TUTORIAL

How to Deploy Falcon Web Applications with Gunicorn and Nginx on Ubuntu 16.04

Nginx Python Python Frameworks Ubuntu 16.04

By kevinisaac

Published on November 16, 2016 45.5k

Introduction

Falcon is a minimal Python framework for building web applications. It's well-suited for building APIs that follow the REST architectural style. It's a low-level, high performance framework that tries to do as little as possible without sacrificing development speed.

In this tutorial, you'll build and deploy a Falcon web application. Falcon is a WSGI framework, so you'll install and use Gunicorn, a WSGI application server, to serve the app. Then you'll create a production-ready environment using Nginx as a reverse proxy server to process incoming requests before they reach Gunicorn.

Prerequisites

To complete this tutorial, you will need:

- One Ubuntu 16.04 server set up by following the Ubuntu 16.04 initial server setup guide, including a sudo non-root user and a firewall.

```
$ ssh sammy@your_server_ip
```

Falcon works with both Python 2.x and Python 3.x but we are going to use the latest version of Python available in Ubuntu 16.04 which is Python 3.5.

We'll use *pip* and *virtualenv* to set up our Falcon application. To learn more about these tools, read our tutorial on common Python tools.

First, install virtualenv:

```
$ sudo apt-get install virtualenv
```

Next, create a directory that will hold your application's source code and the virtual environment, and then change to that directory:

```
$ mkdir falcon_app
$ cd falcon_app
```

Then create the virtual environment:

```
$ virtualenv venv -p /usr/bin/python3
```

This command creates a virtual environment inside the directory `venv`. The `-p` flag specifies which version of Python is used in the virtual environment.

You'll see this output:

```
Output
Already using interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in /home/sammy/falcon_app/venv/bin/python3
Also creating executable in /home/sammy/falcon_app/venv/bin/python
Installing setuptools, pkg_resources, pip, wheel...done.
```

To switch back to the system-wide Python interpreter, deactivate the virtual environment by issuing the command:

```
$ deactivate
```

Now that you have set up your Python virtual environment, let's install the required Python packages.

Step 2 — Installing Falcon and Gunicorn with pip

We need to install the `falcon` package, and since we are using Gunicorn to serve our app, we need to install that too. Both of these are available through `pip`,

You can install Falcon one of two ways. Falcon has a binary you can install with `pip install falcon`, but Falcon can get an extra speed boost when compiled with Cython. Issue the following commands to install Cython and then inform Falcon to detect it and compile itself using the system's C compiler:

```
(venv) $ sudo apt-get install build-essential python3-dev
(venv) $ pip install cython
(venv) $ pip install --no-binary :all: falcon
```

Next, install Gunicorn:

```
(venv) $ pip install gunicorn
```

Let's move on to writing our simple Falcon application.

Step 3 — Writing a Simple Web Application Using Falcon

Populate the file with the following content, which creates a Falcon application that displays a simple test message when people visit the `/test` route:

main.py

```
import falcon

class TestResource(object):
    def on_get(self, req, res):
        """Handles all GET requests."""
        res.status = falcon.HTTP_200  # This is the default status
        res.body = ('This is me, Falcon, serving a resource!')

# Create the Falcon application object
app = falcon.API()

# Instantiate the TestResource class
test_resource = TestResource()

# Add a route to serve the resource
app.add_route('/test', test_resource)
```

In this file, we create a class called `TestResource`. This class contains an `on_get` method that defines the response we want to send. Then we create instances of the Falcon API and `TestResource`. Then we add the route `/test` to the API and attach the resource object `test_resource` to it.

Whenever a `GET` request is sent to the `/test` URL, the `on_get()` method of `TestResource` is invoked. The response status and body are set using the variables `res.status` and `res.body` respectively.

Save the file and close your editor. Let's test the application.

Step 4 — Serving a Falcon Application with Gunicorn

Before we go through the work of making our application production-ready by using Nginx, let's make sure our application works by serving it with Gunicorn.

This starts Gunicorn and serves our web application at `0.0.0.0` on port `5000`, as you can see from its output:

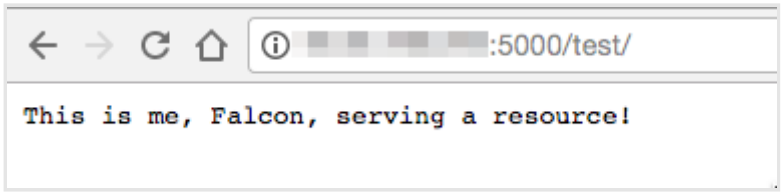
```
Output
[2016-11-14 16:33:41 +0000] [9428] [INFO] Starting gunicorn 19.6.0
[2016-11-14 16:33:41 +0000] [9428] [INFO] Listening at: http://0.0.0.0:5000 (9428)
[2016-11-14 16:33:41 +0000] [9428] [INFO] Using worker: sync
[2016-11-14 16:33:41 +0000] [9431] [INFO] Booting worker with pid: 9431
```

You can use any port number you like, but make sure that it is above `1024` and it's not used by any other program.

The `main:app` option tells Gunicorn to invoke the application object `app` available in the file `main.py`.

Gunicorn provides an optional `--reload` switch that tells Gunicorn to detect any code changes on the fly. This way you can change your code without having to restart Gunicorn.

Test your application by opening your web browser on your local computer, and visiting `http://your_server_ip:5000/test` in your browser. You'll see the following output from your web application:



Stop Gunicorn by pressing `CTRL+C`. Let's set this up in a more production-ready way.

Step 5 — Using Nginx to Proxy Requests to Gunicorn

We'll set up and configure Nginx to proxy all the web requests to Gunicorn instead of letting Gunicorn serve requests from the outside world directly. By doing so, all the

```
(venv) $ sudo apt-get install nginx
```

Next, create a new configuration file called `falcon_app.conf` in the `/etc/nginx/sites-available` directory. This file will configure Nginx to proxy all requests coming to your server's IP address to the Gunicorn server of our Falcon application.

```
(venv) $ sudo nano /etc/nginx/sites-available/falcon_app.conf
```

Add the following contents to the file:

/etc/nginx/sites-available/falcon_app.conf

```
server {
    listen 80;
    server_name your_server_ip_or_domain;

    location / {
        include proxy_params;
        proxy_pass http://localhost:5000;
    }
}
```

This configuration tells Nginx to listen on port `80` and proxy all the HTTP requests to `http://localhost:5000`, which is where Gunicorn will be listening.

Activate this configuration by creating a symbolic link to this file in the `/etc/nginx/sites-enabled` directory:

```
(venv) $ sudo ln -s /etc/nginx/sites-available/falcon_app.conf /etc/nginx/sites-enabled/falcon
```

Then disable the default Nginx configuration file by removing its symlink from the `/etc/nginx/sites-enabled` directory:

```
(venv) $ sudo rm /etc/nginx/sites-enabled/default
```

You'll see this message if you have a working configuration:

```
Output
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

If you see any errors, fix them and test again.

Restart Nginx for the new configuration to take effect.

```
(venv) $ sudo systemctl restart nginx
```

Now start Gunicorn again, but change the listening address from `0.0.0.0` to `localhost` to prevent public access to Gunicorn:

```
(venv) $ gunicorn -b localhost:5000 main:app --reload
```

Allow access to port `80` through the server's firewall if you've enabled it:

```
(venv) $ sudo ufw allow 80
```

Note: If you are using `https` to serve your web application, make sure to allow port `443` using `ufw`. Also, make sure to read our article on How to Secure Nginx Using Let's Encrypt.

Finally, test out the app by visiting `http://your_server_ip/test` and you'll see the same output you saw before.

Notice you no longer need the port number in the URL because your requests are going through Nginx now, which runs on port `80`, the default HTTP port. You'll see the following output in your browser:

Step 7 — Managing Gunicorn with Systemd

We should make sure that our application starts automatically every time our server boots, just like Nginx. If our server was accidentally restarted or had to be rebooted for any reason, we shouldn't have to start Gunicorn manually.

To configure this, we'll create a *Systemd unit file* for our Gunicorn application so we can manage it.

To start, we create a file for our application inside the `/etc/systemd/system` directory with a `.service` extension:

```
(venv) $ sudo nano /etc/systemd/system/falcon_app.service
```

A unit file is made up of sections. The `[Unit]` section is used to specify the metadata and dependencies of our service, including a description of our service and when to start our service.

Add this configuration to the file:

/etc/systemd/system/falcon_app.service

```
[Unit]
Description=Gunicorn instance to serve the falcon application
After=network.target
```

We specify that the service should start *after* the networking target has been reached. In other words, we only start this service after the networking services are ready.

After the `[Unit]` section, we define the `[Service]` section where we specify how to start the service. Add this to the configuration file:

/etc/systemd/system/falcon_app.service

```
ExecStart=/home/sammy/falcon_app/venv/bin/gunicorn --workers 3 -b localhost:5000 main:app
ExecReload=/bin/kill -s HUP $MAINPID
ExecStop=/bin/kill -s TERM $MAINPID
```

We first define the user and group that the service runs under. Then we define a file to store the PID (process ID) of the service; this PID is used to stop or reload the service.

Also, we specify the Python virtual environment, the application's working directory. and the command to execute to start the application. We assign the command to start Gunicorn to the `ExecStart` variable. The `--workers` flag is used to define the number of workers that Gunicorn should start with. The Gunicorn docs suggest you set the number of workers to `2n+1` where `n` is the number of CPU cores. Assuming that your server has a single CPU core, we arrive at the number `3`.

The `ExecReload` and `ExecStop` variables define how the service should be started and stopped.

Finally, we add the `[Install]` section, which looks like this:

/etc/systemd/system/falcon_app.service

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

The `Install` section allows you to enable and disable the service. The `WantedBy` directive creates a directory called `multi-user.target` inside `/etc/systemd/system` and a symbolic link of this file will be created there. Disabling this service will remove this file from the directory.

Save the file, close the editor, and start the new service:

```
(venv) $ sudo systemctl start falcon_app
```

Then enable this service so that every time the server starts, Gunicorn starts serving the web application:

Falcon application, restart the `falcon_app` service:

```
(venv) $ sudo systemctl restart falcon_app
```

To learn more about unit files, read the tutorial Understanding Systemd Units and Unit files.

Conclusion

In this guide, you configured and deployed your first Falcon web application. You set up the Python environment and wrote your application code on the server, then served the web application with Gunicorn. Then you configured Nginx so that it passes web requests to our Gunicorn application. Finally, you wrote a Systemd Unit file and enabled the service so that your web application starts when the server starts.

When you put your own apps into production, you'll want to access them with a host name instead of the IP address. Take a look at How to Set Up a Host Name with DigitalOcean to point your domain name at your server.

About the authors

kevinisaac

has authored 1 tutorial

Brian Hogan

Still looking for an answer?

Ask a question

Search for more help

RELATED

NGINX Config Generator

Easily configure a performant, secure, and stable NGINX server.

Learn More

How To Set Up and Use LXD on Ubuntu 18.04

Tutorial

How To Write Doctests in Python

Tutorial

Comments

2 Comments

Leave a comment...

why do we need virtual environment?

0 Reply Report

Soleil August 15, 2018

0 I create simple python controller in Heroku, thanks for you !!!

here

Reply Report

education, reducing inequality, and spurring economic growth? We'd like to help.

BECOME A CONTRIBUTOR

You get paid; we donate to tech nonprofits.

Featured on Community Kubernetes Course Learn Python 3 Machine Learning in Python Getting started with Go Intro to Kubernetes

DigitalOcean Products Virtual Machines Managed Databases Managed Kubernetes Block Storage Object Storage Marketplace VPC Load Balancers

Company

About
Leadership
Blog
Careers
Partners
Referral Program
Press
Legal
Security & Trust Center

Products

Pricing
Products Overview
Droplets
Kubernetes
Managed Databases
Spaces
Marketplace
Load Balancers
Block Storage
API Documentation
Documentation
Release Notes

Community

Tutorials
Q&A
Tools and Integrations
Tags
Product Ideas
Write for DigitalOcean
Presentation Grants
Hatch Startup Program
Shop Swag
Research Program
Open Source
Code of Conduct

Contact

Get Support
Trouble Signing In?
Sales
Report Abuse
System Status