**FDeploying django application to a production server part I**

[[](https://medium.com/@_christopher?source=post_page-----de78962e95ac----------------------)](https://medium.com/@_christopher?source=post_page-----de78962e95ac----------------------)

[Christopher Shoo](https://medium.com/@_christopher?source=post_page-----de78962e95ac----------------------)

[May 24, 2018](https://medium.com/@_christopher/deploying-my-django-app-to-a-real-server-part-i-de78962e95ac?source=post_page-----de78962e95ac----------------------) · 6 min read



setting up django with gunicorn, nginx and mysql on ubuntu server.

**If** you have been developing your web application with django on a development server and wondered, how i’m i going to put this into production on a real server? well you and i are going to do just that today so congratulations 🎊 you are going to become a pro.

On this post we are going to set up our awesome django app on an ubuntu server ( 17.10.1) with nginx, gunicorn and mysql, don’t worry about those terms, i will explain what they are.

some outline of what we are going to do.

* Installing necessary packages.
* Initial server setups.
* Preparing our django app for production.
* Setting up gunicorn.
* Setting up nginx.

Before we get started my terminal is working under /home/chris directory so change yours to that to follow-up with me. Ofcourse change chris to your username

Okay, i think we’ve had enough of the boring 💤 intro, lets do some real work.

**Installing necessary packages.**

To follow along you should have a fresh instance of your ubuntu server with a non root user created, if you don’t have access to a real server somewhere don’t worry you can install a copy of ubuntu 17.10 on your local machine or virtual machine and follow along, i’m using one myself, it will do just fine.

to get started we need to install some packages, run the following commands and answer the questions to install them

$ sudo apt-get update  
$ sudo apt-get install nginx mysql-server python3-pip python3-dev libmysqlclient-dev ufw virtualenv

i am using python3 on this tutorial but if you like python2 don’t worry you can always run the following commands, with just the 3 removed

$ sudo apt-get update  
$ sudo apt-get install nginx mysql-server python-pip python-dev libmysqlclient-dev ufw virtualenv

Wait! what did we just install?

**nginx**

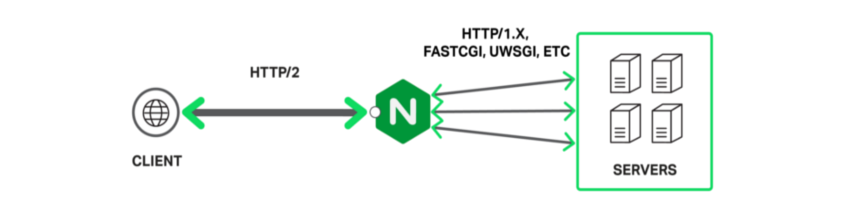


image downloaded from google

This is not a nginx tutorial but basically nginx is the web server that handles the requests from the browser and serves web pages in response to browser request. There are many other softwares like this such as apache and lighttpd, if you get familiar with things i suggest you go over and do a research on these because you might have a change of mind depending on your needs.

We’ve also installed mysql-server and libmysqlclient-dev, these packages will install mysql on your ubuntu and some mysql development files that are required to make things work. Again you could go with some other options such as postgresql, they all work fine, it all depends on your app requirements but avoid sqlite on a production server if you know your site is going to get a heavy traffic.

We’ve also installed python development files and python package manager (pip) which you must be familiar with as you are developing with django. If you haven’t used it, it’s used to install python packages. We’re going to use virtual environment so we installed virtualenv. Virtual environments is a way we use to isolate one project packages from the other in order to avoid conflicts, that’s because each app will be running on it’s own environment. So you can use the old version of this package on one app and new version of the same package on onether within the same server without conflicts. Cool isn’t it?

One last package is ufw, this is the firewall software, we are goin to use it to control at what ports should we be expecting the request to come from and block all other ports, In a few seconds we are going to do that.

**Initial server setup.**

first we are going to setup the firewall, for now we are going to block all ports and allow only port 8800 and 3306 for testing, this will later be changed when everything is setup ready. On your terminal type the following commands

$ sudo ufw default deny  
$ sudo ufw allow 8800  
$ sudo ufw allow 22  
$ sudo ufw enable

Note that i also allowed port 22, that’s in case you are connecting to the remote server with ssh while going along with the article. To see the status of the ufw type sudo ufw status. Now that the firewall is up and running let’s setup mysql.

run the following command to setup mysql on our server

$ mysql\_secure\_installation

follow the steps and make sure you disallow remote login unless you have plans for it, setup new root user password. After setting up mysql we need to create the database that our django app is going to use, i’m going to call my app awesome so the database name is going to be awesome, to do that run the following commands.

$ mysql -u root -p   
mysql> CREATE DATABASE awesome CHARACTER SET 'utf8';  
mysql> CREATE USER chris;  
mysql> GRANT ALL ON awesome.\* TO 'chris'@'localhost' IDENTIFIED BY 'secret';  
mysql> quit

we have created our database and a user chris and gave him all access to our database, this is good because chris can not access other databases on our system.

so now we have our firewall setup, mysql is set and we already have one database on it, let’s make our awesome django app and configure it to use the database we have just created. We said we are going to use virtual environment on our app so how do we do that? this way,

$ virtualenv venv  
$ source venv/bin/activate   
(env) $ pip install mysqlclient django==1.11

I called my virtual environment venv, you can name it anything and it will still work, to know that things went fine you will see the name of your virtual environment in brackets before the money sign on your terminal. note that i installed django too, you should also install it too but replace that 1.11 with the django version your app is running on. I asume you have your django app already so i’m not going to focus only on neccessary parts. For good practice put your django app under the same directory as your virtual environment folder, if you are following along with me put the folder containing your virtual environment under home in the <username> folder. mine is under /home/chris and it’s called webapp. My awesome app has the following structure ( the default one).

/home/chris/webapp/  
 |- venv/  
 |- AWESOME\_PROJECT/  
 |- awesome/  
 |- AWESOME/  
 |- wsgi.py  
 |- settings.py

AWESOME\_PROJECT is the project name and awesome is the app name. ofcourse it has all those models, url, forms files i just didn’t put them as they are not neccessary here.

on your settings file change the following settings.

DEBUG = False  
ALLOWED\_HOSTS = ['example.com', ]

DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.mysql',  
 'OPTIONS': {  
 'sql\_mode': 'traditional',  
 },  
 'NAME': 'awesome',  
 'USER': 'chris',  
 'PASSWORD': 'secret',  
 'HOST': 'localhost',  
 'PORT': '3306',   
 }  
 }  
   
# direcotries for static files

STATIC\_URL='/static/'  
STATIC\_ROOT=os.path.join(BASE\_DIR, 'static/')

MEDIA\_URL='/media/'  
MEDIA\_ROOT=os.path.join(BASE\_DIR, 'media/')

some settings are new and some are just adjusted, for example DEBUG=False in order to prevent django to expose errors to our users. We also added allowed hosts ALLOWED\_HOST=[’example.com’,] you need to put your domain name there if you have purchased one or your server ip address. We also changed the database dictionary field and added our own, this will shift our project from using sqlite to mysql database, if you observe you will see we have put our database info. Lastly we added our static files url where our css will live and media files url where the uploaded images and other media files will live. That’s all you need to setup on your django app, after that run the migration commands.

(env) $ python manage.py makemigrations  
(env) $ python manage.py migrate  
(env) $ python manage.py collectstatic

you can run the app on the development server to test if things are still working fine.

(env) $ python manage.py runserver

okay enough of the first part. The next last part we are going to setup nginx and gunicorn and that will be all. I’m going to the last cool part, things get soo much cooler there, if yo want to come with me click the below link, and feel free to respond or add few claps, that will motivate me to write more.

Welcome back, i’m glad you made it up to here, i promise there is going to be one hell of the happy ending on this one. So on the first part we had our app configured for deployment and we did some initial steps of the server configurations. If you don’t know what i’m talking about, don’t worry below is the link to the first part of this post.

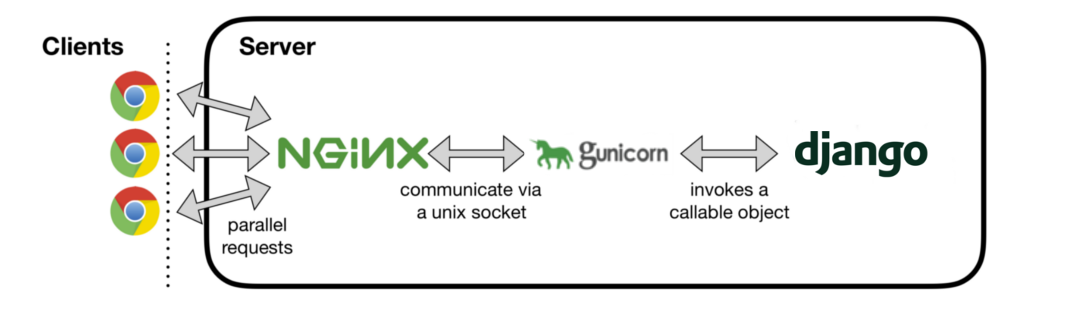
<https://medium.com/@_christopher/deploying-my-django-app-to-a-real-server-part-i-de78962e95ac>

On this part we are going to setup gunicorn first and lastly nginx so both of us can go to sleep.

I realised i say nginx and gunicorn a lot what are they anyway?

nginx is a web server that serves static files to the browser’s request, static files such as html, css and images but if the request requires some logic that your python code executes, that’s where gunicorn comes in. Gunicorn connects to your python code throung web server gateway interface (wsgi). The wsgi was designed to provide a way for the server to interact with your python code.

So how this configuration work is that, when a request from the browser comes, nginx checks if the request is for static or dynamic content. If the request is for static content nginx serves them to the browser but if the request is for dynamic content and requires some logic to be done, it passes the request to gunicorn and through wsgi gunicorn sends the request to your python code and python code will do the logic and provide the response. The response is sent back to nginx and hence the browser receives it.



We need nginx to serve static content and gunicorn to serve dynamic content, though that is not the only thing that nginx does, there are a lot more things that nginx does such as acting as a reverse proxy server, you can visit here [*what is nginx*](https://www.nginx.com/resources/glossary/nginx/)*?* to know more.

**setting up gunicorn**

while still in the virtual environment install gunicorn with the following command

(env) $ pip install gunicorn

if the development server is still running kill it change directory to your project folder and run the following command to run your site with gunicorn

(env) $ gunicorn --bind 0.0.0.0:8800 AWESOME.wsgi:application

remember i told you that gunicorn uses wsgi to interact with your python code, now that is what we just did. We pointed gunicorn to the wsgi located under AWESOME folder in AWESOME\_PROJECT directory (refer to my project directory structure on part one of this post). If you look you will see the file called wsgi.py and inside there will be a variable called application, that is where we pointed gunicorn to and now it’s interacting with our python code. If you visit admin page you will notice that there will be no css, that is because nginx is not run yet.

kill gunicorn and exit the virtual environment with the following command

(env) $ deactivate

to finish up gunicorn setup we need to create gunicorn service file so that it runs when the system starts.

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

paste the following configurations

[Unit]  
Description=gunicorn service  
After=network.target  
   
[Service]  
User=chris  
Group=www-data  
WorkingDirectory=/home/chris/webapp/AWESOME\_PROJECT/  
ExecStart=/home/chris/webapp/env/bin/gunicorn --access-logfile - --workers 3 --bind unix:/home/chris/webapp/AWESOME\_PROJECT/awesome.sock AWESOME.wsgi:application  
   
[Install]  
WantedBy=multi-user.target

ofcourse you need to change some variables to match your project’s location on your server and the user on your server.

on the User field change chris to your username, on the WorkingDirectory put the absolute path of your project’s directory, also on the ExecStrat change the directories to match yours. You should make sure you change the variables right otherwise you will find yourself in trouble. Type the following command to start gunicorn.

$ sudo systemctl enable gunicorn.service  
$ sudo systemctl start gunicorn.service  
$ sudo systemctl status gunicorn.service

If you run into trouble, in most cases you will, just type the following command to see what went wrong

$ journalctl -u gunicorn

if you changed something restart gunicorn with the following command

$ sudo systemctl daemon-reload  
$ sudo systemctl restart gunicorn

if you are intrested to know more about what’s on the config file we just created, visit this documentation [*gunicorn docs*](http://docs.gunicorn.org/en/latest/configure.html)*.* I encourage you to read that, it’s a better way to know what you’re doing.

**configuring nginx**

To configure nginx we need to create a config file that will tell nginx how to respond to a client’s request. The file will live under Nginx’s sites-available folder.

$ sudo nano /etc/nginx/sites-available/awesome

you can change awesome to other name, paste the following configurations

server {  
 listen 80;   
 server\_name 127.0.0.1;  
 location = /favicon.ico {access\_log off;log\_not\_found off;}   
   
 location /static/ {  
 root /home/chris/webapp/AWESOME\_PROJECT/AWESOME;   
 } location /media/ {  
 root /home/chris/webapp/AWESOME\_PROJECT/AWESOME;  
 }  
   
 location / {  
 include proxy\_params;  
 proxy\_pass [http://unix:/home/](http://unix:/home/user/Bookmark/Bookmark)chris/webapp/AWESOME\_PROJECT/AWESOME.sock;  
 }  
 }

change the directories to match yours, the first line tells nginx to listen on port 80 and the second provides it with the servers ip address to respond to, put the ip address of your server or the dns address, each location variable tells nginx where to find files for different requests, the first tells nginx where to find favicon.ico and ignore erros if it could not find it, the second and third tells nginx where to find static and media files respectfully, the last tells nginx to match all other requests. You can visit this link to know more about those settings, [*nginx setup*](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_pass)*.* If you’ve setup everything correctly run the following command

$ sudo ln -s /etc/nginx/sites-available/awesome /etc/nginx/sites-enabled

that will add a link of our new configuration file and that way nginx will recognise it. Run the following command to test your configurations

$ sudo nginx -t

if everything worked fine restart nginx with the following command

$ sudo systemctl restart nginx

Before it’s over we need to change our firewall rules and add port 80 because it is currently blocked. Type the following commands to do that

$ sudo ufw delete allow 8800  
$ sudo ufw allow 'Nginx Full'

if everything worked untill now, you should be good to go. You can now go to your server ip or domain name to see your application.

Thank you for taking your time reading my posts. I appreciate that, feel free to add a response or a few claps, it inspires me to write more.