TONY TEACHES TECH

MENU

How to Set Up Django on Nginx with uWSGI

November 5, 2020 by Tony Florida

Hey there! *Some links on this page may be affiliate links which means that, if you choose to make a purchase I may earn a small commission at no extra cost to you. I greatly appreciate your support!*

Privacy - Terms

In this tutorial, you will learn how to configure an Nginx server on Ubuntu with the Django web framework and WSGI as the web server gateway interface. The server configuration will be production-ready and set up for public release.

Throughout the tutorial, concepts will be introduced and demonstrated so you can gain an understanding of how each piece works.

Before you begin:

- *I will be using the domain name micro.domains throughout this tutorial.*
- *It is recommended that you configure your domain name's DNS A records to point to the IP address of your server. Here's a tutorial on how to do that. Don't have a server? I'll be using Linode to deploy a server running Ubuntu 20.04.*
- It is recommended that you use a user other than root. **Here's a tutorial** about how to add users to Ubuntu.

1. Update Your Server

It's always a good idea to start by updating your server.

```
sudo apt-get update
sudo apt-get upgrade
```

Now we have the latest and greatest packages installed on the server.

2. Use a Virtual Environment for Python

The venv module allows you to create virtual Python environments. This allows you to isolate installed packages from the system.

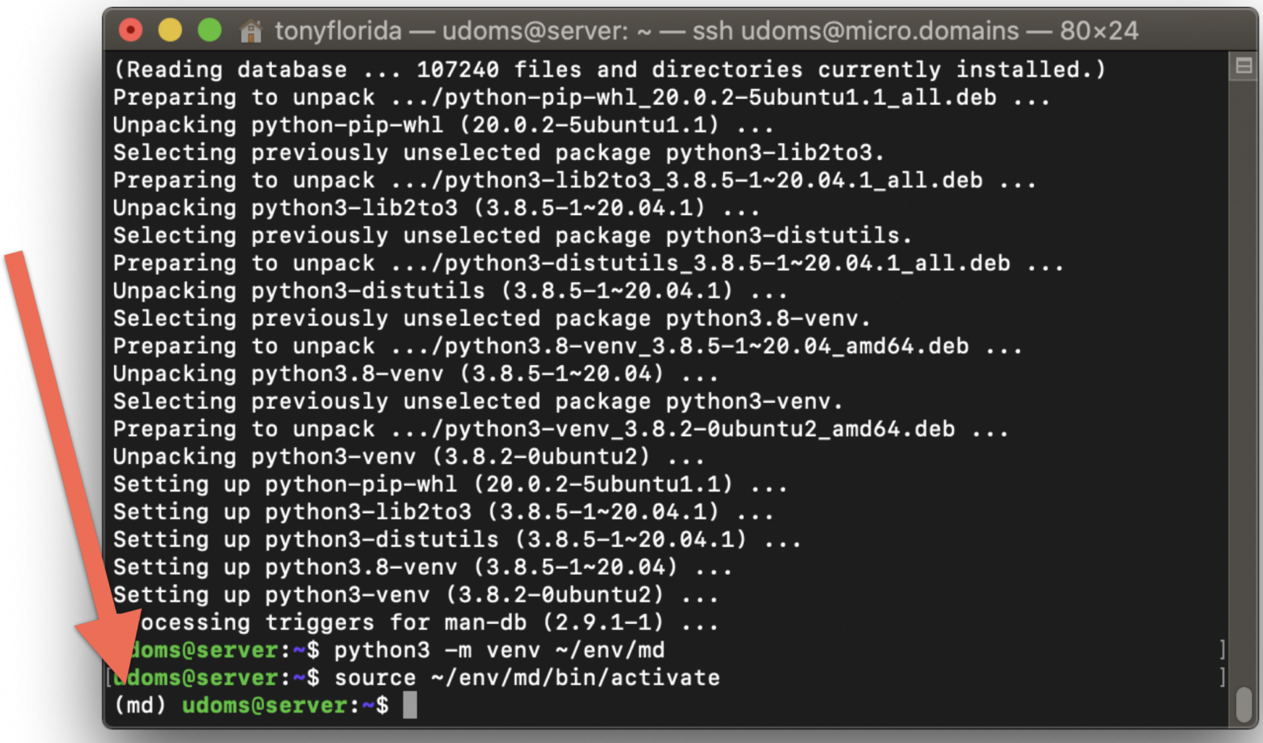
After installing venv, make a directory to house your virtual environments, and then create a virtual environment named `md` using venv. You can call your virtual environment whatever you prefer.

```
apt-get install python3-venv
mkdir /home/udoms/env/
python3 -m venv /home/udoms/env/md
```

Now activate your virtual environment.

```
source /home/udoms/env/md/bin/activate
```

You will see the name of your virtual environment in parenthesis as a prefix to your username and server name in your terminal window.



tonyflorida — udoms@server: ~ — ssh udoms@micro.domains — 80×24

```
(Reading database ... 107240 files and directories currently installed.)
Preparing to unpack .../python-pip-whl_20.0.2-5ubuntu1.1_all.deb ...
Unpacking python-pip-whl (20.0.2-5ubuntu1.1) ...
Selecting previously unselected package python3-lib2to3.
Preparing to unpack .../python3-lib2to3_3.8.5-1~20.04.1_all.deb ...
Unpacking python3-lib2to3 (3.8.5-1~20.04.1) ...
Selecting previously unselected package python3-distutils.
Preparing to unpack .../python3-distutils_3.8.5-1~20.04.1_all.deb ...
Unpacking python3-distutils (3.8.5-1~20.04.1) ...
Selecting previously unselected package python3.8-venv.
Preparing to unpack .../python3.8-venv_3.8.5-1~20.04_amd64.deb ...
Unpacking python3.8-venv (3.8.5-1~20.04) ...
Selecting previously unselected package python3-venv.
Preparing to unpack .../python3-venv_3.8.2-0ubuntu2_amd64.deb ...
Unpacking python3-venv (3.8.2-0ubuntu2) ...
Setting up python-pip-whl (20.0.2-5ubuntu1.1) ...
Setting up python3-lib2to3 (3.8.5-1~20.04.1) ...
Setting up python3-distutils (3.8.5-1~20.04.1) ...
Setting up python3.8-venv (3.8.5-1~20.04) ...
Setting up python3-venv (3.8.2-0ubuntu2) ...
 ocessing triggers for man-db (2.9.1-1) ...
 doms@server:~$ python3 -m venv ~/env/md
[ doms@server:~$ source ~/env/md/bin/activate
(md) udoms@server:~$
```

You can also verify that you are working from within your virtual environment by taking a look at where the Python binary is located.

```
which python
```

In this case, we are using Python version 3.8.5 which is located in the `/home/udoms/env/md/bin/python` directory.

3. Create a Django Project

Now that our Python environment is set up, we can install the Django web framework using the pip package installer.

```
pip install Django
```

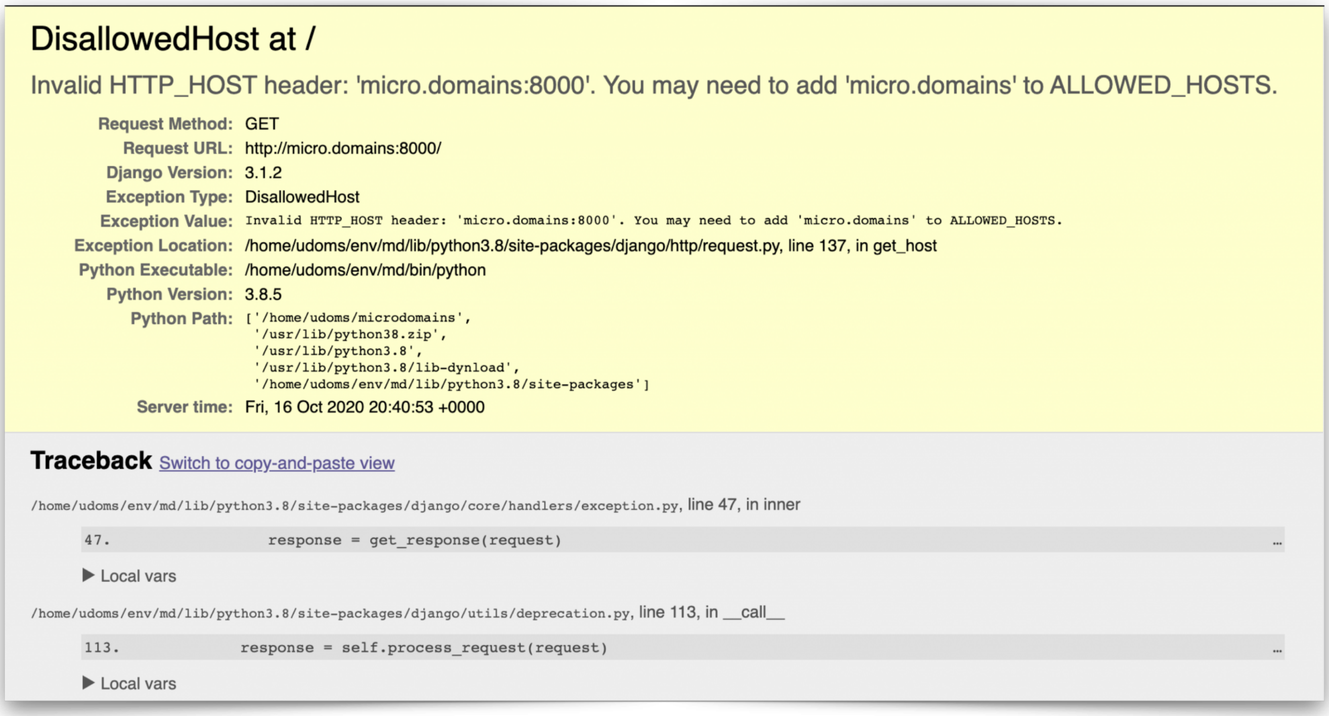
Next let's create a Django project with *django-admin.py* which should be on your path by default. Feel free to choose a project name that suites you.

```
django-admin.py startproject microdomains
cd microdomains
```

Test out your project by spinning up the built-in Django server with the following command:

```
python manage.py runserver 0.0.0.0:8000
```

In a browser, you should be able to visit `micro.domains:8000` and see the default Django landing page. If not, you might you see a Django error message like this:



```
DisallowedHost at /
Invalid HTTP_HOST header: 'micro.domains:8000'. You may need to add 'micro.domains' to ALLOWED_HOSTS.

   Request Method:  GET
      Request URL:  http://micro.domains:8000/
   Django Version:  3.1.2
   Exception Type:  DisallowedHost
  Exception Value:  Invalid HTTP_HOST header: 'micro.domains:8000'. You may need to add 'micro.domains' to ALLOWED_HOSTS.
Exception Location: /home/udoms/env/md/lib/python3.8/site-packages/django/http/request.py, line 137, in get_host
Python Executable:  /home/udoms/env/md/bin/python
   Python Version:  3.8.5
      Python Path:  ['/home/udoms/microdomains',
                     '/usr/lib/python38.zip',
                     '/usr/lib/python3.8',
                     '/usr/lib/python3.8/lib-dynload',
                     '/home/udoms/env/md/lib/python3.8/site-packages']
      Server time:  Fri, 16 Oct 2020 20:40:53 +0000

Traceback Switch to copy-and-paste view

/home/udoms/env/md/lib/python3.8/site-packages/django/core/handlers/exception.py, line 47, in inner
    47.                response = get_response(request)                               …
    ▶ Local vars
/home/udoms/env/md/lib/python3.8/site-packages/django/utils/deprecation.py, line 113, in __call__
    113.           response = self.process_request(request)                           …
    ▶ Local vars
```

Invalid HTTP_HOST header: 'micro.domains:8000'. You may need to add 'micro.domains' to ALLOWED_HOSTS.

If you see this error message, add your domain name to the `microdomains/settings.py` file.

```
...
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ['micro.domains', 'www.micro.domains']

# Application definition
```

```

INSTALLED_APPS = [
    'django.contrib.admin',
...
```

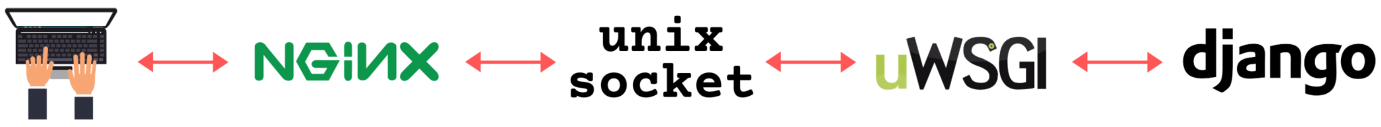
Try to visit `micro.domains:8000` again and this time you will see the default Django landing page.

4. Get Started With uWSGI

First we need to install the web server gateway interface (WSGI). In this case, we will be using uWSGI. You will also need the development packages for your version of Python to be installed.

```
sudo apt-get install python3.8-dev
sudo apt-get install gcc
pip install uwsgi
```

The ultimate goal in this tutorial is to send requests from the client to Nginx which will pass them to a socket that will hand them off to uWSGI before finally being given to Django.



That's a lot to configure right off the bat, so we will start with a much simpler test just to make sure all the individual puzzle pieces are in place.

Create a file called `test.py` with the following content:

```
def application(env, start_response):
    start_response('200 OK', [('Content-Type','text/html')])
    return [b"Hello World!"]
```

Let's test out uWSGI directly talking to Python with the following command:

```
uwsgi --http :8000 --wsgi-file test.py
```

In a browser, you should be able to visit `micro.domains:8000` and see the text "Hello World!"

If this works for you, we have demonstrated that uWSGI is able to pass requests to Python.

Now we can similarly serve the Django project with uWSGI with the following command:

```
uwsgi --http :8000 --module microdomains.wsgi
```

Note that this works because the path `microdomains/wsgi.py` exists. Test it out in a browser and you should see the default Django landing page again.

5. Configure the Nginx Web Server

Install Nginx with `apt-get` as follows:

```
sudo apt-get install nginx
```

Now with Nginx installed, you will be able to visit the default “Welcome to nginx!” page in your browser at `http://micro.domains` .

Let’s tell Nginx about our Django project by creating a configuration file at `/etc/nginx/sites-available/microdomains.conf` . Change the highlighted lines to suite your needs.

```
# the upstream component nginx needs to connect to
upstream django {
    server unix:///home/udoms/microdomains/microdomains.sock;
}

# configuration of the server
server {
    listen      80;
    server_name micro.domains www.micro.domains;
    charset     utf-8;

    # max upload size
    client_max_body_size 75M;

    # Django media and static files
    location /media  {
        alias /home/udoms/microdomains/media;
    }
19.     location /static {
20.         alias /home/udoms/microdomains/static;
21.     }
22.
23.     # Send all non-media requests to the Django server.
24.     location / {
25.         uwsgi_pass  django;
26.         include     /home/udoms/microdomains/uwsgi_params;
27.     }
28. }
```

We need to create the `/home/udoms/microdomains/uwsgi_params` file highlighted above on line 26 as well.

```
uwsgi_param  QUERY_STRING       $query_string;
uwsgi_param  REQUEST_METHOD     $request_method;
uwsgi_param  CONTENT_TYPE       $content_type;
uwsgi_param  CONTENT_LENGTH     $content_length;

uwsgi_param  REQUEST_URI        $request_uri;
uwsgi_param  PATH_INFO          $document_uri;
uwsgi_param  DOCUMENT_ROOT      $document_root;
uwsgi_param  SERVER_PROTOCOL    $server_protocol;
uwsgi_param  REQUEST_SCHEME     $scheme;
uwsgi_param  HTTPS              $https if_not_empty;

uwsgi_param  REMOTE_ADDR        $remote_addr;
uwsgi_param  REMOTE_PORT        $remote_port;
uwsgi_param  SERVER_PORT        $server_port;
uwsgi_param  SERVER_NAME        $server_name;
```

Next we can publish our changes by creating a symbolic link from sites-available to sites-enabled like so:

```
sudo ln -s /etc/nginx/sites-available/microdomains.conf /etc/nginx/sites-enabled/
```

We must edit the `microdomains/settings.py` file to explicitly tell Nginx where our static files reside.

First add `import os` at the very beginning:

```
"""
Django settings for microdomains project.
```

```

Generated by 'django-admin startproject' using Django 3.1.2.

For more information on this file, see
https://docs.djangoproject.com/en/3.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.1/ref/settings/
"""
import os
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

...
```

and `STATIC_ROOT = os.path.join(BASE_DIR, "static/")` at the very end:

```
...
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, "static/")
```

With these changes in place, we can now tell Django to put all static files in the static folder.

```
python manage.py collectstatic
```

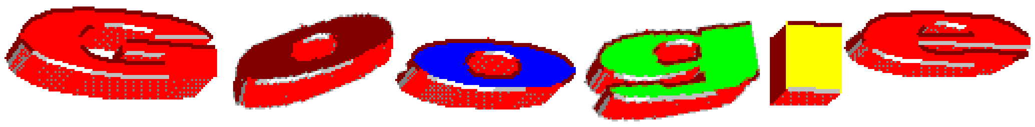
Restart the Nginx server to apply changes.

```
sudo /etc/init.d/nginx restart
```

At this point if you visit `http://micro.domains` in a browser, you will probably see an Nginx **502 Bad Gateway** error, but at this point we just want to test if media files are being properly served. So to test this out, let's put an image in our media directory.

```
mkdir media
wget https://upload.wikimedia.org/wikipedia/commons/b/b9/First-google-logo.gif -O media/media.gif
```

Finally visit `http://micro.domains/media/media.gif` in a browser and you should see Google's first logo from back in 1998.



6. Get Nginx, uWSGI, and Django to Work Together

Let's take this one step further and have Nginx, uWSGI, and Django work together with the help of the UNIX socket.

```
uwsgi --socket microdomains.sock --module microdomains.wsgi --chmod-socket=666
```

You can actually try the above command without the `--chmod-socket=666` argument and/or with a `--chmod-socket=664` argument instead. If either of those work for you, just keep that in mind going forward.

Once again, visit `http://micro.domains` in a browser, and this time you should see the default Django landing page!

7. Configure uWSGI for Production

Rather than passing arguments to uWSGI like we did above, we can put these options in a configuration file at the root of you Django project called `microdomains_uwsgi.ini`.

```
[uwsgi]

# full path to Django project's root directory
chdir                = /home/udoms/microdomains/
# Django's wsgi file
module               = microdomains.wsgi
# full path to python virtual env
home                 = /home/udoms/env/md

# enable uwsgi master process
master              = true
# maximum number of worker processes
processes           = 10
# the socket (use the full path to be safe
```

Privacy - Terms

```
socket          = /home/udoms/microdomains/microdomains.sock
# socket permissions
chmod-socket    = 666
# clear environment on exit
vacuum          = true
# daemonize uwsgi and write messages into given log
daemonize       = /home/udoms/uwsgi-emperor.log
```

You can then proceed to start up uwsgi and specify the ini file:

```
uwsgi --ini microdomains_uwsgi.ini
```

Visit `http://micro.domains` in a browser and you will see the default Django landing page if everything works correctly.

As one last configuration option for uWSGI, let's run uWSGI in emperor mode. This will monitor the uWSGI config file directory for changes and will spawn vassals (i.e. instances) for each one it finds.

```
cd /home/udoms/env/md/
mkdir vassals
sudo ln -s /home/udoms/microdomains/microdomains_uwsgi.ini
/home/udoms/env/md/vassals/
```

Now you can run uWSGI in emperor mode as a test.

```
uwsgi --emperor /home/udoms/env/md/vassals --uid www-data --gid www-data
```

Visit `http://micro.domains` in a browser and you will see the default Django landing page if everything works correctly.

Finally, we want to start up uWSGI when the system boots. Create a systemd service file at / `etc/systemd/system/emperor.uwsgi.service` with the following content:

```
[Unit]
Description=uwsgi emperor for micro domains website
After=network.target

[Service]
User=udoms
Restart=always
ExecStart=/home/udoms/env/md/bin/uwsgi --emperor
/home/udoms/env/md/vassals --uid www-data --gid www-data
```

Privacy - Terms

```

[Install]
WantedBy=multi-user.target
```

Enable the service to allow it to execute on system boot and start it so you can test it without a reboot.

```
systemctl enable emperor.uwsgi.service
systemctl start emperor.uwsgi.service
```

Visit `http://micro.domains` in a browser and you will see the default Django landing page if everything works correctly.

Check the status of the service and stop it as follows:

```
systemctl status emperor.uwsgi.service
systemctl stop emperor.uwsgi.service
```

For good measure, reboot your system to make sure that your website is accessible at startup.

Before going public with your website, it is highly recommended that you configure a few additional Django settings.

Please let me know if you have any questions about setting up Django with Nginx and uWSGI.

Server Administration

Meet Tony

With a strong software engineering background, Tony is determined to help as many people as possible start their online busines. Discover **why Tony quit his hedge fund job** to pursue this mission. You can **send Tony a message here**.

Leave a Comment

Name *

Email *

☐ Save my name, email, and website in this browser for the next time I comment.

Post Comment

Privacy - Terms