**A Simplified Guide to Hosting Multiple Django Apps on One Linux Server with Nginx and Gunicorn**

[[Mwangi Anthony](https://kenyattaanthony.medium.com/?source=post_page-----82ba6cc140d1--------------------------------)](https://kenyattaanthony.medium.com/?source=post_page-----82ba6cc140d1--------------------------------)

[[Stackademic](https://blog.stackademic.com/?source=post_page-----82ba6cc140d1--------------------------------)](https://blog.stackademic.com/?source=post_page-----82ba6cc140d1--------------------------------)

[Mwangi Anthony](https://kenyattaanthony.medium.com/?source=post_page-----82ba6cc140d1--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Fb54eead2c4ab&operation=register&redirect=https%3A%2F%2Fblog.stackademic.com%2Fa-simplified-guide-to-hosting-multiple-django-apps-on-one-linux-server-with-nginx-and-gunicorn-82ba6cc140d1&user=Mwangi+Anthony&userId=b54eead2c4ab&source=post_page-b54eead2c4ab----82ba6cc140d1---------------------post_header-----------)

Published in

[Stackademic](https://blog.stackademic.com/?source=post_page-----82ba6cc140d1--------------------------------)

·

6 min read

·

Nov 15, 2023

49

As an ambitious developer, you might wish to host more than one Django application on a solitary Linux server. Although achieving this with Nginx and Gunicorn may seem intricate, I’m here to simplify the procedure with detailed instructions.



**Setting the Stage**

/Imagine you have two Django sites, namely app\_1 and app\_2, both placed in a folder named PyApps. For each Django app, create a virtual environment to isolate dependencies. To follow along, create them inside apps. This ensures that each app operates independently, avoiding conflicts between packages.

python3 -m venv ./venv\_1

python3 -m venv ./venv\_2

Activate both environments one at a time as you install Python dependencies for both apps, not forgetting to install Gunicorn in their respective virtual environments.

pip install gunicorn

Go to your domain name registrar and create a records for both apps pointing to the same IP address of your Linux server.

**Understanding Nginx and Gunicorn**

Before diving into the specifics of serving multiple Django applications on a single Linux server, let’s briefly explore the roles of Nginx and Gunicorn in this setup.

* **Nginx**is a high-performance web server that excels at handling concurrent connections and acting as a reverse proxy, efficiently managing incoming web requests.
* **Gunicorn** (Green Unicorn): A WSGI HTTP server for Python web applications like Django. Gunicorn serves as an interface between your Django app and the outside web, managing multiple app instances and handling incoming web requests.

**Configuring Gunicorn for Each App**

With understanding Nginx and Gunicorn out of the way, the next step is configuring Gunicorn for each application. You will create specific configuration files for each app and manipulate these files as per the app’s requirements. The objective is to set up Gunicorn, so it is activated alongside other socket units when the system springs into action.

For the first app, open app\_1.socket file

# sudo nano /etc/systemd/system/app\_1.socket

Copy this code, paste it and save

[Unit]  
Description=gunicorn socket  
  
[Socket]  
ListenStream=/run/app\_1.sock  
  
[Install]  
WantedBy=sockets.target

The above configuration file sets up Gunicorn to use a specific stream socket and ensures that it is activated alongside other socket units when the system starts up. It contains three main sections:

1. **[Unit]:** This section describes the unit and includes a brief description of the *gunicorn socket*. This indicates that it is related to Gunicorn and is likely responsible for managing its socket.
2. **[Socket]:** This section specifies the type of socket that Gunicorn should use. In this case, it is a stream socket located at */run/app\_1.sock*.
3. **[Install]:** This section indicates that this unit is wanted by *sockets.target*, meaning it should be activated when*sockets.target* is activated during the system’s startup.

Open app\_1.service file

# sudo nano /etc/systemd/system/app\_1.service

Copy this code, paste it and save

[Unit]  
Description=gunicorn daemon  
Requires=app\_1.socket  
After=network.target  
  
[Service]  
User=yourusername  
Group=www-data  
WorkingDirectory=/home/yourusername/pyapps/app\_1  
ExecStart=/home/yourusername/pyapps/app\_1/venv/bin/gunicorn \  
 --access-logfile - \  
 --workers 3 \  
 --bind unix:/run/app\_1.sock \  
 app\_1.wsgi:application  
  
[Install]  
WantedBy=multi-user.target

This configuration sets up a Gunicorn service for the app\_1 application to run as a daemon, defining its dependencies, execution details, and installation behavior within the systemd ecosystem.

1. **[Unit]:** This section includes a description of the unit as a “*gunicorn daemon*.” It specifies that this unit requires *app\_1.socket* and should start after the network target is reached.
2. **[Service]:** This section defines the behavior of the service. It specifies the user and group under which the service will run, the working directory, and the command to start the Gunicorn server using a virtual environment. Additionally, it sets the access log file, the number of worker processes to 3, and the socket to bind to for communication.
3. **[Install]:** This section specifies that this unit should be enabled and started when the system reaches the multi-user target. Ensure that you set the path right for WSGI, depending on where it’s located in your project and the way your folders are named.

Now for app\_2, let’s repeat the above steps, taking care of paths and different file names. We open the app\_2.socket file using the command:

# sudo nano /etc/systemd/system/app\_2.service

Copy this code, paste it and save

[Unit]  
Description=gunicorn socket  
  
[Socket]  
ListenStream=/run/app\_2.sock  
  
[Install]  
WantedBy=sockets.target

The code has been explained above; it’s the same; the only thing that changed is where the stream socket for app\_2 is located.

Open app\_2.service file

# sudo nano /etc/systemd/system/app\_2.service

Copy this code, paste it and save

[Unit]  
Description=gunicorn daemon  
Requires=app\_2.socket  
After=network.target  
  
[Service]  
User=yourusername  
Group=www-data  
WorkingDirectory=/home/yourusername/pyapps/app\_2  
ExecStart=/home/yourusername/pyapps/app\_2/venv/bin/gunicorn \  
 --access-logfile - \  
 --workers 3 \  
 --bind unix:/run/app\_1.sock \  
 app\_2.wsgi:application  
  
[Install]  
WantedBy=multi-user.target

This configuration sets up a Gunicorn service for the “app\_2” application, the same as for “app\_1”.

Now we are going to start and enable Gunicorn sockets that we set up above

# sudo systemctl start app\_1.socket  
# sudo systemctl enable app\_1.socket  
  
# sudo systemctl start app\_2.socket  
# sudo systemctl enable app\_2.socket

Check the Gunicorn status of the above sockets

# sudo systemctl status app\_1.socket  
# sudo systemctl status app\_2.socket

I hope all are healthy and running. Also, we will check the existence of sock files for the two apps.

# file /run/app\_1.sock  
# file /run/app\_2.sock

**Setting up NGINX for Each App**

After the Gunicorn configuration, you need to establish suitable Nginx server blocks for each app. Effectively, an Nginx server block is a configuration piece defining how the web server handles requests for a specific server name and associated URL paths. This step includes creating a Nginx configuration file, ensuring it’s enabled, and checking its syntax.

We will start by creating an Nginx server block for app\_1.

# sudo nano /etc/nginx/sites-available/app\_1

Let's copy this code and paste it into the file.

server {  
 listen 80;  
 server\_name app\_1.com;  
  
 location = /favicon.ico { access\_log off; log\_not\_found off; }  
 location /static/ {  
 root /home/yourusername/pyapps/app\_1;  
 }  
   
 location /media/ {  
 root /home/yourusername/pyapps/app\_1;   
 }  
  
 location / {  
 include proxy\_params;  
 proxy\_pass http://unix:/run/app\_1.sock;  
 }  
}

This configuration is used to define how the web server should handle requests for a specific server name (in this case, app\_1.com) and its associated URL paths. Let’s break it down:

1. listen 80;This defines that the server is listening on port 80, the standard HTTP port.
2. server\_name app\_1.com;This sets the domain name that this server block should respond to.
3. location = /favicon.ico { access\_log off; log\_not\_found off; }This block controls access and error logs regarding a web browser's request for favicon.ico. The logs for this location are turned off.
4. location /static/ { root /home/yourusername/pyapps/app\_1; }This block serves static files, such as CSS, JavaScript, or images, for the app\_1.com domain. For any request that matches the /static/ URL pattern, Nginx serves the files from the root directory.
5. location /media/ { root /home/yourusername/pyapps/app\_1; }Similar to the static block, this location block similarly defines the root directory for serving media files (e.g., user-uploaded images or videos) for the app\_1.com domain.
6. location / { include proxy\_params; proxy\_pass http://unix:/run/app\_1.sock; }This is the main application block. If a requested URL does not match any of the previously mentioned location blocks, it will be handled by this block. This location uses Nginx as a reverse proxy to pass requests to a Gunicorn server listening on a Unix socket at */run/app\_1.sock*. The "*proxy\_params*" file includes parameters for the proxy.

The last leg involves creating a symbolic link, which is a cabinet that points to another location.

sudo ln -s /etc/nginx/sites-available/app\_1 /etc/nginx/sites-enabled

By running this command, you are creating a symbolic link that points from the sites-available directory to the sites-enabled directory. This means that the configuration file located insites-available/app\_1 is now enabled and available for use by Nginx.

In other words, this command enables the Nginx configuration file for the app\_1 application by making it available for use by Nginx.

Let’s check the syntax of our Nginx configuration file and display any errors or warnings, if there are any, by running the following command:

# sudo nginx -t

Now we restart Nginx

sudo systemctl restart nginx

Now take on the challenge of repeating the above steps for configuring Nginx for app\_2. When done, reload Gunicorn and Nginx for both apps.

# sudo systemctl daemon-reload  
# sudo systemctl restart app\_1.socket app\_1.service  
# sudo systemctl restart app\_2.socket app\_2.service  
# sudo systemctl restart nginx  
# sudo systemctl restart gunicorn

**Wrap-Up**

Having journeyed through the steps, you’re now empowered to host multiple Django applications on a single Linux server. Repeat the outlined steps for every new app you wish to host, ensuring you adjust paths and file names accordingly. Always remember to reload Gunicorn and Nginx to implement changes. It’s time to embrace a new level of ease and flexibility in managing your Django applications.

**Stackademic**

*Thank you for reading until the end. Before you go:*

* *Please consider****clapping****and****following****the writer! 👏*
* *Follow us on*[***Twitter(X)***](https://twitter.com/stackademichq)*,*[***LinkedIn***](https://www.linkedin.com/company/stackademic)*, and*[***YouTube***](https://www.youtube.com/c/stackademic)***.***
* *Visit*[***Stackademic.com***](http://stackademic.com/)*to find out more about how we are democratizing free programming education around the world.*