AWS, gunicorn, nginx setup

*Important note: anywhere you see text within [ ], ecd Des .g. [name of project], this is a placeholder and you will need to replace it with, for example the name of your project.

AWS

AWS - EC2 instances - Launch instance - (Check the free tier only option) - Ubuntu server, 18.04 (or whatever version they have live) - t2 micro - Key pair (must create and download)

SSH

Mac/Ubuntu: Chmod 400 [/path/to/ssh/key.pem]
Windows solution:
(https://superuser.com/questions/1296024/windows-ssh-permissions-for-private-key-are-too-open)

Ssh -i /path/to/key ubuntu@public_dns_name

========== Youre in ==========

Upload local django files to github:
Git init
Git remote add origin (github url)
Git add -A
Git commit -a
Git push -u origin master

Get files onto instance:

Git clone (url for repository)

Set up the machine with what it needs:

```
sudo apt-get update
sudo apt-get install python3-pip python3-dev libpq-dev postgresql
postgresql-contrib nginx
```

Also, to be sure

pip3 install django gunicorn psycopg2 sklearn
In Django settings file, add IPv4 Public IP (e.g. '3.8.140.139') to ALLOWED_HOSTS

In AWS sidebar scroll down to 'Network & Security'. Under this, select 'Security groups'. Click 'Create security group' (top left). Enter the following information:

Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ
Custom TCP Rule	TCP	8000	0.0.0.0/0

Go back to your instance (scroll to the top of the sidebar, select EC2 Dashboard, then click on Running instances. Right click on your instance, hover over 'Networking', and select 'Change security groups'. Select the one you've just created and then click 'Assign Security Groups'.

You now have your django files ready and the system ready to run. Cd into your django folder and run: python3 manage.py runserver `0.0.0.0:8000`.

Once the server is running, enter into your browser your IPv4 Public IP address (viewable on your instance, next to Public DNS, followed by :8000. For example, if your IPv4 Public IP address were 3.8.140.139, you would put in your browser http://3.8.140.139:8000

===== Hooray your site is now live for everyone to see! Anyone can now get onto your website =====

Keeping website running constantly, without runserver - Gunicorn:

```
cd into the top level folder of your django project (If you do an ls you
should see manage.py, among others).

gunicorn --bind 0.0.0.0:8000 [name of django project].wsgi
```

Test it's working by refreshing page in the browser - should still be working normally.

Create a new gunicorn configuration file. Enter the command:

sudo vim /etc/systemd/system/gunicorn.service

To make things easy on yourself, I would complete the following below in a text editor locally, and then paste it in in one go in vim. Otherwise you can type it manually into vim.

[Unit]
Description=gunicorn daemon
After=network.target

[Service]
User=ubuntu
Group=www-data
WorkingDirectory=/home/[path to top folder of django project]
ExecStart=/home/ubuntu/.local/bin/gunicorn --access-logfile - --workers 3 --bind unix:/home/[path to top folder of django project]/[django project name].sock [django project name].wsgi:application

[Install]
WantedBy=multi-user.target

```
sudo systemctl start gunicorn
sudo systemctl enable gunicorn
```

sudo systemctl status gunicorn

To make sure it's working, check to see if a .sock file has been created in the top folder of your django project.

From now on, if you make any changes to the project files, you'll need to restart gunicorn. This can be done with the following commands:

```
sudo systemctl restart gunicorn
```

If you change gunicorn configuration files run - `sudo systemctl daemon-reload`
Before you run the command above.
=== Gunicorn configuration complete ===

=== Nginx configuration ===

Create a new file - sudo vim /etc/nginx/sites-available/[name of django project]

In this file add the following text:
server {
    listen 80;

```
    server_name [server_domain_or_IP];

    location = /favicon.ico { access_log off; log_not_found off; }
    location /static/ {
        root /home/[path to top level folder of django];
    }
    location /media/ {
        root /home/[path to top level folder of django];
    }
    location / {
        include proxy_params;
        proxy_pass http://unix:/home/[path to top level folder of django]/[Project name].sock;
    }
}
```

** Important: The two blocks - static and media should only be included if you have either of these folders in your django project, e.g. if you're using CSS or JS scripts, or images, etc. Otherwise delete this blocks **

Enable by linking to the nginx sites-available folder
sudo ln -s /etc/nginx/sites-available/[name of django project] /etc/nginx/sites-enabled

Check for syntax errors:

sudo nginx -t

Restart nginx:

sudo systemctl restart nginx

Remove port 8000 permissions and allow nginx full access:

```
sudo ufw delete allow 8000
sudo ufw allow 'Nginx Full'
```

Go back into your AWS security group that you created before, and add another inbound rule:

Go to your browser, this time just entering the IP address, without :8000 at the end. You should now be able to see your site.

Troubleshooting:

If you're seeing the nginx welcome page instead, most likely there's something incorrect here - /etc/nginx/sites-available/[name of django project]. Make sure it's all written correctly, and if you don't have a static and/or media folder in your django project, make sure they're not included here.

If you're seeing bad gateway, 502 take a look at the logs, it will most likely tell you there what the problem is:

sudo tail -F /var/log/nginx/error.log

========== Nginx configuration complete ========

===== Changing IP to a url =======
For this section you will need to pay for a domain. If you do not want to spend money on this at this point then you can ignore this section.

First you must purchase a domain. The easiest to use is:

http://domains.google.com

Other options include:

https://ecohosting.co.uk/

https://www.godaddy.com

There are hundreds of providers out there who you could use to purchase a domain. Of course all of these will have different user interfaces, and so carrying out the task of directing the IP address to the domain will be slightly different in terms of what you need to click, but the foundations will be the same, so the instructions here will be quite generic. I recommend using a search engine to find the correct steps for your provider if it isn't intuitive.

After purchasing a domain, e.g. mydomain.com, you will need to navigate to what will be called DNS, DNS settings or DNS records.

Note: I will be using an example url of mysite.com.

Depending on your provider you will either have an existing type 'A' or will need to create a new one if one doesn't already exist. We are only interested in the A type.

After ensuring the type selected is A, you only need to add two pieces of information. One is your IPv4 address. The input field may say 'IPv4 address' or 'Point to' or something like this. The second thing is optional - TTL (Time to live). If you want to see the changes quickly, change this to

something like one minute. It's usually set to a default of an hour, which means that you won't be able to see your site at the url until an hour after you've saved this information. You can leave Host name blank.

After the duration of your TTL you can then access your site at your url, mysite.com!

You can add another A record, changing the Host name from blank to www. This will mean users can then access the site by typing either mysite.com or www.mysite.com

If you're still not seeing changes after your TTL duration, try doing a force refresh on your browser (usually ctrl or cmd + shift + r).

At this point you should see a "Welcome to Nginx" webpage. You need to configure two existing files before you can see your site:

Add your url - 'mysite.com' to ALLOWED_HOSTS in your settings.py file in your django project - ALLOWED_HOSTS = ['[IP address]', '[mysite.com]']
Add this as your server name in your /etc/nginx/sites-available/[name of django project] file. If you want to add in more than one url you can simply add them, separated by a space (note: not comma separated):

Server name '[IP address]' '[mysite.com]'

You will now need to restart gunicorn and nginx:

```
sudo systemctl restart gunicorn
sudo systemctl restart nginx
```

Adding CSS and Js to your site:

- To your settings.py, at the bottom, add the following lines:
    - Import os.path
    - STATICFILES_DIRS = (os.path.join('static'),) *Make sure to include that comma*
  - Create a CSS file in your static folder (you should create a static folder on the same directory level as your templates folder). Add whatever CSS you want in there.
  - Create a js file, e.g. main.js in your static folder.
-
  - To your html file, in the <head> section, add:
    - {% load static %}
    - <link rel="stylesheet" href="{% static 'style.css' %}">
    - At the bottom of your body, still inside the body tags, add:

- <script type="text/javascript" src="{% static 'main.js' %}"></script> (make sure your js file is called main.js, our otherwise change it here to your file name.

-

(If you're trying this locally, you'll have to remove the allowed hosts 'mysite.com', etc from settings.py. Remember to add them back in at the end).