Instalação e Configuração Vikunja Docker

- Vikunja Docker Walkthrough
- Full docker example Vikunja
- OpenID example configurations Vikunja

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.

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.

OpenID example configurations Vikunja

Link: https://vikunja.io/docs/openid-example-configurations/
EM 22/08/2026

On this page you will find examples about how to set up Vikunja with a third-party OAuth 2.0 provider using OpenID Connect. To add another example, please edit this document and send a PR.

Important: Redirect URL Format The redirect URL format is: `https://vikunja.mydomain.com/auth/openid/<provider-id>` Use the provider ID (the key in your config) as the identifier. For example, if your config uses `authentiklogin:` as the provider key, the redirect URL would be `/auth/openid/authentiklogin`.

Authelia#

Vikunja Config:

```
auth:
  openid:
    enabled: true
    providers:
      authelia:
        name: Authelia
        authurl: https://login.mydomain.com
        clientid: <vikunja-id>
        clientsecret: <vikunja secret>
```

Authelia config:

```
- client_id: <vikunja-id>
  client_name: Vikunja
  client_secret: <vikunja secret>
  redirect_uris:
```

```
    - https://vikunja.mydomain.com/auth/openid/authelia
  scopes:
    - openid
    - email
    - profile
```

Also see the Authelia documentation.

Google / Google Workspace#

Vikunja Config:

```
auth:
  openid:
    enabled: true
    providers:
      google:
        name: Google
        authurl: https://accounts.google.com
        clientid: <google-oauth-client-id>
        clientsecret: <google-oauth-client-secret>
```

Google config:

- Navigate to `https://console.cloud.google.com/apis/credentials` in the target project
- Create a new OAuth client ID
- Configure an authorized redirect URI of `https://vikunja.mydomain.com/auth/openid/google`

Note that there currently seems to be no way to stop creation of new users, even when `enableregistration` is `false` in the configuration. This means that this approach works well only with an “Internal Organization” app for Google Workspace, which limits the allowed users to organizational accounts only. External / public applications will potentially allow every Google user to register.

Keycloak#

Vikunja Config:

```
auth:
  openid:
    enabled: true
    providers:
      keycloak:
        name: Keycloak
        authurl: https://keycloak.mydomain.com/realms/<realm-name>
        logouturl: https://keycloak.mydomain.com/realms/<realm-name>/protocol/openid-
connect/logout
        clientid: <vikunja-id>
        clientsecret: <vikunja secret>
```

Keycloak Config:

- Navigate to the keycloak instance
- Create a new client with the type `OpenID Connect`, add a unique `Client ID`.
- Set `Client authentication` to On
- Set `Root Url` to `https://vikunja.mydomain.com`
- Set `Valid redirect URIs` to `/auth/openid/keycloak`
- Create the client then navigate to the credentials tab and copy the `Client secret`

Authentik#

Authentik Config:

- Create a new Provider called “Vikunja” in Authentik
- Set the `Redirect URIs/Origins (RegEx)` to `https://vikunja.mydomain.com/auth/openid/authentik` (This matches the `name: authentik` in the Vikunja config below)
- Copy the Client ID and Client Secret

Vikunja Config:

```
auth:
  openid:
    enabled: true
    providers:
      authentik:
        name: authentik
        authurl: "https://authentik.mydomain.com/application/o/vikunja/"
```

```
        logouturl: "https://authentik.mydomain.com/application/o/vikunja/end-session/"
        clientid: "" # copy from Authentik
        clientsecret: "" # copy from Authentik
```

Note: The `authurl` that Vikunja requires is not the `Authorize URL` that you can see in the Provider. OpenID Discovery is used to find the correct endpoint to use automatically, by accessing the `OpenID Configuration URL` (usually `https://authentik.mydomain.com/application/o/vikunja/.well-known/openid-configuration`). Use this URL without the `.well-known/openid-configuration` as the `authurl`. Typically, this URL can be found in the metadata section within your identity provider.

Azure Entra ID#

Vikunja Config:

```
auth:
  openid:
    enabled: true
    providers:
      azuread:
        name: AzureAD
        authurl: https://sts.windows.net/<tenant_id>/
        clientid: <azure-client-id>
        clientsecret: <azure-client-secret>
```

Azure AD Config:

- Navigate to the Azure Portal and create a new App Registration
- Set the Redirect URI to `https://vikunja.mydomain.com/auth/openid/azuread`
- Under API permissions, add the following delegated API permissions:
  - email
  - openid
  - profile
  - User.Read
- Create a new client secret and copy its value
- In Token configuration, add an optional claim:
  - Select ID, then email

Note: Replace `<tenant_id>` in the `authurl` with your Azure AD tenant ID. Ensure that the provider name in the Vikunja config matches the one used in the redirect URI (e.g., “azuread” in this example).

Dex#

Dex config:

```
staticClients:
- id: <vikunja-id>
  redirectURIs:
    - 'https://vikunja.mydomain.com/auth/openid/dex'
  name: 'Vikunja'
  secret: <dex-client-secret>
```

Note: All scopes required by Vikunja are already in the default scope definition of Dex (see Dex docs).

Vikunja config:

```
auth:
  openid:
    enabled: true
    providers:
      dex:
        name: dex
        authurl: <dex-url>
        clientid: <vikunja-id>
        clientsecret: <dex-client-secret>
        scope: openid profile email
        forceuserinfo: false
```