DigitalOcean

By: Justin Ellingwood

Subscribe

How To Serve Flask Applications with Gunicorn and Nginx on Ubuntu 14.04

29

Posted March 20, 2015 181.5k DEPLOYMENT PYTHON FRAMEWORKS NGINX PYTHON UBUNTU

Introduction

In this guide, we will be setting up a simple Python application using the Flask micro-framework on Ubuntu 14.04. The bulk of this article will be about how to set up the Gunicorn application server to launch the application and Nginx to act as a front end reverse proxy.

Prerequisites

Before starting on this guide, you should have a non-root user configured on your server. This user needs to have `sudo` privileges so that it can perform administrative functions. To learn how to set this up, follow our initial server setup guide.

To learn more about the WSGI specification that our application server will use to communicate with our Flask app, you can read the linked section of this guide. Understanding these concepts will make this guide easier to follow.

When you are ready to continue, read on.

Install the Components from the Ubuntu Repositories

Our first step will be to install all of the pieces that we need from the repositories. We will install `pip`, the Python package manager, in order to install and manage our Python components. We will also get the Python development files needed to build some of the Gunicorn components. We'll install Nginx now as well.

Update your local package index and then install the packages by typing:

```
sudo apt-get update
sudo apt-get install python-pip python-dev nginx
```

Create a Python Virtual Environment

Next, we'll set up a virtual environment in order to isolate our Flask application from the other Python files on the system.

Start by installing the `virtualenv` package using `pip`:

```
sudo pip install virtualenv
```

Now, we can make a parent directory for our Flask project. Move into the directory after you create it:

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

We can create a virtual environment to store our Flask project's Python requirements by typing:

```
virtualenv myprojectenv
```

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

Before we install applications within the virtual environment, we need to activate it. You can 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$`.

Set Up a Flask Application

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

Install Flask and Gunicorn

We can use the local instance of `pip` to install Flask and Gunicorn. Type the following

```
pip install gunicorn flask
```

Create a Sample App

Now that we have Flask available, we can create a simple application. Flask is a micro-framework. 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, which we will call `myproject.py`:

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

Within this file, we'll place our application code. Basically, we need to import flask and instantiate a Flask object. We can use this to define the functions that should be run when a specific route is requested. We'll call our Flask application in the code `application` to replicate the examples you'd find in the WSGI specification:

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

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

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

This basically defines what content to present when the root domain is accessed. Save and close the file when you're finished.

You can test your Flask app by typing:

```
python myproject.py
```

Visit your server's domain name or IP address followed by the port number specified in the terminal output (most likely `:5000`) in your web browser. You should see something like this:

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

Create the WSGI Entry Point

Next, we'll create a file that will serve as the entry point for our application. This will tell our Gunicorn server how to interact with the application.

We will call the file `wsgi.py`:

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

The file is incredibly simple, we can simply import the Flask instance from our application and then run it:

```
from myproject import application

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

Save and close the file when you are finished.

Testing Gunicorn's Ability to Serve the Project

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

We can do this by simply passing it the name of our entry point. We'll also specify the interface and port to bind to so that it will be started on a publicly available interface:

```
cd ~/myproject
gunicorn --bind 0.0.0.0:8000 wsgi
```

If you visit your server's domain name or IP address with `:8000` appended to the end in your web browser, you should see a page that looks like this:

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

```
deactivate
```

Any operations now will be done to the system's Python environment.

Create an Upstart Script

The next piece we need to take care of is the Upstart script. Creating an Upstart script will allow Ubuntu's init system to automatically start Gunicorn and serve our Flask application whenever the server boots.

Create a script file ending with `.conf` within the `/etc/init` directory to begin:

```
sudo nano /etc/init/myproject.conf
```

Inside, we'll start with a simple description of the script's purpose. Immediately afterwards, we'll define the conditions where this script will be started and stopped by the system. The normal system runtime numbers are 2, 3, 4, and 5, so we'll tell it to start our script when the system reaches one of those runlevels. We'll tell it to stop on any other runlevel (such as when the server is rebooting, shutting down, or in single-user mode):

```
description "Gunicorn application server running myproject"

start on runlevel [2345]
stop on runlevel [!2345]
```

We'll tell the init system that it should restart the process if it ever fails. Next, we need to define the user and group that Gunicorn should be run as. Our project files are all owned by our own user account, so we will set ourselves as the user to run. The Nginx server runs under the `www-data` group. We need Nginx to be able to read from and write to the socket file, so we'll give this group ownership over the process:

```
description "Gunicorn application server running myproject"

start on runlevel [2345]
stop on runlevel [!2345]

respawn
setuid user
setgid www-data
```

Next, we need to set up the process so that it can correctly find our files and process them. We've installed all of our Python components into a virtual environment, so we need to set an environmental variable with this as our path. We also need to change to our project directory. Afterwards, we can simply call the Gunicorn application with the options we'd like to use.

We will tell it to start 3 worker processes (adjust this as necessary). We will also tell it to create and bind to a Unix socket file within our project directory called `myproject.sock`. We'll set a umask value of `007` so that the socket file is created giving access to the owner and group, while restricting other access. Finally, we need to pass in the WSGI entry point file name:

```
description "Gunicorn application server running myproject"

start on runlevel [2345]
stop on runlevel [!2345]

respawn
setuid user
setgid www-data

env PATH=/home/user/myproject/myprojectenv/bin
chdir /home/user/myproject
exec gunicorn --workers 3 --bind unix:myproject.sock -m 007 wsgi
```

Save and close the file when you are finished.

You can start the process immediately by typing:

```
sudo start myproject
```

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. We need to 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. We'll simply call this `myproject` to keep in line with the rest of the guide:

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


tell it to

```
server {
    listen 80;
    server_name server_domain_or_IP;
}
```

The only other thing that we need to add is a location block that matches every request. Within this block, we'll include the `proxy_params` file that specifies some general proxying parameters that need to be set. We'll then pass the requests to the socket we defined using the `proxy_pass` directive:

```
server {
    listen 80;
    server_name server_domain_or_IP;

    location / {
        include proxy_params;
        proxy_pass http://unix:/home/user/myproject/myproject.sock;
    }
}
```

That's actually all we need to serve our application. Save and close the file when you're finished.

To enable the Nginx server block configuration we'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, we can test for syntax errors by typing:

```
sudo nginx -t
```

If this returns without indicating any issues, we can restart the Nginx process to read the our new config:

```
sudo service nginx restart
```

You should now be able to go to your server's domain name or IP address in your web browser and see your application:

Conclusion

In this guide, we've created a simple Flask application within a Python virtual environment. We create 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, we created an Upstart script to automatically launch the application server on boot. We created an Nginx server block that passes web client traffic to the application server, relaying external requests.

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.

By: Justin Ellingwood

Related Tutorials

How To Deploy a Jekyll Site Using Git Hooks on Ubuntu 16.04

How to Automatically Deploy Laravel Applications with Deployer on Ubuntu 16.04

How to Write a Slash Command with Flask and Python 3 on Ubuntu 16.04

How To Automate Elixir-Phoenix Deployment with Distillery and edeliver on Ubuntu 16.04

58 Comments

Leave a comment...

Log In to Comment

thataintworking *April 2, 2015*

0 Don't you need to activate the virtualenv in the upstart script before running gunicorn?

jellingwood MOD *April 2, 2015*

1 For this example, you don't need to do that since we are specifying the PATH where all of the Python executables we care about are located. If this doesn't suit your needs however, you can convert the `exec` line into a `script` block like this:

```
. . .

script
    cd /home/user/myproject
    source myprojectenv/bin/activate
    gunicorn --workers 3 --bind unix:myproject.sock -m 007 wsgi
end script
```

With a little testing and tweaking, that should correctly load the virtual environment for you if the PATH isn't enough to catch all of the things you need from your Virtual env. Hope that helps.

a chance to
ache. So this

is like the modern way of doing everything I did earlier. Thanks for that!

lequ *May 7, 2015*

1 When I try to run the "gunicorn --bind 0.0.0.0:8000 wsgi" I get a "Failed to find application : 'wsgi'". I'm in the same dir as wsgi.py file. Any ideas?

Thanks in advance!

ryliemn *May 9, 2015*

4 I just had to figure out this same problem. Had to run the following command:

gunicorn --bind 0.0.0.0:8000 wsgi:app

Hope this works for you.

jordantkrueger *September 29, 2015*

1 This works for me, but do you know why I had to add that specification to the command? It seems like the tutorial (implicitly anyway) worked fine without it, so I'm trying to see why it was necessary for my particular project/configuration.

hukevinxiaochen *May 21, 2016*

0 Same here. I think the tutorial has incorrect syntax. At the very least, it'd be nice to include the alternative syntax:

```
gunicorn --bind 0.0.0.0:8000 wsgi:app
```

in the instructions.

salah93 *February 3, 2018*

0 Its because gunicorn looks for the variable "application" by default, not "app"

neekburm *May 15, 2015*

0 I'm having trouble accessing the app after initially configuring it any running the server with python myproject.py. The server is running (It says "Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)")

able to
rom there,

but what could I be doing wrong?

marktur *March 17, 2016*

0 '127.0.0.1' is the address of 'localhost'. This means when you have the application runtime bound to the localhost, only the local host device can access that application. If you were to run this application on your home desktop, you could open chrome and view the application. The application is hosted on your droplet, however, and you're trying to access the application from your home desktop, which is not the local host.

To properly view this application, you will want to bind the application to 0.0.0.0:xxxx (any free port would work) and you would view the application by going IP_ADDRESS:xxxx. I presume, when you ran it with gunicorn (according to your updated comment below) you set the bind option as "--bind 0.0.0.0:5000".

The "0.0.0.0" address means "no particular address" and, in the context of running applications on a server (the server being the localhost), further means run this application on all IPv4 addresses on the localhost.

neekburm *May 15, 2015*

0 Once I ran it with gunicorn, it worked.

sudogaron *June 15, 2015*

0 Following the setup exactly, I get a 502 Bad Gateway error when going to my server IP address. Now I can access the page if I'm running the "gunicorn test:application -b 0.0.0.0:8000" and it loads fine.

I even followed the comment fixes by using the script section instead to activate the env with the right python settings. Still no go.

Has things changed since this was written?

jellingwood MOD *June 15, 2015*

0 @sudogaron: I just ran through the entire guide again, copying and pasting the commands and changing only the parts highlighted in red. I wasn't able to reproduce your issues.

It sounds like you are having problems with the gunicorn process. Can you verify that the application server is actually running? Check the `/var/log/upstart/myproject.log` file. My guess is that your upstart init file has an issue.

0 This happened to me too. I think it was because i had created a LAMP application with the Apache server interfering with NGinx. I created a new Ubuntu droplet without selecting the Lamp application and it works now.

backhousejnr *June 16, 2015*

0 Justin could you please help me with my setup? I have my python app running nginx and gunicorn successfully on windows azure. However, i have a website running on another instance on azure and would like my python app to be served through the website to show something like www.mywebsite.com/nameofmyapp/. I understand its got to do with proxies. Any guide? Thanks.

tomcek112 *June 25, 2015*

1 I get an error when I try to run "sudo start myproject", saying "sudo: start: command not found". How can I fix this? Thanks for the help!

jellingwood MOD *June 26, 2015*

0 @tomcek112: Upstart, Ubuntu's init system on 14.04, comes with the `start` command to execute init scripts. If you do not have that command, either you are attempting to complete this tutorial on a different operating system or your Ubuntu 14.04 server is in a very unexpected state. Did you start this guide on a fresh Ubuntu 14.04 installation?

chivalry *March 16, 2017*

1 First of all, tyvm for this tutorial. I've gotten further with it than any other in setting up Flask.

I'm having the same problem as @tomcek112. This is under Ubuntu 16.04, a fresh copy from AWS EC2. As a shot in the dark I tried `sudo apt-get start` without luck.

Any suggestions?

jellingwood MOD *March 16, 2017*

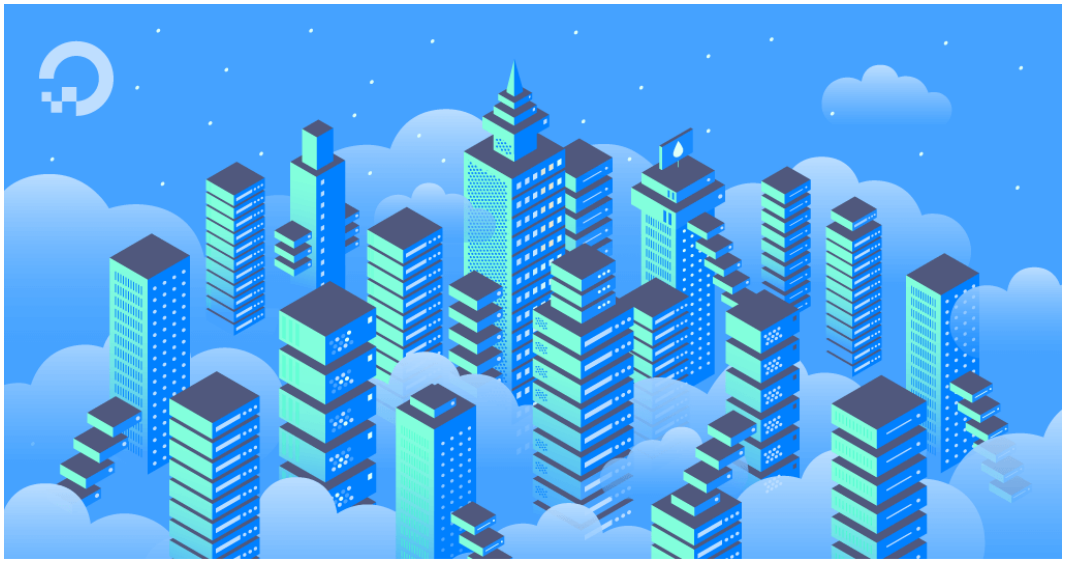
1 @chivalry Hey there.

As the comment you responded to mentions, Ubuntu 14.04 uses the Upstart init system which includes the `start` command. If you're running this on Ubuntu 16.04, that command will be unavailable because that release transitioned away from Upstart, replacing it with systemd. This guide won't function as-is on Ubuntu 16.04 because of that.

So to sort this out, you can either:

- Spin up an Ubuntu 14.04 server instead of Ubuntu 16.04 and follow this tutorial.
- Use the Ubuntu 16.04 version of this guide. The Ubuntu 16.04 version has the instructions to set this up with systemd instead of Upstart.

I hope that helps!



How To Serve Flask Applications with Gunicorn and Nginx on Ubuntu 16.04

In this guide, we will be setting up a simple Python application using the Flask micro-framework on Ubuntu 16.04. The bulk of this article will be about how to set up the Gunicorn application server to launch the application and Nginx to act

chivalry *March 17, 2017*

0 @jellingwood tyvm, I was actually looking at the 16.04 article when I noticed the reply. I'm about to try that out. :)

lequ *June 26, 2015*

0 Strange, I don't think you've done anything wrong. "Start" is a command that might be different on your machine or not installed by default. What are you running? Ubuntu, Debian?

trimagnus *June 27, 2015*

2 For those who are getting timeouts at the first attempt to connect via port 5000 (or whatever is assigned), I got around the issue by adding that port to the ufw firewall, if you set it up per the initial server setup guide.

matmarsiglio *July 14, 2015*

0 I got and error trying to run `sudo start myproject`. It says: `start: Job failed to start`. All the files are exactly like the tutorial. How can I solve this, please?

julianlaval *July 23, 2015*

1 I ran into this issue too - make sure that you are using a non-root account so that the /home/user paths are correct in /etc/init/myproject.conf!

matmarsiglio *August 4, 2015*

0 @julianlaval Thank you! I will work on it now. I hope it works. :)

ktizzel *July 27, 2015*

0 Great writeup thanks a lot! I have a question: How can I set this up so that changes to the Flask application will automatically appear? I tried sudo service nginx restart.

jellingwood MOD *July 27, 2015*

1 @ktizzel: Try reloading your application server instead:

```
$ sudo reload myproject
```

If that doesn't work, you can try restarting it:

```
$ sudo restart myproject
```

Hope that helps!

ktizzel *July 29, 2015*

0 Hi again,

I have another question for you. I am trying to run "sudo reload myproject" using a python script in my flask app. I tried using os.system("sudo reload myproject") but that does not work. Do you know how I would do this?

ktizzel *July 27, 2015*

0 Thanks that worked!

---

igorotak6 *August 9, 2015*

0 How come it only works if i remove the -m 007 ?
edit:
It only works perfect if i run the following without (-m 007) directly as the logged in user.

```
gunicorn --workers 3 --bind unix:myproject.sock  wsgi:app
```

if i use the above in the upstart script, some pages wont work. If i include -m 007, nothing works.
Any ideas?

---

igorotak6 *August 16, 2015*

0 What does `include proxy_params` do?
How can i serve the flask static folder with this setup?

---

jellingwood MOD *August 17, 2015*

0 @igorotak6: The `include proxy_params` reads the contents of the `/etc/nginx/proxy_params` file into that point in the configuration file. Basically, this allows us to set up a lot of basic configuration items without having to go through them one by one.

You can serve static content by adding an additional location block to your `/etc/nginx/sites-available/myproject` file. For instance, if your application looks for static content at the `/static` endpoint, you could adjust your file to look like this:

```
server {
    listen 80;
    server_name server_domain_or_IP;

    location / {
        include proxy_params;
        proxy_pass http://unix:/home/user/myproject/myproject.sock;
    }

    # Assuming your static files are located in `/home/user/myproject/static
    location /static {
        root /home/user/myproject;
```

```
        # If you just want to match by extension you could try something like th
        location ~ \.(jpg|jpeg|png|gif|css|js)$ {
            root /home/user/myproject/static;
        }
    }
```

jordantkrueger *September 23, 2015*

0 I'm getting a 502 error.

Running `gunicorn --bind 0.0.0.0:8000 wsgi:app` worked fine.

One deviation from the tutorial is that I cloned in a project I had been working on locally with github, then installed flask and project dependencies via a requirements.txt file, but I don't think that should cause errors, especially since running gunicorn manually in the virtualenv worked for me.

I tried `sudo tail -f /var/log/upstart/flask-portfolio.log` (my project is called flask-portfolio) and it spit out this:

```
      self.manage_workers()
    File "/home/jkrueger/flask-portfolio/portfolio-env/local/lib/python2.7/site-p
      self.spawn_workers()
    File "/home/jkrueger/flask-portfolio/portfolio-env/local/lib/python2.7/site-p
      time.sleep(0.1 * random.random())
    File "/home/jkrueger/flask-portfolio/portfolio-env/local/lib/python2.7/site-p
      self.reap_workers()
    File "/home/jkrueger/flask-portfolio/portfolio-env/local/lib/python2.7/site-p
      raise HaltServer(reason, self.APP_LOAD_ERROR)
  gunicorn.errors.HaltServer: <HaltServer 'App failed to load.' 4>
```

I've restarted multiple times, and just can't get this working. I'm very new to this, so I could be missing something obvious.

jordantkrueger *September 29, 2015*

0 I think I found what specifically was messing up my deployment.

When I ran gunicorn inside my virtualenv, I needed to stray a little from the tutorial. The tutorial initially had:

I had to change it to:

```
gunicorn --bind 0.0.0.0:8000 wsgi:app
```

I'm not exactly sure why that change was necessary, but I tried it when the tutorial command wasn't working.

Now keeping that in mind, I forgot to make that same change to the gunicorn exec command in the upstart script:

```
exec gunicorn --workers 3 --bind unix:myproject.sock -m 007 wsgi
```

Once I changed it to:

```
exec gunicorn --workers 3 --bind unix:myproject.sock -m 007 wsgi:app
```

Everything started working.

---

pguilford *November 8, 2015*

1 After I restart nginx, in the very last step, when I navigate to my domain I'm shown the nginx welcome screen instead of the flask application. All other checkpoints were successful. Any ideas what I could be missing? Thank you for the tutorial.

KeithWhatling *November 8, 2015*

0 Exactly the same here. I tried restarting, triple checking etc, just get the screen.

KeithWhatling *November 8, 2015*

0 Ok so finally managed to get it working.

```
server {
    listen 80;
    server_name www.yoursitename.com;
    location / {
        include proxy_params;
        proxy_pass http://unix:/home/keith/TrounceEm/TrounceEm.sock;
    }
```

So edit sudo nano /etc/nginx/sites-available/myproject
and restart Ngixn with

```
sudo service nginx restart
```

It did not work straight away, so I did what any English man would do, made some tea, and when I came back!!!! boom! it was working.

Also to make any changes to your flask app work you need to restart mypyproject

```
sudo start myproject
```

phew!

---

donohoe *November 24, 2015*

0 This is an AWESOME tutorial - thank you soo much. I follwed it exactly and it worked flawlessly.

---

joey75d6a5f630e *December 3, 2015*

0 I just wanted to say thank you for this awesome tutorial!

---

samsonnjogu *December 8, 2015*

0 hello please help me ive been trying to use this tutorial to deploy my application but im getting a strange error (111 connection to upstream client refused) please help ive been stuck for days, when i ru with gunicorn --bind 0.0.0.0:8000 wsgi it works what could be the issue

Load More Comments