AI-03959 FileBrowser Setup in GCP

Owner	Ⓡ Rigel Arcayan
Tags	Project
Created time	June 21, 2026 12:36 AM

What is FileBrowser? How to install Filebrowser in Docker GCP VM. I have an NPM and a bridge network - dockerlab-net. Provide a step by step procedure on how to setup Filebrowser using the docker-compose.yml method. Also provide the NPM Proxy instructions.

What is FileBrowser?

FileBrowser is a lightweight, self-hosted web-based file manager. It essentially turns a directory on your server into a sleek, accessible cloud storage interface. It allows you to upload, download, delete, preview, and edit files directly from your web browser without needing to rely on heavier alternatives like Nextcloud or standard FTP.

Key features include:

- **Low resource footprint** (written in Go).
- **User management** with customizable permissions (read-only, admin, file creation, etc.).
- **Built-in text editor** and media player.
- **Shell command execution** (optional, for advanced users).

Prerequisites

Before starting, ensure you have:

1. A **GCP VM instance** running with Docker and Docker Compose installed.
2. An existing custom bridge network named `dockerlab-net`.
3. **Nginx Proxy Manager (NPM)** running on that same network or accessible by it.

Step 1: Prepare the Host Directory Structure

Connect to your GCP VM via SSH and set up the directories where FileBrowser will store its configuration data, database, and the actual files you want to manage.

```
mkdir -p ~/dockerlab/filebrowser/config
mkdir -p ~/dockerlab/filebrowser/data
```

Next, create an empty database file. FileBrowser requires this file to exist before boot, or it might accidentally create a directory instead of a file.

```
touch ~/dockerlab/filebrowser/config/filebrowser.db
```

Step 2: Create the docker-compose.yml File

Navigate to your FileBrowser directory:

```
cd ~/dockerlab/filebrowser
```

Create and edit the `docker-compose.yml` file:

```
nano docker-compose.yml
```

Paste the following configuration into the file:

```
version: '3.8'

services:
  filebrowser:
    image: filebrowser/filebrowser:latest
    container_name: filebrowser
    user: "${UID}:${GID}" # Runs container as your current user to avoid permission issues
    restart: unless-stopped
    networks:
      - dockerlab-net
    volumes:
      - /~/dockerlab/filebrowser/data:/srv
      - /~/dockerlab/filebrowser/config/filebrowser.db:/database/filebrowser.db
      - /~/dockerlab/filebrowser/config/settings.json:/config/settings.json
    environment:
      - PUID=1000  # Adjust to match your user ID if not using the user tag
      - PGID=1000  # Adjust to match your group ID if not using the user tag

networks:
  dockerlab-net:
    external: true
```

> **Note on Volumes:** > * `/srv` inside the container maps to your files (`~/dockerlab/filebrowser/data`). Anything you put in this host folder will appear in the FileBrowser UI.
>
> - If you want FileBrowser to access other parts of your VM or media drives, change the host path side of the mapping (e.g., `/home/user:/srv`).

Save and exit (`Ctrl+O`, `Enter`, `Ctrl+X`).

Step 3: Deploy the Container

Launch the FileBrowser container in detached mode:

```
docker compose up -d
```

Verify that the container is running successfully and connected to your network:

```
docker ps
```

Step 4: Nginx Proxy Manager (NPM) Configuration

Since FileBrowser is attached to the `dockerlab-net` bridge network alongside NPM, NPM can route traffic directly using the container name as the hostname.

1. Log into your **Nginx Proxy Manager** admin dashboard.
2. Navigate to **Hosts** > **Proxy Hosts** and click **Add Proxy Host**.
3. Configure the **Details** tab:

- **Domain Names:** Enter your domain or subdomain (e.g., `files.yourdomain.com`).
- **Scheme:** `http`
- **Forward Hostname / IP:** `filebrowser` (This resolves because both are on `dockerlab-net`).
- **Forward Port:** `80` (The default internal port for FileBrowser).
- **Block Common Exploits:** Enabled.
- **Websockets Support:** Enabled (highly recommended for smoother file streaming/uploads).

1. Configure the **SSL** tab:

- Select **Request a new SSL Certificate** from Let's Encrypt.
- Check **Force SSL** and **HTTP/2 Support**.
- Agree to the Let's Encrypt Terms of Service.

1. Click **Save**.

Step 5: Initial Login and Hardening

1. Navigate to your configured subdomain (e.g., `https://files.yourdomain.com`).
2. Log in using the default credentials:

- **Username:** `admin`
- **Password:** `admin`

1. **CRITICAL STEP:** Immediately go to **Settings** > **User Management**, edit the admin user, and change the default password to a strong, secure one.

Explain which id to use

rarcayan@wordpress-docker-vm:~$iduid = 1001(rarcayan)gid = 1002(rarcayan)groups = 1002(rarcayan), 4(adm), 20(dialout), 24(cdrom), 25(floppy), 29(audio), 30(dip), 44(video), 46(plugdev), 105(lxd) sudoers)rarcayan@wordpress - docker - vm$:

Based on your terminal output, you should use the IDs assigned to your active logged-in user, `rarcayan`.

Here are the specific numbers you need to use:

- `PUID = 1001` (corresponds to your `uid=1001(rarcayan)`)
- `PGID = 1002` (corresponds to your `gid=1002(rarcayan)`)

Why these IDs matter

When FileBrowser creates or moves files on your GCP VM, it maps its internal container permissions to these numbers. By using `1001` and `1002`, any file you upload via the web browser interface will be owned by `rarcayan` on the host system.

If you were to leave the default `1000` inside the configuration, the files would be owned by the `ubuntu` system user instead. This would trigger permission-denied errors whenever you tried to delete, edit, or move those files using your regular SSH account.

Update your `docker-compose.yml`

Go back into your `docker-compose.yml` file and update the `environment` section to look exactly like this:

YAML

```
environment:
  - PUID=1001
  - PGID=1002
```

(Note: If you are using the `user: "${UID}:${GID}"` directive at the top of your service file, it will dynamically fetch these values automatically, but setting the environment variables explicitly ensures absolute compatibility with the underlying Linux image).

Apply the changes

After saving the file, apply the updated IDs by recreating the container:

Bash

```
docker compose up -d --force-recreate
```