Deploy flask app with Nginx using Gunicorn

Tasnuva Zaman Follow
Apr 9, 2019 · 6 min read

Today we're going to deploy a micro flask app with **nginx** using **gunicorn.**

Prerequisites:

1. Python
2. python3 venv
3. Flask
4. Nginx
5. Gunicorn

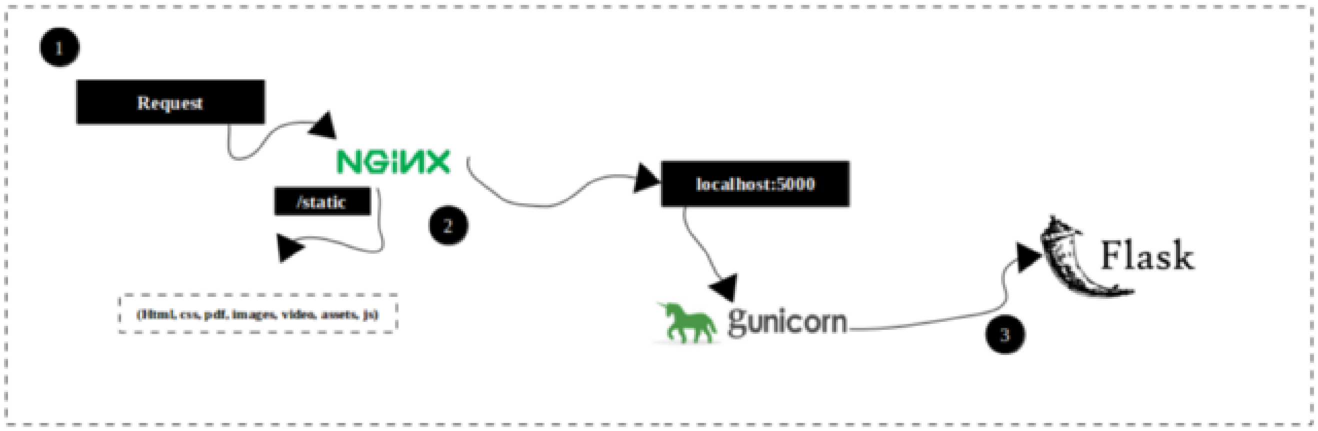
Gunicorn:

> *Green Unicorn (**Gunicorn**) is a Python WSGI server that runs Python web application code. Gunicorn is one of many WSGI server implementations, but it's particularly important because it is a stable, commonly-used part of web app deployments that's powered some of the largest Python-powered web applications , such as **Instagram**.*

ginx:

N NGINX, is an open-source web server that is also used as a reverse proxy, HTTP cache, and load balancer.

*Note: You may need to install **python2-dev** or **python3-dev** tool based on which python version you are using. Anyways, we're going to use python 3.6.*



Architecture

Step 1 || Update your local package index and then install the packages.

First of all, you should update your local package index and then install the packages. To do so follow the below instructions:

For python2:

```
# update your local packages
1.sudo apt-get update

# install dependencies
2. sudo apt-get install python-pip python-dev nginx
```

For python3:

```
# update your local packages
1.sudo apt-get update

# install dependencies
2. sudo apt-get install python3-pip python3-dev nginx
```

Step 2 || Create Venv

Venv is a package comes with Python3. i.e you need not to install venv separately. It serves a similar purpose to `virtualenv` , and works in a very similar way, but it doesn't need to copy Python binaries around (except on Windows). Though `vitualenv` is more popular here I'm using `venv` just for familiarity. To create a venv use below command:

```
python3 -m venv <your venv>

Note: Venv is only for python3 for python2 you should use virtualenv
```

Or Step 2 || Create virtualenv

virtualenv is a very popular tool that creates isolated Python environments for Python libraries.

Install Virtualenv in python2:

```
sudo pip install virtualenv
```

Install Virtualenv in python3:

```
sudo pip3 install virtualenv
```

Create Virtualenv:

```
virtualenv <your virtualenv>
```

Activate virtualenv:

```
source yourvitualenv/bin/activate

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

Step 3 || Set Up a Flask Application

> ***Install Flask and Gunicorn:***

```
pip install gunicorn flask
```

Create a Sample App:

Now as we have installed Flask we can create a flask application. Create a file named `app.py` and paste below content:

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

```
@app.route('/')
def hello_world():
    return "Hello World!"

if __name__ == '__main__':
    app.run(debug=True,host='0.0.0.0')
```

Now, you can test your Flask app by typing:

```
python app.py
```

Visit `http://localhost:5000`

you should see: `Hello World!` in your browser. When you are finished hit CTRL-C in your terminal window to stop the application server.

Step 4 || 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.*

```
nano ~/src/wsgi.py
```

we can simply import the Flask instance from our application and then run it:

```
from app import app

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

> *Save and close the file when you are finished.*

Folder Structure:

```
src
  |____ app.py
  |____ wsgi.py
  |____ myprojectvenv
```

Step 5 || Testing Gunicorn’s Ability to Serve the Project

Now test the ability of Gunicorn to serve the project. We can do it by the name of the module (except .py extension) plus the name of the callable within the application (i.e) wsgi:app. We’ll also specify the interface and port to bind to so that it will be started on a publicly available interface:

```
cd ~/src

gunicorn --bind 0.0.0.0:5000 wsgi:app
```

Visit `http://localhost:5000` you should see: `Hello World!` in your browser again.

Now deactivate `virtualenv` by following command:

```
deactivate
```

Step 6 || Create a `systemd` Unit File

`systemd` unit file will allow Ubuntu's init system to automatically start Gunicorn and serve our 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/app.service
```

```
[Unit]

#  specifies metadata and dependencies

Description=Gunicorn instance to serve myproject
After=network.target

# tells the init system to only start this after the networking
target has been reached

# We will give our regular user account ownership of the process
since it owns all of the relevant files

[Service]

# Service specify the user and group under which our process will
run.
User=yourusername

# give group ownership to the www-data group so that Nginx can
communicate easily with the Gunicorn processes.

Group=www-data

# We'll then map out the working directory and set the PATH
environmental variable so that the init system knows where our the
executables for the process are located (within our virtual
environment).
```

```
WorkingDirectory=/home/tasnuva/work/deployment/src
Environment="PATH=/home/tasnuva/work/deployment/src/myprojectvenv/bin
"

# We'll then specify the commanded to start the service

ExecStart=/home/tasnuva/work/deployment/src/myprojectvenv/bin/gunicor
n --workers 3 --bind unix:app.sock -m 007 wsgi:app

# 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:

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

> ***Note:*** *In the last line of [Service] We tell it to start 3 worker processes. We will also tell it to create and bind to a Unix socket file within our project directory called app.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 and the Python callable within.*

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

```
sudo systemctl start app

sudo systemctl enable app
```

Folder Structure:

A new file `app.sock` will be created in the project directory automatically.

```
src
  |____ app.py
  |____ wsgi.py
  |____ myprojectvenv
  |____ app.sock
```

Final Step || Configuring Nginx

Gunicorn application server is now be up and running and it waits 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.

We'll need to tell NGINX about our `app` and how to serve it.

`cd` into `/etc/nginx/` . This is where the NGINX configuration files are located.

The two directories we will work on are `sites-available` and `sites-enabled` .

- `sites-available` contains individual configuration files for all of your possible static app.
- `sites-enabled` contains links to the configuration files that NGINX will actually read and run.

create a new server block configuration file in Nginx's `sites-available` directory named `app`

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

Open up a server block in which `nginx` will listen to port **`80`** . This block will also be used for requests for our server's domain name or IP address:

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

Let's add a location block that matches every request. In this block, let's 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/tasnuva/work/deployment/src/app.sock;
    }
}
```

Enable `Nginx` server block:

Link the file to the sites-enabled directory to enable Nginx server block we've just created. The syntax is as follows:

```
ln -s <SOURCE_FILE> <DESTINATION_FILE>
```

The actual syntax will look like:

```
sudo ln -s /etc/nginx/sites-available/app /etc/nginx/sites-enabled
```

Note: Test syntax errors by typing: `sudo nginx -t` If there is no issues, restart the Nginx process to read our new config:

```
sudo systemctl restart nginx
```

The last thing we need to do is adjust our firewall to allow access to the Nginx server:

```
sudo ufw allow 'Nginx Full'
```

Now go to your server's domain name or IP address in your web browser , your application is running.

```
http://server_domain_or_IP
```

Congratulations!! Your deployment is done!

Python Nginx Gunicorn Flask DevOps