permission steps above.

Common mistakes:

- Forgetting to `chown` the directory to UID `1000` before starting the container
- Using `PUID`/`PGID` environment variables (these have no effect — use Docker's `user:` directive instead)
- Not using the `-R` flag when running `chown` on a directory that already contains files: `chown -R 1000 $PWD/files`

Data export fails with permission denied#

If data export fails with an error like:

```
mkdir /app/vikunja/files/user-export-tmp/: permission denied
```

or:

```
open /tmp/vikunja-export-*.zip: no such file or directory
```

This is the same permissions issue. Make sure the `files` volume is writable. If you see the `/tmp` error, your container may lack a writable `/tmp` directory — mount one in your compose file:

```
volumes:
  - /tmp:/tmp
```

"/.cache: permission denied" warning#

If you see this warning at startup:

```
failed to create modcache index dir: mkdir /.cache: permission denied
```

This is a known Go runtime warning and is harmless. Vikunja will work correctly despite this message.

Database hostname mismatch#

If you see an error like:

```
Migration failed: dial tcp: lookup db on 127.0.0.11:53: server misbehaving
```

Make sure the `VIKUNJA_DATABASE_HOST` value matches the service name of your database container in `docker-compose.yml`. For example, if your database service is called `db`, the host must be `db`. If you renamed the service to `postgres`, update the environment variable to match.

Email test fails with “not connected to SMTP server”#

If the `testmail` command fails with this error but you believe your SMTP settings are correct, note that Vikunja sends an SMTP `NOOP` command to verify the connection. Some minimal SMTP relays (like `msmtpd`) do not support this command. Use a full-featured SMTP server or relay instead.

Also ensure your SMTP configuration is defined in **one place only** — either environment variables or a config file. If both are set, the environment variable takes precedence, which can cause confusion if they have different values.

Try it#

Head over to `http://<host-ip or url>/api/v1/info` in a browser. You should see something like this:


```
{
  "version": "v0.23.0",
  "frontend_url": "https://try.vikunja.io/",
  "motd": "",
  "link_sharing_enabled": true,
  "max_file_size": "20MB",
  "registration_enabled": true,
  "available_migrators": [
    "vikunja-file",
    "ticktick",
    "todoist"
  ],
  "task_attachments_enabled": true,
  "enabled_background_providers": [
    "upload",
    "unsplash"
  ],
  "totp_enabled": false,
  "legal": {
    "imprint_url": "",
    "privacy_policy_url": ""
```

```
  },
  "caldav_enabled": true,
  "auth": {
    "local": {
      "enabled": true
    },
    "openid_connect": {
      "enabled": false,
      "providers": null
    }
  },
  "email_reminders_enabled": true,
  "user_deletion_enabled": true,
  "task_comments_enabled": true,
  "demo_mode_enabled": true,
  "webhooks_enabled": true
}
```

This shows you can reach the API through the API proxy.

Now head over to `http://<host-ip or url>/` which should show the login mask.

Make it persistent#

Currently, Vikunja runs in foreground in your terminal. For a real-world scenario this is not the best way.

Back in your terminal, stop the stack by pressing `CTRL-C` on your keyboard. Then run `sudo docker compose up -d` in your terminal again. The `-d` flag at the end of the command will tell docker to run the containers in the background. If you need to check the logs after that, you can run `sudo docker compose logs`.

Vikunja does not have any default users, you'll need to register an account. After that, you can use it.

Tear it all down#

If you want to completely stop all containers run `sudo docker compose down` in your terminal.

Improve this guide#

We'll happily accept suggestions and improvements for this guide. Please reach out to us if you have any.

Revision #1
Created 2026-08-22 21:04:19 UTC by Administrador
Updated 2026-08-22 21:05:52 UTC by Administrador