Deploying the project to the internet has several steps, in this document, I will cover mainly the steps concerning adding the layers Gunicorn and Nginx and then how to turn HTTP into HTTPS. Also, how to serve the static files with Nginx and how to redirect from HTTP to HTTPS request.

Domain Name

For this example, I will be using the Google domain name from domains.google.com. The process to buy a domain is pretty straightforward. Once the domain name is bought, you can add several subdomains and then liked those to an IP address. The Type of domain used to link a domain name to IPV4 is Type A. The Type of domain used for IPV6 is AAAA.

A screenshot of a computer

Description automatically generated

Once saved, we have successfully linked our domain name to our IP address. Even when is mostly depreciated, we should add at least the “www” subdomain, and point it to our address.

Gunicorn

Django is a web framework, not a web server, and its maintainers want to make that distinction clear. In this section, you’ll replace Django’s <runserver> command with the <gunicorn> command. Gunicorn is first and foremost a Python WSGI app server, and a battle-tested one at that:

* It’s fast, optimized, and designed for production.
* It gives you more fine-grained control over the application server itself.
* It has more complete and configurable logging.
* It’s well-tested, specifically for its functionality as an application server.4

Installation command:

python -m pip install 'gunicorn==20.1.\*'

now that gunicorn is installed, we can just run the WSGI (Web Server Gateway Interface) app with gunicorn, and this will run our Django project. For us to do so, we have to pass several arguments to the gunicorn command. Instead of doing so, we will give our command a file where the configuration for the deployment is.

First, let’s create this file inside a conf directory, at the same height that our project’s father directory.

(django\_env) mike@localhost:~$ cd ~/folder-where-father-is

(django\_env) mike@localhost:~$ mkdir gunicorn\_config

(django\_env) mike@localhost:~$ nano gunicorn\_config.py

(django\_env) mike@localhost:~$

Now, inside this python file, we will simply create some variables with the general configuration settings for the deployment of our project.

GNU nano 5.6.1 conf/gunicorn\_config.py

"""gunicorn WSGI server configuration."""

from os import environ

bind = '127.0.0.1'

max\_requests = 1000

workers = 2

command = '/home/mike/ django\_env/bin/gunicorn'

pythonpath = '/home/mike/Cleaning\_Busines/Cleaning\_Busines/'

bind : the IP where you are running your Django server since we will proxy it

(redirect it), we want to run it in the localhost, and later redirect

to Nginx.

max\_request : the maximum number of requests the server will take

workers : the number of processes occurring at the same time. Ideally, this

the number will be an integer, and it will match the number of cores we

have in our server

command : the location of the binary file that will run gunicorn

pythonpath : the path from where our dot notation will start to look for our wsgi.py

file in our project

Run command:

gunicorn -c conf/gunicorn\_config.py <pythonpath to wsgi>.wsgi

Once we have done that, the first step is done, our server is running through gunicorn locally, but this is not what we want, we want to show it to the world, and also, we want to use HTTPS instead of HTTP which means that we still have work to do, we still need Nginx.

Nginx

As it is right now, we cannot even serve out static files to our server or use HTTPS protocol which means that our project is far from being ready for the production stage. Nginx takes care of serving the static files, as well as creating a proxy connection to the gunicorn server on the localhost and mediating all of the requests. The first thing to do to use Nginx is to install it. We will just install it as normal, with any package manager we have at hand, in my case, I am using apt.

Installation command:

sudo apt install nginx

At this point, if we check which processes are using what ports this is what we see:

(django\_env) mike@localhost:~$ sudo lsof -nP -iTCP -sTCP:LISTEN

COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME

systemd-r 461 systemd-resolve 14u IPv4 19073 0t0 TCP 127.0.0.53:53 (LISTEN)

sshd 616 root 3u IPv4 19321 0t0 TCP \*:22 (LISTEN)

sshd 616 root 4u IPv6 19323 0t0 TCP \*:22 (LISTEN)

gunicorn 21937 root 5u IPv4 260265 0t0 TCP 127.0.0.1:8000 (LISTEN)

gunicorn 21938 root 5u IPv4 260265 0t0 TCP 127.0.0.1:8000 (LISTEN)

gunicorn 21939 root 5u IPv4 260265 0t0 TCP 127.0.0.1:8000 (LISTEN)

(django\_env) mike@localhost:~$

The lines in blue are our gunicorn process, which is running the Django server locally, on port 8000. We will run this command later again so we can check our internet traffic and understand what is Django, Nginx, and gunicorn doing.

Whenever Nginx is installed, some files are installed as well, of course. For us to understand and use the power of Nginx, we must take a look inside those files. The first file we will look into is located in /etc/nginx/ and is called nginx.conf . This file will have the configuration for the default Nginx server, and of course for the one that we will serve from the localhost to the internet.

Then we will have to create a file containing some new configurations for the server, this file will contain the same type of content as nginx.conf, in fact, this file is a superset containing the new file as we will see now

GNU nano 5.6.1 /etc/nginx/nginx.conf

user www-data;

worker\_processes auto;

pid /run/nginx.pid;

include /etc/nginx/modules-enabled/\*.conf;

events {

worker\_connections 768;

# multi\_accept on;

}

http {

##

# Basic Settings

##

sendfile on;

tcp\_nopush on;

types\_hash\_max\_size 2048;

# server\_tokens off;

# server\_names\_hash\_bucket\_size 64;

# server\_name\_in\_redirect off;

include /etc/nginx/mime.types;

default\_type application/octet-stream;

##

# SSL Settings

##

ssl\_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping SSLv3, ref: POODLE

ssl\_prefer\_server\_ciphers on;

##

# Logging Settings

##

access\_log /var/log/nginx/access.log;

error\_log /var/log/nginx/error.log;

##

# Gzip Settings

##

GNU nano 5.6.1 continues down below...

GNU nano 5.6.1 /etc/nginx/nginx.conf(cont)

gzip on;

# gzip\_vary on;

# gzip\_proxied any;

# gzip\_comp\_level 6;

# gzip\_buffers 16 8k;

# gzip\_http\_version 1.1;

# gzip\_types text/plain text/css application/json application/javascript text/>

##

# Virtual Host Configs

##

include /etc/nginx/conf.d/\*.conf;

include /etc/nginx/sites-enabled/\*;

}

#mail {

# # See sample authentication script at:

# # http://wiki.nginx.org/ImapAuthenticateWithApachePhpScript

#

# # auth\_http localhost/auth.php;

# # pop3\_capabilities "TOP" "USER";

# # imap\_capabilities "IMAP4rev1" "UIDPLUS";

#

# server {

# listen localhost:110;

# protocol pop3;

# proxy on;

# }

#

# server {

# listen localhost:143;

# protocol imap;

# proxy on;

# }

#}

The lines above are the ones that make nginx.conf a superset, these lines include in this document, any other document that exists in the directories mentioned before. Now let’s create the file for our project.

This file must be inside /etc/nginx/sites-enabled/.

GNU nano 5.6.1 /etc/nginx/sites-enabled/AppsRepo

1.server {

2.

3. listen 443 ssl;

4. proxy\_set\_header Host $host;

5. proxy\_set\_header X-Real-IP $remote\_addr;

6. proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

7.

8. server\_name mikeapps.info www.mikeapps.info;

9. ssl\_certificate /etc/letsencrypt/live/mikeapps.info/fullchain.pem;

10. ssl\_certificate\_key /etc/letsencrypt/live/mikeapps.info/privkey.pem;

11.

12. location /static/ {

13. root /home/mike/AppsRepo/AppsRepo/;

14. }

15.

16. location / {

17. proxy\_pass http://127.0.0.1:8000/;

18. }

19.}

20.server {

21. listen 80;

22. server\_name mikeapps.info www.mikeapps.info;

23. return 301 https://$server\_name$request\_uri;

24.}

Here we have something called directives, directives are commonly used in configuration files, and they have features like inheritance, which means that a child directive from a parent context receives the parent settings. For example. the location in server one, the one who listens on 443 takes all the settings in the general scope of the server.

Now let’s break down this script to see what does each line means.

(1) Creates the first server context

(3) Server listening on port 443 with ssl (security) #See description for ssl

(4-6) Takes the original info from the client and delivers it to the gunicorn server,

instead of delivering the proxy’s info

(8) Define the server\_name variable with the allowed server names(the domain names)

(9-10) ssl certificate is necessary to send and receive a request with HTTPS. We

need to get a certificate and a key, we will cover this topic in depth below in its

own section

(12-14) first, context-directive defining where will the static files be located, at the end,

will be nginx who serves them

(16-18) second context-directive defining where will the root of the project be. This means

that we will use a proxy\_pass, we will take whatever is being served on the

localhost, and serve it on the IP we define later on

(19) first server context ends

(20) Creates the second server context, this server runs on port 80, and what it does is

to redirect whenever someone uses port 80, or HTTP to the first server, running on

port 443 with HTTPS and the ssl certificate, with this, the connection will always

be secure

(21) listens on port 80

(22) defines the server\_name

(23) redirects to the first server on port 443

(24) second context server ends

SSL Certificate and SSL Certificate key

On for our script to work, we need an ssl certificate and an ssl key. If you look closely at lines 9 and 10 in the last script, there are two files that which we haven’t talked about, those files are not there by default, in fact, at this moment in time they don’t exist at all because we don’t have them. The following section will cover how to add the cert and the key to the project so we can use port 443 with an ssl certificate and key.

Note: At this moment, you could perfectly use port 443, but you would have to delete lines 9 and 10, of course, or you would be looking for inexistent files, and also, the ssl in line 3.

To generate the Certificate and the key, we are going to use a package called certbot.

Installation command:

sudo apt install certbot python3-certbot-nginx

When you run certbot, the Nginx config file gets automatically updated to include the SSL certificate path. It does that by identifying the block that contains the server\_name we are passing into the command:

Cert and key creation command:

sudo certbot –-nginx -d mikeapps.com -d www.mikeapps.com

If the file doesn’t get the update, simply take out the ssl form line 3 and run the code again, this time you will have to manually add lines 9 and 10 with the location of the key and the certificate.

We have our nginx pointing to the localhost, where the server is, and the configuration is done, let’s run the Nginx server as a daemon, and then check again the status of the processes and their respective ports.

Run the nginx command:

sudo systemctl start nginx

Now let’s take a look at our processes:

(django\_env) mike@localhost:~$ sudo lsof -nP -iTCP -sTCP:LISTEN

COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME

systemd-r 461 systemd-resolve 14u IPv4 19073 0t0 TCP 127.0.0.53:53 (LISTEN)

sshd 616 root 3u IPv4 19321 0t0 TCP \*:22 (LISTEN)

sshd 616 root 4u IPv6 19323 0t0 TCP \*:22 (LISTEN)

nginx 23799 root 6u IPv4 289577 0t0 TCP \*:443 (LISTEN)

nginx 23799 root 7u IPv4 289578 0t0 TCP \*:80 (LISTEN)

nginx 23799 root 8u IPv6 289579 0t0 TCP \*:80 (LISTEN)

nginx 23800 www-data 6u IPv4 289577 0t0 TCP \*:443 (LISTEN)

nginx 23800 www-data 7u IPv4 289578 0t0 TCP \*:80 (LISTEN)

nginx 23800 www-data 8u IPv6 289579 0t0 TCP \*:80 (LISTEN)

nginx 23801 www-data 6u IPv4 289577 0t0 TCP \*:443 (LISTEN)

nginx 23801 www-data 7u IPv4 289578 0t0 TCP \*:80 (LISTEN)

nginx 23801 www-data 8u IPv6 289579 0t0 TCP \*:80 (LISTEN)

gunicorn 21937 root 5u IPv4 260265 0t0 TCP 127.0.0.1:8000 (LISTEN)

gunicorn 21938 root 5u IPv4 260265 0t0 TCP 127.0.0.1:8000 (LISTEN)

gunicorn 21939 root 5u IPv4 260265 0t0 TCP 127.0.0.1:8000 (LISTEN)

(django\_env) mike@localhost:~$

Now we can see that the nginx process is using ports 80 and 443, and also, the gunicorn process is using 127.0.0.1:8000, localhost on port 8000.

From the request to the python view function

From the Request to the python view function

Request

Python/html

WSGI

Gunicorn

Nginx

CSRF\_ TRUSTED\_ORIGINS:

The project just as it is it won’t let you post anything. There is then, another concept here to master, the cookies. The sites must be allowed, not just on the ALLOWED\_HOSTS list, but also in a CSRF\_TRUSTED\_ORIGINS list, right above this one on the settings.py from our project.

GNU nano 5.6.1 .../settings.py

(. . .)

CSRF\_TRUSTED\_ORIGINS = [

'https://mikeapps.info',

'https://www.mikeapps.info'

]

ALLOWED\_HOSTS = [ '127.0.0.1',

'45.79.15.247',

'www.mikeapps.info',

'mikeapps.info'

]

(. . .)