Deploying the Research Planner Web App to a TUoS Ubuntu Virtual Server

References

The section devoted to deploying a Django app to Microsoft Azure in the LinkedIn Learning course, Deploying Django Apps: Make your site go live, was used as a reference during this work. It can be found at https://www.linkedin.com/learning/deploying-django-apps-make-your-site-go-live/azure-account

The following web article contains a more in depth treatment of this subject: https://www.digitalocean.com/community/tutorials/how-to-set-up-django-with-postgres-nginx-and-gunicorn-on-ubuntu-22-04

Introduction

Prior to deployment, TUoS IT Services were asked to create a virtual Ubuntu server called qibsheffield with a user called sa_md1sshi. In the following description, these names are used where appropriate in commands. Substitute your own server and usernames in the commands as necessary.

IT Services should also be informed that a Postgres database will be used with this server and port 80 should be opened.

You will be installing Django within a virtual environment.

Once you have your database and application up and running, you will install and configure the Gunicorn application server. This will serve as an interface to our application, translating client requests from HTTP to Python calls that our application can process. You will then set up Nginx in front of Gunicorn. Nginx will handle requests for static content such as CSS file & send requests for dynamic content to Gunicorn.

Connecting to the TUoS virtual server

If you are working remotely, open a VPN connection first. Then open a Windows powershell terminal & enter the following command.

```
ssh sa_md1sshi@qibsheffield.shef.ac.uk
```

After, entering the password and performing MFA, you will be connected to the virtual server.

Enter the following command to reveal the current working directory as /home/sa_md1sshi were the web app will be installed.

```
$ pwd
```

Install updates and Python packages.

First you need to update the local apt package index and then download and install the packages.

Enter the command

```
$sudo apt update
```

then

```
$sudo apt upgrade
```

sudo is an abbreviation of "super user do" and is a Linux command that allows programs to be executed as a super user (aka root user) or another user.

The following command will install a tool to create virtual environments for your Python projects, the Python development files needed to build Gunicorn, the Postgres database system and the libraries needed to interact with it, and the Nginx web server.

```
$sudo apt install python3-venv python3-dev libpq-dev
postgresql postgresql-contrib nginx curl
```

Creating the PostgreSQL Database and User

Now create a database and database user for the Django web application.

Log into an interactive Postgres session by typing,

```
$sudo -u postgres psql
```

You will be given a PostgreSQL prompt where you can issue database commands.

If the above command does not work, contact IT Services to give you the necessary permissions to run a Postgres database on the virtual server.

First, create a database for your project:

```
postgres=# CREATE DATABASE plannerdb;
```

Note: Every Postgres statement must end with a semi-colon, so make sure that your command ends with one if you are experiencing issues.

Next, create a database user for our project. Make sure to select a secure password:

```
postgres=# CREATE USER planneruser WITH PASSWORD
'planner1234';
```

Now modify a few of the connection parameters for the user that you just created. This will speed up database operations so that the correct values do not have to be queried and set each time a connection is established.

You will set the default character encoding to `UTF-8`, which Django expects. You are also setting the default transaction isolation scheme to “read committed”, which blocks reads from uncommitted transactions. Lastly, you are setting the time zone. By default, Django projects will be set to use `UTC`.

Enter the following commands one by one, press enter after each command.

```
postgres=# ALTER ROLE planneruser SET client_encoding TO
'utf8';
```

```
postgres=# ALTER ROLE planneruser SET
default_transaction_isolation TO 'read committed';
```

```
postgres=# ALTER ROLE planneruser SET timezone TO 'UTC';
```

Now, you can give the new user access to administer the new database:

```
postgres=# GRANT ALL PRIVILEGES ON DATABASE plannerdb TO
planneruser;
```

When you are finished, exit out of the PostgreSQL prompt by typing:

```
postgres=# \q
```

Postgres is now set up so that Django can connect to and manage its database information.

Preparing Django web application for production.

To make the transition from a development environment to a production environment the following changes must be made to the files \research_planner\settings.py & \research_planner\.env

\research_planner\settings.py

Set the DEBUG directive to False

```
DEBUG = False
```

Locate the `ALLOWED_HOSTS` directive. This defines a list of the server’s addresses or domain names may be used to connect to the Django instance. The ALLOWED_HOSTS directive must include "qibsheffield.shef.ac.uk",

```
ALLOWED_HOSTS = ["localhost", "qibsheffield.shef.ac.uk"]
```

\research_planner\.env

Apart from the

```
SECRET_KEY=0e417e5f-d7c7-4a20-a11f-43833a868776
```

keyword-value pair which must not be modified, the .env file contains information required for the research planner web application to send emails to its users. Therefore, it is first necessary to create a GMail account that uses the GMail SMTP server to send these emails.

How to create a GMail account to send emails from the Gmail SMTP server.

1. Create a Gmail account, called for the purposes of these instructions, `planner.research.2022@gmail.com`
2. Go into the Account (https://myaccount.google.com/) of this account.
3. Under **Security**, go to **Signing in to Google** and turn on **2-Step Verification**
4. Complete all the steps to enable **2-Step Verification**
5. The **App Passwords** link should now be visible below **2-Step Verification**
6. Click the **App Passwords** link and follow the steps to create a 16-digit app password.
7. Two pieces of information are required for 6. The first is an option akin to Windows device. For the second, select 'Other' and type 'Django app'.
8. In the .env file, EMAIL_HOST_PASSWORD holds the value of the 16 digit app password.

The .env file should contain the following keyword-value pairs. Note there should be no spaces either side of the '=' sign; i.e., name=value not name = value.

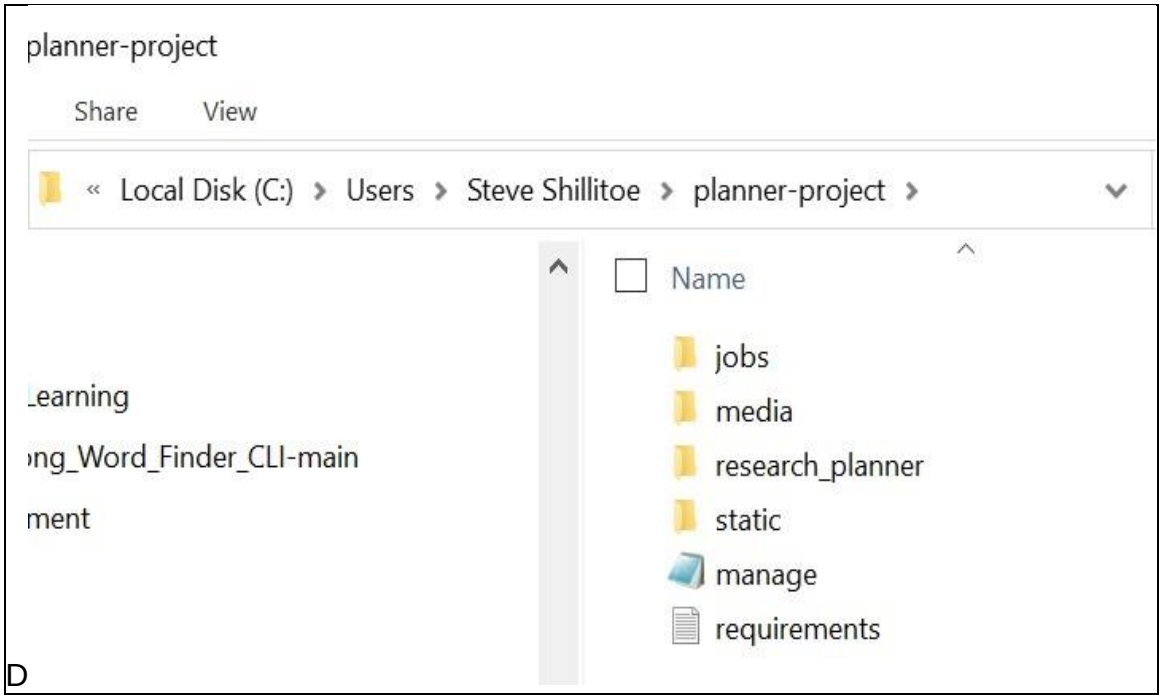
```
EMAIL_HOST_USER=planner.research.2022@gmail.com

EMAIL_HOST_PASSWORD=16 digit app password

SECRET_KEY=0e417e5f-d7c7-4a20-a11f-43833a868776

DOMAIN=qibsheffield.shef.ac.uk
```

Note: Change the value of the DOMAIN keyword if your server name differs from the above.

Transferring the Django web app to the TUoS virtual server.

Copy all the research planner folders and files into a folder called planner-project on your PC/laptop.

Delete all files and folders except for the 6 items in the figure below. Delete the `__pycache__` folder in each sub-folder.



Open a separate Windows PowerShell terminal and type the following command to transfer the planner-project folder and its contents to the virtual server. I suggest you first navigate using the cd command to the folder containing the planner-project folder to save typing its file path. You will be prompted for your password for the virtual server.

```
scp -r planner-project/
sa_md1sshi@qibsheffield.shef.ac.uk:/home/sa_md1sshi
```

If you need to update only views.py on the server, issue the following command,

```
scp -r planner-project/jobs/views.py
sa_md1sshi@qibsheffield.shef.ac.uk:/home/sa_md1sshi/planner-
project/jobs
```

When the file transfer is complete, you can close this Windows PowerShell terminal.

In the first Windows PowerShell terminal connected to the virtual server, type the following command to grant permissions on transferred folders

```
$sudo chmod 755 -R /home
```

Setting up the Python virtual environment on the virtual server.

In the Windows PowerShell terminal connected to the virtual server, type the following command to install the package to create a Python virtual environment.

```
$sudo apt install virtualenv
```

Create a Python virtual environment called venv,

```
$virtualenv venv
```

Activate the virtual environment,

```
$source venv/bin/activate
```

Navigate to where requirements.txt is located,

```
$cd planner-project
```

Install packages using requirements.txt

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

It was found that the following packages had to be manually installed,

```
$pip install gunicorn
```

```
$pip install psycopg2-binary
```

```
$pip install xlwt
```

```
$pip install crispy_bootstrap4
```

```
$pip install openpyxl
```

```
$pip install django-allauth
```

Populate the Postgres database

While still in the Python virtual environment, type the following command to populate the Postgres database,

```
$python manage.py migrate
```

Create a database superuser (administrator) by typing the following command and when prompted enter a username and password.

```
$python manage.py createsuperuser
```

You can now quit the virtual environment by typing the following comman,

```
$deactivate
```

Setting up Gunicorn

Creating systemd Socket and Service Files for Gunicorn

You'll configure the Gunicorn application server to interface with the research planner web application. You will then set up Nginx to reverse proxy to Gunicorn. Nginx will then serve static content such as CSS files to a web browser viewing the research planner application and it will direct requests for dynamic content to Gunicorn, which will run the corresponding Python script(s). To accomplish this, you'll make systemd service and socket files.

The Gunicorn socket will be created at boot and will listen for connections. When a connection occurs, systemd will automatically start the Gunicorn process to handle the connection.

Start by creating and opening a systemd socket file for Gunicorn with `sudo` privileges using the nano text editor:

```
$ sudo nano /etc/systemd/system/gunicorn.socket
```

Copy the following into your gunicorn.socket file.

```
[Unit]

Description=gunicorn socket

[Socket]

ListenStream=/run/gunicorn.sock

[Install]

WantedBy=sockets.target
```

- The `[Unit]` section describes the socket.
- The `[Socket]` section defines the socket location.

- The `[Install]` section makes sure the socket is created at the right time.

Save and close the file when you are finished by pressing ctrl + X, followed by Y and enter.

Next, create and open a systemd service file for Gunicorn with `sudo` privileges using the nano text editor. The service filename should match the socket filename with the exception of the extension:

```
$ sudo nano /etc/systemd/system/gunicorn.service
```

Copy the following into your gunicorn.service file. Save and close the file when you are finished by pressing ctrl + X, followed by Y and enter.

```
[Unit]

Description=gunicorn daemon

Requires=gunicorn.socket

After=network.target

[Service]

User= sa_md1sshi

Group=www-data

WorkingDirectory=/home/sa_md1sshi/planner-project

ExecStart=/home/sa_md1sshi/venv/bin/gunicorn \

          --access-logfile - \

          --workers 3 \

          --bind unix:/run/gunicorn.sock \

          research_planner.wsgi:application

[Install]

WantedBy=multi-user.target
```

- The `[Unit]` section specifies metadata and dependencies. It includes a description of the service and tells the init system to only start this after the

networking target has been reached. Because your service relies on the socket from the socket file, you need to include a `Requires` directive to indicate that relationship.

- The `[Service]` section specifies the user and group that the process runs under. You give your regular user account ownership of the process since it owns all of the relevant files. Group ownership is given to the `www-data` group so that Nginx can communicate easily with Gunicorn.

  Then the working directory and the command to use to start the service are specified. In this case, you have to specify the full path to the Gunicorn executable, which is installed within our virtual environment. You will then bind the process to the Unix socket you created within the `/run` directory so that the process can communicate with Nginx. You log all data to standard output so that the `journald` process can collect the Gunicorn logs. You can also specify any optional Gunicorn tweaks here. For example, you specified 3 worker processes in this case:

- `[Install]` section. This tells systemd what to link this service to if you enable it to start at boot. You want this service to start when the regular multi-user system is up and running.

You can now start and enable the Gunicorn socket. This will create the socket file at /run/gunicorn.sock now and at boot. When a connection is made to that socket, systemd will automatically start the gunicorn.service to handle it:

```
sudo systemctl start gunicorn.socket

sudo systemctl enable gunicorn.socket
```

After creating or if you change Gunicorn socket or service files, reload the daemon and restart the process by typing:

```
sudo systemctl daemon-reload

sudo systemctl restart gunicorn
```

Configure Nginx to Proxy Pass to Gunicorn

Now that Gunicorn is set up, you need to configure Nginx to pass traffic to it.

Start by creating and opening a new server block configuration file in Nginx's sites-available directory. A server block is a Nginx directive that defines settings for a specific domain, allowing you to run more than one websites on a single server. For each website, you can set the site document root (the directory which contains the website files), create a separate security policy, use different SSL certificates, and much more.

```
$ sudo nano /etc/nginx/sites-available/research_planner
```

Copy the following into your Nginx server block configuration file. Save and close the file when you are finished by pressing ctrl + X, followed by Y and enter.

```
server {

    listen 80;

    server_name qibsheffield.shef.ac.uk;

    location = /favicon.ico { access_log off; log_not_found off; }

    location /static/ {

        root /home/sa_md1sshi/planner-project;

    }



    location / {

        include proxy_params;

        proxy_pass http://unix:/run/gunicorn.sock;

    }

}
```

- This server should listen on port 80
- This server should respond to your server's domain name, qibsheffield.shef.ac.uk
- First location directive tells Nginx to ignore any problems with finding a favicon.
- Second location directive tells Nginx where to find the static assets that you collected in your ~/planner-project/static directory.
- The final location / {} block matches all other requests. So, inside this location, you'll include the standard proxy_params file included with the Nginx installation and instruct the Nginx server to pass all other requests directly to the Gunicorn socket.

Now, you can enable the file by linking it to the sites-enabled directory, which Nginx reads during start-up, using the following command,

```
$ sudo ln -s /etc/nginx/sites-available/research_planner /etc/nginx/sites-
enabled
```

Test your Nginx configuration for syntax errors by typing:

```
$ sudo nginx -t
```

If there are no errors, the output will look like this:

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

If no errors are reported, go ahead and restart Nginx by typing:

```
$ sudo systemctl restart nginx
```

Now you should be able to display the research planner web application in a web browser using the URL: http://qibsheffield.shef.ac.uk

SSL

server {

listen 443 ssl http2 default_server;

server_name qibsheffield.shef.ac.uk;

include snippets/certs.conf;

include snippets/ssl.conf;

add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

location = /favicon.ico { access_log off; log_not_found off; }

location /static/ {

root /home/sa_md1sshi/planner-project;

```
    }

    location / {
        include proxy_params;
        proxy_pass http://unix:/run/gunicorn.sock;
    }
}
```

Changing the research planner web application files or its server configuration.

Changing research planner Django files

If you make any changes to the files comprising the research planner Django application, you must restart the Gunicorn process to pick up the changes by typing:

```
$  sudo systemctl restart gunicorn
```

Changing the Gunicorn socket or service files

If you change Gunicorn socket or service files, reload the daemon and restart the process by typing:

```
$ sudo systemctl daemon-reload

$ sudo systemctl restart gunicorn
```

Changing the Nginx server block configuration

If you change the Nginx server block configuration, test the configuration and then restart Nginx by typing:

```
$ sudo nginx -t && sudo systemctl restart nginx
```

Troubleshooting the Nginx server

Start by looking at the Nginx error log. To view the Nginx error log, type the following two commands.

```
$ cd /var/log/nginx/
```

```
$ sudo nano error.log
```