



TUTORIAL

How To Serve Flask Applications with Gunicorn and Nginx on Ubuntu 20.04

Ngix Ubuntu Python Python Frameworks Ubuntu 20.04

By [Kathleen Juell](#) and [Jamon Camisso](#)

Last Validated on May 20, 2020 • Originally Published on May 20, 2020 © 16k

🌐 English ▼

Not using Ubuntu 20.04?

Choose a different version or distribution.

Ubuntu 20.04 ▼

Introduction

In this guide, you will build a Python application using the Flask microframework on Ubuntu 20.04. The bulk of this article will be about how to set up the [Gunicorn application server](#) and how to launch the application and configure [Nginx](#) to act as a front-end reverse proxy.

Prerequisites

Before starting this guide, you should have:

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ✕

Enter your email address

Sign Up

[Namecheap](#) or get one for free on [Freenom](#). You can learn how to point domains to DigitalOcean by following the relevant [documentation on domains and DNS](#). Be sure to create the following DNS records:

- An A record with `your_domain` pointing to your server's public IP address.
 - An A record with `www.your_domain` pointing to your server's public IP address.
- Familiarity with the WSGI specification, which the Gunicorn server will use to communicate with your Flask application. [This discussion](#) covers WSGI in more detail.

Step 1 – Installing the Components from the Ubuntu Repositories

Our first step will be to install all of the pieces we need from the Ubuntu repositories. This includes `pip`, the Python package manager, which will manage our Python components. We will also get the Python development files necessary to build some of the Gunicorn components.

First, let's update the local package index and install the packages that will allow us to build our Python environment. These will include `python3-pip`, along with a few more packages and development tools necessary for a robust programming environment:

```
$ sudo apt update
$ sudo apt install python3-pip python3-dev build-essential libssl-dev libffi-dev python3-set
```

With these packages in place, let's move on to creating a virtual environment for our project.

Step 2 – Creating a Python Virtual Environment

Next, we'll set up a virtual environment in order to isolate our Flask application from the

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ✕

Sign Up

Next, let's make a parent directory for our Flask project. Move into the directory after you create it:

```
$ mkdir ~/myproject
$ cd ~/myproject
```

Create a virtual environment to store your Flask project's Python requirements by typing:

```
$ python3 -m venv myprojectenv
```

This will install a local copy of Python and `pip` into a directory called `myprojectenv` within your project directory.

Before installing applications within the virtual environment, you need to activate it. Do so by typing:

```
$ source myprojectenv/bin/activate
```

Your prompt will change to indicate that you are now operating within the virtual environment. It will look something like this: `(myprojectenv) user@host:~/myproject$`.

Step 3 – Setting Up a Flask Application

Now that you are in your virtual environment, you can install Flask and Gunicorn and get started on designing your application.

First, let's install `wheel` with the local instance of `pip` to ensure that our packages will install even if they are missing wheel archives:

```
$ pip install wheel
```

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ✕

Sign Up

activated, you should use the `pip` command (not `pip3`).

Next, let's install Flask and Gunicorn:

```
(myprojectenv) $ pip install gunicorn flask
```

Creating a Sample App

Now that you have Flask available, you can create a simple application. Flask is a microframework. It does not include many of the tools that more full-featured frameworks might, and exists mainly as a module that you can import into your projects to assist you in initializing a web application.

While your application might be more complex, we'll create our Flask app in a single file, called `myproject.py`:

```
(myprojectenv) $ nano ~/myproject/myproject.py
```

The application code will live in this file. It will import Flask and instantiate a Flask object. You can use this to define the functions that should be run when a specific route is requested:

```
~/myproject/myproject.py
```

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "<h1 style='color:blue'>Hello There!</h1>"

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ✕

Sign Up

```
(myprojectenv) $ sudo ufw allow 5000
```

Now you can test your Flask app by typing:

```
(myprojectenv) $ python myproject.py
```

You will see output like the following, including a helpful warning reminding you not to use this server setup in production:

Output

```
* Serving Flask app "myproject" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Visit your server's IP address followed by `:5000` in your web browser:

```
http://your_server_ip:5000
```

You should see something like this:



Hello There!

When you are finished, hit `CTRL-C` in your terminal window to stop the Flask development server.

Creating the WSGI Entry Point

Next, let's create a file that will serve as the entry point for our application. This will tell

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ✕

Sign Up

In this file, let's import the Flask instance from our application and then run it:

```
~/myproject/wsgi.py
```

```
from myproject import app

if __name__ == "__main__":
    app.run()
```

Save and close the file when you are finished.

Step 4 – Configuring Gunicorn

Your application is now written with an entry point established. We can now move on to configuring Gunicorn.

Before moving on, we should check that Gunicorn can serve the application correctly.

We can do this by simply passing it the name of our entry point. This is constructed as the name of the module (minus the `.py` extension), plus the name of the callable within the application. In our case, this is `wsgi:app`.

We'll also specify the interface and port to bind to so that the application will be started on a publicly available interface:

```
(myprojectenv) $ cd ~/myproject
(myprojectenv) $ gunicorn --bind 0.0.0.0:5000 wsgi:app
```

You should see output like the following:

Output

```
[2020-05-20 14:13:00 +0000] [16410] [INFO] Starting gunicorn 20.0.4
```

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ✕

Sign Up

again:

```
http://your_server_ip:5000
```

You should see your application's output:

Hello There!

When you have confirmed that it's functioning properly, press `CTRL-C` in your terminal window.

We're now done with our virtual environment, so we can deactivate it:

```
(myprojectenv) $ deactivate
```

Any Python commands will now use the system's Python environment again.

Next, let's create the `systemd` service unit file. Creating a `systemd` unit file will allow Ubuntu's `init` system to automatically start Gunicorn and serve the Flask application whenever the server boots.

Create a unit file ending in `.service` within the `/etc/systemd/system` directory to begin:

```
$ sudo nano /etc/systemd/system/myproject.service
```

Inside, we'll start with the `[Unit]` section, which is used to specify metadata and dependencies. Let's put a description of our service here and tell the `init` system to only start this after the networking target has been reached:

```
/etc/systemd/system/myproject.service
```

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ×

Sign Up

process since it owns all of the relevant files. Let's also give group ownership to the `www-data` group so that Nginx can communicate easily with the Gunicorn processes. Remember to replace the username here with your username:

```
/etc/systemd/system/myproject.service
```

```
[Unit]
Description=Gunicorn instance to serve myproject
After=network.target

[Service]
User=sammy
Group=www-data
```

Next, let's map out the working directory and set the `PATH` environmental variable so that the init system knows that the executables for the process are located within our virtual environment. Let's also specify the command to start the service. This command will do the following:

- Start 3 worker processes (though you should adjust this as necessary)
- Create and bind to a Unix socket file, `myproject.sock`, within our project directory. We'll set an `umask` value of `007` so that the socket file is created giving access to the owner and group, while restricting other access
- Specify the WSGI entry point file name, along with the Python callable within that file (`wsgi:app`)

Systemd requires that we give the full path to the Gunicorn executable, which is installed within our virtual environment.

Remember to replace the username and project paths with your own information:

```
/etc/systemd/system/myproject.service
```

```
[Unit]
Description=Gunicorn instance to serve myproject
```

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ✕

Sign Up

```
Environment="PATH=/home/sammy/myproject/myprojectenv/bin"  
ExecStart=/home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind unix:myproject.
```

Finally, let's add an `[Install]` section. This will tell `systemd` what to link this service to if we enable it to start at boot. We want this service to start when the regular multi-user system is up and running:

```
/etc/systemd/system/myproject.service
```

```
[Unit]  
Description=Gunicorn instance to serve myproject  
After=network.target  
  
[Service]  
User=sammy  
Group=www-data  
WorkingDirectory=/home/sammy/myproject  
Environment="PATH=/home/sammy/myproject/myprojectenv/bin"  
ExecStart=/home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind unix:myproject.  
  
[Install]  
WantedBy=multi-user.target
```

With that, our `systemd` service file is complete. Save and close it now.

We can now start the Gunicorn service we created and enable it so that it starts at boot:

```
$ sudo systemctl start myproject  
$ sudo systemctl enable myproject
```

Let's check the status:

```
$ sudo systemctl status myproject
```

You should see output like this:

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ✕

[Sign Up](#)

```

Active: active (running) since Wed 2020-05-20 14:15:18 UTC; 1s ago
Main PID: 46430 (gunicorn)
Tasks: 4 (limit: 2344)
Memory: 51.3M
CGroup: /system.slice/myproject.service
└─46430 /home/sammy/myproject/myprojectenv/bin/python3 /home/sammy/myproject/myproje
└─46449 /home/sammy/myproject/myprojectenv/bin/python3 /home/sammy/myproject/myproje
└─46450 /home/sammy/myproject/myprojectenv/bin/python3 /home/sammy/myproject/myproje
└─46451 /home/sammy/myproject/myprojectenv/bin/python3 /home/sammy/myproject/myproje

```

If you see any errors, be sure to resolve them before continuing with the tutorial.

Step 5 – Configuring Nginx to Proxy Requests

Our Gunicorn application server should now be up and running, waiting for requests on the socket file in the project directory. Let's now configure Nginx to pass web requests to that socket by making some small additions to its configuration file.

Begin by creating a new server block configuration file in Nginx's `sites-available` directory. Let's call this `myproject` to keep in line with the rest of the guide:

```
$ sudo nano /etc/nginx/sites-available/myproject
```

Open up a server block and tell Nginx to listen on the default port `80`. Let's also tell it to use this block for requests for our server's domain name:

```
/etc/nginx/sites-available/myproject
```

```
server {
    listen 80;
    server_name your_domain www.your_domain;
}
```

Next, let's add a location block that matches every request. Within this block, we'll

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ✕

Sign Up

```
server {  
    listen 80;  
    server_name your_domain www.your_domain;  
  
    location / {  
        include proxy_params;  
        proxy_pass http://unix:/home/sammy/myproject/myproject.sock;  
    }  
}
```

Save and close the file when you're finished.

To enable the Nginx server block configuration you've just created, link the file to the sites-enabled directory:

```
$ sudo ln -s /etc/nginx/sites-available/myproject /etc/nginx/sites-enabled
```

With the file in that directory, you can test for syntax errors:

```
$ sudo nginx -t
```

If this returns without indicating any issues, restart the Nginx process to read the new configuration:

```
$ sudo systemctl restart nginx
```

Finally, let's adjust the firewall again. We no longer need access through port 5000, so we can remove that rule. We can then allow full access to the Nginx server:

```
$ sudo ufw delete allow 5000  
$ sudo ufw allow 'Nginx Full'
```

You should now be able to navigate to your server's domain name in your web

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. 

Sign Up

Hello There!

If you encounter any errors, try checking the following:

- `sudo less /var/log/nginx/error.log`: checks the Nginx error logs.
- `sudo less /var/log/nginx/access.log`: checks the Nginx access logs.
- `sudo journalctl -u nginx`: checks the Nginx process logs.
- `sudo journalctl -u myproject`: checks your Flask app's Gunicorn logs.

Step 6 – Securing the Application

To ensure that traffic to your server remains secure, let's get an SSL certificate for your domain. There are multiple ways to do this, including getting a free certificate from [Let's Encrypt](#), [generating a self-signed certificate](#), or [buying one from another provider](#) and configuring Nginx to use it by following Steps 2 through 6 of [How to Create a Self-signed SSL Certificate for Nginx in Ubuntu 20.04](#). We will go with option one for the sake of expediency.

Install Certbot's Nginx package with `apt`:

```
$ sudo apt install python3-certbot-nginx
```

Certbot provides a variety of ways to obtain SSL certificates through plugins. The Nginx plugin will take care of reconfiguring Nginx and reloading the config whenever necessary. To use this plugin, type the following:

```
$ sudo certbot --nginx -d your_domain -d www.your_domain
```

This runs `certbot` with the `--nginx` plugin, using `-d` to specify the names we'd like the

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ✕

Sign Up

requesting a certificate for.

If that's successful, `certbot` will ask how you'd like to configure your HTTPS settings:

Output

Please choose whether or not to redirect HTTP traffic to HTTPS, removing HTTP access.

```
-----
1: No redirect - Make no further changes to the webserver configuration.
2: Redirect - Make all requests redirect to secure HTTPS access. Choose this for
new sites, or if you're confident your site works on HTTPS. You can undo this
change by editing your web server's configuration.
-----
```

Select the appropriate number [1-2] then [enter] (press 'c' to cancel):

Select your choice then hit `ENTER`. The configuration will be updated, and Nginx will reload to pick up the new settings. `certbot` will wrap up with a message telling you the process was successful and where your certificates are stored:

Output

IMPORTANT NOTES:

- Congratulations! Your certificate and chain have been saved at:
`/etc/letsencrypt/live/your_domain/fullchain.pem`
 Your key file has been saved at:
`/etc/letsencrypt/live/your_domain/privkey.pem`
 Your cert will expire on 2020-08-18. To obtain a new or tweaked version of this certificate in the future, simply run `certbot` again with the "certonly" option. To non-interactively renew **all** of your certificates, run "`certbot renew`"
- Your account credentials have been saved in your Certbot configuration directory at `/etc/letsencrypt`. You should make a secure backup of this folder now. This configuration directory will also contain certificates and private keys obtained by Certbot so making regular backups of this folder is ideal.
- If you like Certbot, please consider supporting our work by:

Donating to ISRG / Let's Encrypt: <https://letsencrypt.org/donate>

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ✕

Sign Up

```
$ sudo ufw delete allow 'Nginx HTTP'
```

To verify the configuration, navigate once again to your domain, using `https://`:

```
https://your_domain
```

You should see your application output once again, along with your browser's security indicator, which should indicate that the site is secured.

Conclusion

In this guide, you created and secured a simple Flask application within a Python virtual environment. You created a WSGI entry point so that any WSGI-capable application server can interface with it, and then configured the Gunicorn app server to provide this function. Afterwards, you created a systemd service file to automatically launch the application server on boot. You also created an Nginx server block that passes web client traffic to the application server, relaying external requests, and secured traffic to your server with Let's Encrypt.

Flask is a very simple, but extremely flexible framework meant to provide your applications with functionality without being too restrictive about structure and design. You can use the general stack described in this guide to serve the flask applications that you design.

Was this helpful?

Yes

No



[Report an issue](#)

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ×

Sign Up

About the authors



Kathleen Juell

Developer
@digitalocean/community



Jamon Camisso

has authored 34 tutorials.

Still looking for an answer?



Ask a question



Search for more help

RELATED

DigitalOcean App Platform

You bring your web app in a

- GitHub repo
- App Platform handles deployments and builds
- DNS, HTTPS, CDN, DDoS Mitigation, Vertical Scaling, Horizontal Scaling, and more.

[More Info](#)

What is nginx?

[Tutorial](#)

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ×

[Sign Up](#)