FGPE+ Platform
Installation Manual

Version 1.0

Written by: Filip Miernik

Supervised by: Jakub Swacha

Date: 2023-05-30

1. Prerequisites

This guide is intended to explain how to set up your own FGPE platform, using both Linux (Debian) and Docker.

Minimum technical requirements:

- Linux Server with network access and minimum 2GB of RAM and 20GB of free disk space (Debian GNU/Linux 11 bullseye or Ubuntu 18.04.4 LTS),
- web server software such as Nginx (version 1.14.0),
- Docker (version 23.0.3, build 3e7cbfd).

First, you'll need to install Docker - detailed instructions for this can be found in the official Docker documentation, which covers various systems. Along with this, if you're planning to use the platform beyond your local network, ensure that your server is publicly accessible. Additionally, if you wish for users to have a memorable address instead of an IP, you'll need to set up a domain.

Next, you'll need a reverse proxy tool such as Nginx or Apache to establish an HTTPS connection. It's also important to monitor your server's hard drive space regularly to provide the best user experience. The more users you accommodate, the larger the hard drive space required to store all the data. Mooshak, the default code evaluation engine, tends to consume a significant amount of hard drive space.

For ensuring your confirmation emails don't land in spam folders, you'll also need an email server. Your IT administration should be able to provide you with the necessary authorization data for the email server.

Here’s a small summary of the prerequisites section:

1. Make sure that you have a stable secure connection between your server and your server is available publicly.
2. Install Docker.
3. Install NGINX or other reverse proxy tool.
4. Point your custom domain to your server and configure NGINX.
5. Set up an e-mail server.

This guide also outlines how to run Mooshak or other components separately on various servers, which can sometimes simplify maintaining the entire platform. However, this step isn't mandatory, and the platform can function effectively using just one server.

This document also includes a few handy commands specifically for the **Debian** Linux distribution, which you may find helpful.

2. Server setup

Let's start with setting up the server. Ensure that you have a reliable and secure connection that is always accessible. Certain servers necessitate the configuration of iptables policy. Here are some useful commands to help you with that:

- `sudo iptables --policy INPUT ACCEPT`
- `sudo apt-get install iptables-persistent`
- `sudo iptables-save > /etc/iptables/rules.v4`
- `sudo sh -c "iptables-save > /etc/iptables/rules.v4"`

Next, proceed with setting up Docker and Nginx. For detailed instructions, please refer to the official Docker and Nginx documentations. Verify whether your reverse proxy tool interacts effectively with your firewall. For this, you can use the Linux ufw tool (use `sudo systemctl status ufw` to check status). Here are some helpful commands:

- `sudo apt install nginx`
- `sudo ufw allow "Nginx Full"`
- `sudo ufw status`
- `sudo systemctl restart nginx`
- `sudo nginx -t` (This command is used to check the Nginx configuration).

We recommend using Nginx as we have tested the platform with this tool. However, keep in mind that you'll need to set up Nginx correctly to make your platform publicly accessible.

Towards the end of this document, you'll find a sample Nginx configuration that could be of help.

3. Platform setup

To start, clone the main Git repository from this location: https://github.com/FGPE-Erasmus/framework-docker to your server. You'll need to adjust the `.env` files to suit your needs. Each component of the platform has its own .env file. You can then initiate the platform using the command `docker compose up -d`.

Docker will download all necessary images and prepare everything for you. Note that it's always possible to delete all containers and restart the platform without losing any data. This is due to Docker volumes handling the data, and Docker containers being disposable, allowing you to experiment with the configuration freely. If you wish to fully reset the entire platform, including all created databases and files, you'll also need to remove the Docker volumes.

Please refer to the official Docker documentation for further information. Pay special attention to the Docker Compose section to comprehend how the Dockerfile and docker-compose.yml files operate.

After the platform has initiated, you can open the Keycloak server to finalize the configuration of emails and admin access. The default admin account password can be found in the Keycloak .env file (it's set to 'password' by default unless you've modified this config file). By default keycloak is available under the `/auth` url.

⚠ Before making the platform public, please also change the number of available *inodes* on your server using this command:

```
sysctl -w fs.inotify.max_user_watches=100000
```

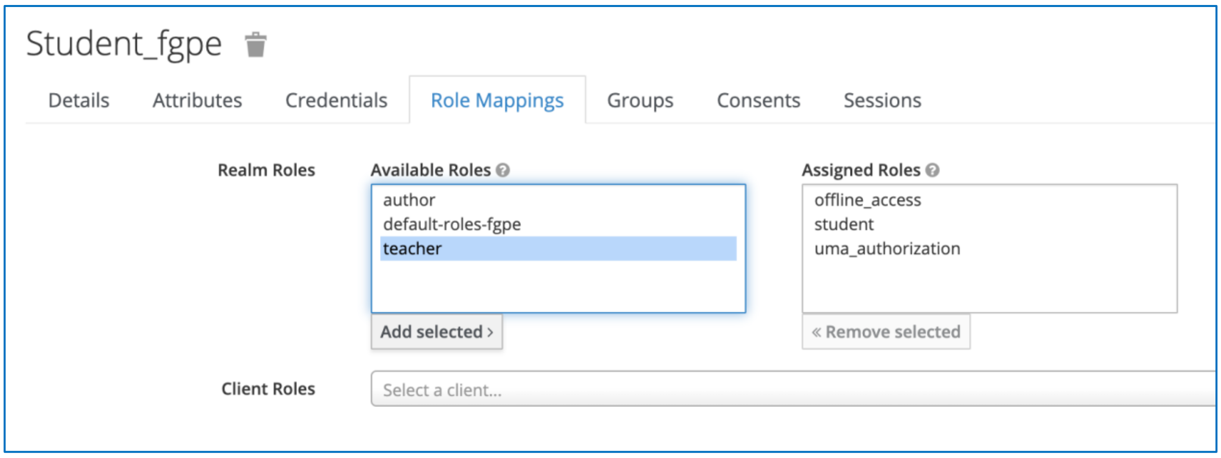
The platform is almost ready – you should now open Keycloak to set it up to your needs (you don't need to make any changes to Keycloak, but it's strongly recommended).

The main user interface of the platform (PLE) should become accessible once Keycloak launches successfully. To ensure the platform is functioning correctly, you can check the following test accounts:

- Default student test account: username - student_fgpe, password - student123
- Default teacher test account: username - teacher_fgpe, password - teacher123

It's important to note that Authorkit is a separate platform and isn't linked to PLE.

Both PLE teachers and students can register using the same Keycloak form. This form can be accessed by clicking the login button located at the top right corner of PLE. To assign a user the *teacher* role, you need to locate this user in the Keycloak dashboard and add them to the *teacher* group.


Student_fgpe

Realm Roles

Available Roles: author, default-roles-fgpe, teacher

Assigned Roles: offline_access, student, uma_authorization

Client Roles: Select a client...

Keycloak Users panel – a test account student_fgpe. Select "teacher" and then "Add selected" to add this account to the teacher group.

4. Custom platform configuration

You're free to experiment with the configuration file of each component (the `.env` and `docker-compose.yml` files) and tailor the entire platform to your preferences. A common use case involves separating components - running certain components on distinct servers.

For example, if you wish to run only the Mooshak component, you can modify the docker-compose.yml file by excluding all other components. The rest of the procedure remains unchanged - simply initiate everything using docker-compose (in this specific case, only the Mooshak component would run).

Don't forget to edit the files on other servers by excluding Mooshak from the docker-compose and modifying the .env files by changing the Mooshak IP.

5. Custom component images

This section is intended for those who are more experienced with Docker. Occasionally, you may find the need to alter the functionality of certain platform components. For example, you wish to add a completely new page to the PLE component - it would require you to edit the source code and create a brand-new image for it. You can definitely do this – the platform setup is versatile and allows you to use your images, even without access to the official FGPE Docker images repository, instead of the official ones provided by the project creators.

Each component of the platform has its own distinct Git repository, which you can fork or copy and modify on your own. You can find all repositories at this location: https://github.com/orgs/FGPE-Erasmus/repositories?type=all.

Once you're done with your setup, use `Dockerfile` files to construct a new image and tag it for easy identification. Subsequently, modify the main `docker-compose.yml` file to use your custom image instead of the official one. For instance, suppose your custom image name is 'ple-custom'. By default, the PLE component uses the 'fgpe/learning-platform' image. You can simply alter the `docker-compose.yml learning-platform` section to use your own image instead of the official one by changing 'fgpe/learning-platform' to 'ple-custom'. Please note that you need to have the custom image available on your server.

6. Updating the platform

Upgrades and improvements are a natural part of a project's life cycle. You may have to upgrade the platform or its components occasionally, especially if you create your own custom images. Docker-compose makes this process quite easy. You just need to pull the latest images from the Docker registry and restart the platform. This can be done using the following commands (please refer to the official Docker documentation, these commands can be different):

- `docker-compose pull`
- `docker-compose down`
- `docker-compose up -d`

Note: You should be cautious while upgrading as new images may require new environment variables or configurations. Always backup your current setup and data before performing an upgrade.

7. Backup and Restore

Always have a backup strategy in place. The data generated and processed by your platform is valuable. Regular backups help to mitigate the risk of data loss. Docker volumes can be backed up using various methods. Choose one that suits your deployment scenario.

In case of any catastrophic failure, you need to restore your application from the backups. Document the steps required for a full restoration of the platform.

Here's a small step-by-step instruction on how to create a backup of the Mooshak volume and move it to another server:

1. First, run the command to create a backup of the Mooshak volume using Docker. In this command, 'busybox' is a small utility Docker image, '37386205cc91' is the container ID of the Mooshak container, and '/home/mooshak' is the directory in the Mooshak container you want to backup.

   ```
   docker run --volumes-from [container_id] -v $(pwd):/backup busybox tar cvf /backup/backup.tar /home/mooshak
   ```

   Remember to replace [container_id] with the actual ID of your Mooshak container.

2. Next, securely copy (scp) the backup file to a destination server.

   ```
   scp backup.tar [user]@[destination-server]:/[path/to/destination]
   ```

   Here, replace [user] with your username, [destination-server] with the destination server's address, and [path/to/destination] with the directory where you want to save the backup.

3. Additionally, you may want to copy the backup file to another location. For instance, to copy it to your directory on a different server, use this command:

   ```
   scp backup.tar [username]@[server-address]:/
   ```

   Make sure to replace [username] and [server-address] with your username and server's address.

4. To copy the backup file from the server to your local machine, use the following command:

   ```
   scp [username]@[server-address]:~/backup.tar ./
   ```

   As before, replace [username] and [server-address] with your actual username and the server's address.

5. In case your server uses a non-standard SSH port (other than 22), use the -P option followed by the port number. This command copies the local backup file to your directory on a different server with a non-standard port:

   ```
   scp -P [port_number] ./backup.tar [username]@[server-address]:~/
   ```

   Don't forget to replace [port_number], [username], and [server-address] with the actual port number, your username, and the server's address.
6. Lastly, use the tar archive inside Mooshak container using built-in linux utilities. Mooshak container is based on linux image, and you can easily copy files between the host and container. Place all data in the same directory: `/home/mooshak`. The container must be running during this operation. You can use following commands:

   ```
   docker cp [OPTIONS] SRC_PATH CONTAINER:DEST_PATH
   docker exec -it [container_id] bash
   ```

Remember to maintain the confidentiality of your data during these operations. Only replace the placeholders with actual values when running the commands.

8. Troubleshooting

We strongly recommend creating your own documentation since every system is unique, especially if you encounter any problems. Docker logs is an excellent place to start with diagnosing problems. Every container has a log system (integrated with Docker) that can help you identify the problem. You can access the logs of a Docker container using the following command:

- `docker logs [container_id]`

The [container_id] is a placeholder which should be replaced with the actual ID of the container whose logs you want to view.

The Docker community and official Docker documentation are excellent resources for resolving common issues.

Appendix A. Sample nginx config

This NGINX configuration defines several upstream servers for different services. Each upstream server represents a backend service running on a specific port on the local machine. The server block listens on port 443 for HTTPS connections and includes SSL/TLS configuration.

Inside the server block, there are various location blocks that handle different URL paths. The proxy_pass directive is used to proxy requests to the corresponding upstream servers. The rewrite directive is used to modify the URL path before passing it to the upstream server.

Additionally, there are configuration settings for timeouts, buffer sizes, logging, and access control. The configuration also includes specific rules for handling requests to favicon.ico and robots.txt.

Finally, there's a rule to redirect requests to the root URL (/) to the *learning-platform* and a rule to deny access to any file or directory starting with a dot.

Please note that this is a simplified explanation, and NGINX is a powerful web server with many more features and directives.

```
# Define upstream servers for each service
upstream keycloak {
  server 127.0.0.1:8081;
}

upstream authorkit-api {
  server 127.0.0.1:8082;
}

upstream authorkit-ui {
  server 127.0.0.1:8083;
}

upstream gamification-service {
  server 127.0.0.1:8084;
}

upstream evaluation-engine {
  server 127.0.0.1:8085;
}

upstream learning-platform {
  server 127.0.0.1:8086;
}

upstream pwa {
  server 127.0.0.1:8088;
}

# Define the server block
server {
  listen 443 ssl http2;
  listen [::]:443 ssl http2;
  include ssl.conf;  # Include SSL/TLS configuration

  index index.html index.htm index.nginx-debian.html;

  server_name _;  # Server name wildcard to match any host

  # Set timeout values for client requests
  client_header_timeout 3000;
  client_body_timeout 3000;
  fastcgi_read_timeout 3000;

  # Set maximum allowed client body size and buffer sizes
  client_max_body_size 32m;
  fastcgi_buffers 8 128k;
  fastcgi_buffer_size 128k;

  # Disable logging and access for favicon.ico
  location = /favicon.ico {
    log_not_found off;
    access_log off;
  }

  # Allow access to robots.txt
  location = /robots.txt {
    allow all;
    log_not_found off;
    access_log off;
  }

  # Handle requests to the /auth path
  location /auth {
    proxy_pass          http://keycloak;  # Proxy requests to the keycloak
upstream server
    proxy_buffering off;
    proxy_set_header    Referer            $http_referer;
    proxy_set_header    Host               $http_host;
    proxy_set_header    X-Real-IP          $remote_addr;
    proxy_set_header    X-Forwarded-For    $proxy_add_x_forwarded_for;
    proxy_set_header    X-Forwarded-Port   $server_port;
    proxy_set_header    X-Forwarded-Proto  $scheme;
  }

  # Handle requests to the /authorkit/api path
  location /authorkit/api {
    proxy_pass          http://authorkit-api;  # Proxy requests to the
authorkit-api upstream server
    rewrite             ^/authorkit/api/(.*) /$1 break;
  }

  # Handle requests to the /authorkit/ui path
  location /authorkit/ui {
    proxy_pass          http://authorkit-ui;  # Proxy requests to the
authorkit-ui upstream server
    rewrite             ^/authorkit/ui/(.*) /$1 break;
  }

  # Handle requests to the /mooshak path
  location /mooshak {
    proxy_pass          http://evaluation-engine;  # Proxy requests to the
evaluation-engine upstream server
```

```
    proxy_set_header    Referer             $http_referer;
    proxy_set_header    Host                $http_host;
    proxy_set_header    X-Real-IP           $remote_addr;
    proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;
    proxy_set_header    X-Forwarded-Port    $server_port;
    proxy_set_header    X-Forwarded-Proto   $scheme;
  }

  # Handle requests to the /gamification-service path
  location /gamification-service {
    proxy_pass          http://gamification-service;  # Proxy requests to
the gamification-service upstream server
    rewrite             ^/gamification-service/(.*) /$1 break;
    proxy_set_header    upgrade             $http_upgrade;
    proxy_set_header    Connection          "upgrade";
    proxy_set_header    Referer             $http_referer;
    proxy_set_header    Host                $http_host;
    proxy_set_header    X-Real-IP           $remote_addr;
    proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;


    proxy_set_header    X-Frame-Options     SAMEORIGIN;
  }

  # Handle requests to the /learning-platform path
  location /learning-platform {
    proxy_pass          http://learning-platform;  # Proxy requests to the
learning-platform upstream server
    rewrite             ^/learning-platform/(.*) /$1 break;
  }

  # Handle requests to the /pwa path
  location /pwa {
    proxy_pass          http://pwa;  # Proxy requests to the pwa upstream
server
    rewrite             ^/pwa/(.*) /$1 break;
    add_header Service-Worker-Allowed /pwa/;  # Add a header to allow
service workers in the /pwa path
  }

  # Deny access to any file or directory starting with a dot
  location ~ /\. {
    deny all;
  }

  # Handle requests to the root URL (/) and redirect to the learning-
platform
  location = / {
    return 301 /learning-platform;
  }
}
```