Installation Manual for Napier Map Elites

William Hutcheson

October 1, 2018

Contents

1 Pre-requisites

This project is designed to be hosted using Docker[1] and docker-compose[2]. While the project is based on Flask and can be hosted using a standard web setup (e.g Nginx with uWSGI) this is not recommended.

Docker can be installed easily using the script at https://get.docker.com/ (although from a security point this is not recommended) or from your package manager. Windows users should be able to find an installer on the docker website. Docker-compose is a python package available through pip[3] or your package manager.

1.1 Downloading files

All the files required to host the website are available on github at https://github.com/hintofbasil/Napier-Map-Elites. It is unlikely that pull requests will be accepted into this repository and so you should forking would be a good idea.

1.2 Different environments

There are three environments built into the application. Development, Simple Production and Full Production. The core settings of each environment can be seen in flask/app/config.py

1.2.1 Development

This is the environment designed for developing new features. This does not require Docker and runs locally on a laptop only. It enables watch features to update the application code when changes are made.

Further details of how to set this up can be found in flask/app/README.md.

This will likely be updated in the future through Flask Docker Boilerplate (see **??**) to use Docker to help mitigate NPM issues (e.g https://blog.npmjs.org/post/175824896885/incident-report-npm-inc-operations-incident-of).

1.2.2 Simple Production

This is the environment designed for simple testing. It is designed to be very simple to launch, requiring no configuration, and to be secure enough to expose to the internet. This environment is hosted using Docker. It does not enable persistence of uploaded solution files. It does not require or allow secure connections (https).

[1]https://www.docker.com/
[2]https://docs.docker.com/compose/
[3]https://packaging.python.org/tutorials/installing-packages/

1.2.3 Full Production

This is the recommended environment for hosting the website on an internet facing web server. It enables solution persistence (although it is wiped at 00:00 daily). It also enables the use of secure connections (https). This does require some configuration, detailed below, and is only recommended for advanced users.

2 Development

The development environment is primarily used when updating the code. However with it being the only environment which works on the Windows operating system and also not requiring the use of Docker it may be of more use. This environment should not be made publicly accessible as it is not securely configured.

2.1 Installation

The application is built using Python and Node. As such it is assumed that Python[4] (v3.6+), Pip[5], Node[6] and npm[7] are installed and correctly configured. Download pages for these can be found in the footnotes.

The installation process is entirely performed on the command line. If you are unfamiliar with the command line there is a version of this program hosted at commute.napier.ac.uk which requires no set up.

The commands used here were tested on a Linux environment. While Windows alternatives are provided these are untested and may require some modification.

2.1.1 Command line setup

To install the application the following environment variables must be set. These must be set each time the program is run, or if using Linux can be saved in the .bashrc, .bash_profile or other configuration files.

```
--- Linux ---
export APP_SETTINGS=Development
export SECRET_KEY="change_me"
export FLASK_APP=$(pwd)/main.py

--- Windows ---
setx APP_SETTINGS Development
setx SECRET_KEY "change_me"
setx FLASK_APP %cd%/main.py
```

After this the python requirements must be installed. Do to this move into the flask/app directory.

```
--- Linux ---
cd flask/app/

--- Windows ---
cd flask\app\
```

[4]https://www.python.org/downloads/
[5]Is included with Python
[6]https://nodejs.org/en/download/
[7]Is included with Node

While in this directory run the following command. This command should be run inside a virtual environment[8]. A virtual environment can be created by running the following commands. If a virtual environment is not used administrator privileges are required.

```
--- Linux ---
python -m venv env
. env/bin/activate

--- Windows ---
python -m venv env
env\Scripts\activate.bat
```

2.1.2 Static file generation

To generate the static files we must first install the Node requirements. While in the flask/app folder run the following command.

```
npm i
```

When this command has finished the static files can be generated by running the following command.

```
npm run prod
```

2.1.3 Python installation

Once the virtual environment is set up run the following command.

```
pip install -r requirements.txt
```

Finally run the following command to launch the server. This binds to port 5000 and so the website can be accessed at 127.0.0.1:5000.

```
flask run
```

2.2 Updating

To update the code to the latest version run the following commands.

```
git pull origin master
npm run prod
flask run
```

[8]https://docs.python.org/3/tutorial/venv.html

3 Simple Production

Both production environments are build for Unix systems only. This means they will not work on Windows.

3.1 Systemd

Using systemd is the preferred method for launching the project as it allows for simple implementation of environment variables and allows for simple restarts.

3.1.1 Installation

First all the files in the systemd directory must be renamed to replace 'APP_NAME' with the name of your application.

```
export APP_NAME=YOUR_APP_NAME
cd systemd
rename APP_NAME $APP_NAME $(ls)
cd ..
```

The environment variables should then be set in `systemd/$APP_NAME.service.d/local.conf`. Any paths must be absolute paths.

These files must then be copied over to the systemd folder.

```
cp -r systemd/* /etc/systemd/system/
```

Finally systemd needs to be reloaded.

```
systemctl daemon-reload
```

3.1.2 Launching

To install the simple production environment the following commands should be run in the root folder of the repository. These commands require root or being a member of the 'docker' group.

```
systemctl start $APP_NAME.service
```

3.1.3 Updating

To update the test environment run the following commands while in the repository. Some of these commands require root.

```
git pull origin master
docker-compose build
systemctl restart $APP_NAME.service
```

3.2 Manual

Manual installation should typically only be used if systemd is not installed on the system or the above method failed for some reason.

3.2.1 Launching

To install the simple production environment the following commands should be run in the root folder of the repository. These commands require root or being a member of the 'docker' group.

```
docker-compose build
docker-compose up -d
```

3.2.2 Updating

To update the test environment run the following commands while in the repository. Some of these commands require root.

```
git pull origin master
docker-compose build
docker-compose down
docker-compose up -d
```

4 Full Production

Both production environments are build for Unix systems only. This means they will not work on Windows.

4.1 Certificate Generation

Generate the SSL certificates. The generated certificates only last 90 days and must be renewed before that (see 4.5). Email reminders will be sent to the registered email address.

```
docker run --rm -it -p 80:80 -p 443:443 -v
↪  /etc/letsencrypt:/etc/letsencrypt certbot/certbot certonly
*** Follow prompts selecting 'spin up a tempory web server' when
↪  asked ***
cp /etc/letsencrypt/live/DOMAIN_NAME/* $SSL_LOCATION
```

4.2 Systemd

Using systemd is the preferred method for launching the project as it allows for simple implementation of environment variables and allows for simple restarts.

4.2.1 Installation

First all the files in the systemd directory must be renamed to replace 'APP_NAME' with the name of your application.

```
export APP_NAME=YOUR_APP_NAME
cd systemd
rename APP_NAME $APP_NAME $(ls)
cd ..
```

The environment variables should then be set in `systemd/$APP_NAME.production.service.d/local.conf`. Any paths must be absolute paths.

These files must then be copied over to the systemd folder.

```
cp -r systemd/* /etc/systemd/system/
```

Finally systemd needs to be reloaded.

```
systemctl daemon-reload
```

4.2.2 Launching

To install the simple production environment the following commands should be run in the root folder of the repository. These commands require root or being a member of the 'docker' group.

```
systemctl start $APP_NAME.production.service
```

4.2.3 Updating

To update the test environment run the following commands while in the repository. Some of these commands require root.

```
git pull origin master
docker-compose build
systemctl restart $APP_NAME.production.service
```

4.3 Renewing certificates

```
systemctl stop $APP_NAME.production.service
docker run --rm -it -p 80:80 -p 443:443 -v
 ↪  /etc/letsencrypt:/etc/letsencrypt certbot/certbot renew
cp /etc/letsencrypt/live/DOMAIN_NAME/* $SSL_LOCATION
systemctl start $APP_NAME.production.service
```

4.4 Manual

Manual installation should typically only be used if systemd is not installed on the system or the above method failed for some reason.

4.4.1 Installation

Set the following environment variables. You may want to put these into your .bashrc file so you don't have to set them each time. Both paths should be absolute paths.

```
        export SSL_LOCATION="Location to host ssl certificates."
        export SOLUTIONS_FOLDER="Location to save solutions zip
         ↪  files to."
```

4.4.2 Launching

```
        docker-compose -f docker-compose.production.yml build
        docker-compose -f docker-compose.production.yml up -d
```

4.4.3 Updating the production environment

To update the production environment run the following commands while in the root directory of the repository. Some of these commands require root.

```
        git pull origin master
        docker-compose -f docker-compose.production.yml build
        docker-compose down
        docker-compose -f docker-compose.production.yml up -d
```

4.5 Renewing certificates

```
docker-compose down
docker run --rm -it -p 80:80 -p 443:443 -v
↪  /etc/letsencrypt:/etc/letsencrypt certbot/certbot
↪  renew
cp /etc/letsencrypt/live/DOMAIN_NAME/* $SSL_LOCATION
docker-compose -f docker-compose.production.yml up -d
```

5 Useful commands

5.1 Manually adding a solutions zip file

To manually add a solutions file to the server you simply need to copy it to `$SOLUTIONS_FOLDER`. An example is shown below where it is assumed `$SOLUTIONS_FOLDER` has been set to the same value as the remote server and `$SOLUTIONS_FILE`, `$CSV_FILE` and `$REMOTE_USERNAME` have been set.

```
mv "$SOLUTIONS_FILE" "$(md5sum $CSV_FILE | cut -d ' ' -f 1).zip"
scp "$SOLUTIONS_FILES"
↪  "$REMOTE_USERNAME@commute.napier.ac.uk:$SOLUTIONS_FOLDER"
```

5.2 Manually clearing locks

Lock files are created when solution files are uploaded. These locks are deleted on a successful upload but may not be deleted on a failed upload. They are automatically cleared 15 minutes after they are created but occasionally it is useful to clear the locks manually. The following command does this.

This command assumes the environment variables have been correctly set.

```
docker exec -it napiermapelites_flask_1
↪  /app/clear_locks.py
```

5.3 Clearing server data

While the server automatically deletes all uploaded solutions at midnight GMT it can be useful to manually clear the server. The following command achieves this.

This command assumes the environment variables have been correctly set.

```
docker exec -it napiermapelites_flask_1
↪  /app/clear_locks.py --ignore-locks --delete-solutions
```

5.4 Migrating data

To migrate data from one server to another install the application as normal. Then copy the $SOLUTIONS_FOLDER to the new server and set the $SOLUTIONS_FOLDER variable on the new server.