



## How to Deploy Python WSGI Apps Using Gunicorn HTTP Server Behind Nginx



Posted December 12, 2013

 227.2k

PYTHON

NGINX

### Introduction

Perhaps it was the article on [Python Web Server Comparison](#) tempting you to switch, or the fact that you have simply outgrown your current application deployment stack. You are interested in finding out more about Gunicorn Web Server and want to learn how to deploy a Python application thoroughly from the start.

In this DigitalOcean article, our aim is to help you with all the above and then some. We will begin with expanding our knowledge on the excellent Gunicorn WSGI HTTP Server and continue with deploying Python WSGI web applications built atop various popular frameworks.

### Glossary

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



Sign Up

1. Gunicorn
2. Web Application Deployment Using Nginx

## 2. Preparing Your Droplet for Production

1. Updating the default operating system
2. Setting up Python, pip and virtualenv
3. Creating a Virtual (Python) Environment
4. Downloading and installing Gunicorn
5. Downloading and installing Nginx

## 3. Serving Python Web Applications with Gunicorn

1. WSGI
2. WSGI Application Object (Callable): wsgi.py
3. Running the server
4. Configuring and Optimising Gunicorn
5. Configuring Nginx
6. Miscellaneous Tips and Suggestions

## About Gunicorn and Nginx

### Gunicorn

**Gunicorn** is a stand-alone WSGI web application server which offers a lot of functionality. It natively supports various frameworks with its adapters, making it an extremely easy to use drop-in replacement for many development servers that are used during development.

Technically, the way Gunicorn works is very similar to the successful Unicorn web server for Ruby applications. They both use what's referred to as the pre-fork model. This, in essence, tasks the central [Gunicorn] master process to handle the management of workers, creation of sockets and bindings, etc.

### Gunicorn Server Highlights

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



Sign Up

- Can be used as a drop-in replacement for Paster (Pyramid), Django's Development Server, web2py etc.
- Comes with various worker types and configurations
- Manages worker processes automatically
- HTTP/1.0 and HTTP/1.1 (Keep-Alive) support through synchronous and asynchronous workers
- Supports SSL
- Extensible with hooks
- Python 2.6+ and 3.x support

## Web Application Deployment Using Nginx

Nginx is a very high performant web server / (reverse)-proxy. It has reached its current popularity due to being light weight, relatively easy to work with, and easy to extend (with add-ons / plug-ins). Thanks to its architecture, it is capable of handling *a lot* of requests (virtually unlimited), which - depending on your application or website load - could be really hard to tackle using some other, older alternatives.

**Remember:** "Handling" connections technically means not dropping them and being able to serve them with *something*. You still need your application and database functioning well in order to have Nginx serve clients \*esponses that are not error messages.

### Why use Nginx as a reverse-proxy in front of an application server?

Many frameworks and application servers (including Gunicorn) can serve static files (e.g. javascript, css, images etc.) together with responses. However, the better thing to do is to let a (reverse-proxy) server such as Nginx handle the task of serving these files and managing connections (requests). This relieves a lot of the load from the application servers, granting you a much better overall performance.

As your application grows, you will want to optimize it— and when the time comes, distribute it across servers (VPS) to be able to handle more connections simultaneously (and have a generally more robust architecture). Having a reverse-proxy in front of your application server(s) helps you with this from the very beginning.

Nginx's extensibility (e.g. native caching along with failover and other mechanisms) is also a great feat that benefits web applications unlike (simpler) application servers.

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



Sign Up

```

Client Request ----> Nginx (Reverse-Proxy)
                        |
                        /|\
                    | | `-> App. Server I.   127.0.0.1:8081
                    | | `--> App. Server II. 127.0.0.1:8082
                    `----> App. Server III. 127.0.0.1:8083

```

**Note:** Please see the section *Configuring and Optimising Gunicorn* to learn about number of servers / workers to use.

## Preparing Your Droplet for Production

In this section, we are going to prepare our virtual for production (i.e. for deploying our application).

We will begin with:

- updating the default operating system
- downloading and installing common Python tools (i.e. pip, virtualenv)
- creating a virtual environment to contain the application (inside which its dependencies such as Gunicorn reside)

**Note:** Instructions given here are kept brief. To learn more, check out our how-to article on pip and virtualenv: [Common Python Tools: Using virtualenv, Installing with Pip, and Managing Packages](#).

### Updating the default operating system

**Note:** We will be performing the following setup and preparations on a new droplet, using recent versions of operating systems. In theory, you should not have problems trying them on your VPS. However, if you already actively use it, we highly recommend switching to a new system.

To ensure that we have the latest available versions of default applications, we need to update our system.

**For Debian Based Systems (i.e. Ubuntu, Debian), run the following:**

aptitude update

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




Sign Up

For RHEL Based Systems (i.e. CentOS), run the following:

```
yum -y update
```

## Setting up Python, pip and virtualenv

### Note for CentOS / RHEL Users:

CentOS / RHEL, by default, comes as a very lean server. Its toolset, which is likely to be dated for your needs, is **not** there to run your applications but to power the server's system tools (e.g. YUM).

In order to prepare your CentOS system, Python needs to be set up (i.e. compiled from the source) and pip / virtualenv need installing using that interpreter.

To learn about **How to Set Up Python 2.7.6 and 3.3.3 on CentOS 6.4 and 5.8**, with pip and virtualenv, please refer to: [How to Set Up Python 2.7.6 and 3.3.3 on CentOS](#).

**On Ubuntu and Debian**, a recent version of Python interpreter which you can use comes by default. It leaves us with only a limited number of additional packages to install:

- python-dev (development tools),
- pip (to manage packages),
- virtualenv (to create isolated, virtual environments).

### python-dev:

*python-dev* is an operating-system level package which contains extended development tools for building Python modules.

Run the following command to install **python-dev** using **aptitude**:

```
aptitude install python-dev
```

### pip:

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

✕ we

Sign Up

Run the following commands to install pip:

```
curl https://bitbucket.org/pypa/setuptools/raw/bootstrap/ez_setup.py | python -  
curl https://raw.githubusercontent.com/pypa/pip/master/contrib/get-pip.py | python -  
export PATH="/usr/local/bin:$PATH"
```

You might need sudo privileges.

### virtualenv:

It is best to contain a Python application within its own environment together with all of its dependencies. An environment can be best described (in simple terms) as an isolated location (a directory) where everything resides. For this purpose, a tool called *virtualenv* is used.

Run the following to install virtualenv using pip:

```
sudo pip install virtualenv
```

## Creating a self-contained Virtual (Python) Environment

Having all the necessary tools ready, we can create an environment to deploy our application.

**Remember:** If you haven't got a virtualenv on your development (local) machine for your project, you should consider creating one and moving your application (and its dependencies) inside.

Let's begin with creating a folder which will contain both the virtual environment and your application module:

You can use any name here to suit your needs.

```
mkdir my_app
```

We can continue with entering this folder and creating a new virtual environment inside:

You can also choose any name you like for your virtual environment.

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



Sign Up

Let's create a new folder there to contain your Python application module as well:

This is the folder where your application module will reside.

```
mkdir app
```

And activate the interpreter inside the virtual environment to use it:

Please make sure to use the name you chose for your virtual environment if you went for something other than "myappvenv".

```
source my_app_venv/bin/activate
```

In the end, this is how your main application deployment directory should look like:

```
my_app          # Main Folder to Contain Everything Together
|
|== my_app_venv # V. Env. folder with the Python Int.
|== app        # Your application module
|..
|.
|.
```

## Downloading and Installing Gunicorn

It is always the recommended way to contain all application related elements, as much as possible, together inside the virtual environment. Therefore, we will download and install Gunicorn there.

If you are not working inside an environment, Gunicorn will be installed globally (i.e. available systemwide). This is **not** recommended. Always opt for using **virtualenv**.

To install Gunicorn using pip, run the following:

```
pip install gunicorn
```

## Downloading and installing Nginx

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



Sign Up

The below instructions will not work on CentOS systems. Please see the instructions [here](#) for CentOS.

Run the following command to use the default system package manager *aptitude* install Nginx:

```
sudo aptitude install nginx
```

To run Nginx, you can use the following:

```
sudo service nginx start
```

To stop Nginx, you can use the following:

```
sudo service nginx stop
```

To restart Nginx, you can use the following:

After each time you reconfigure Nginx, a restart or reload is needed for the new settings to come into effect.

```
sudo service nginx restart
```

**Note:** To learn more about Nginx on Ubuntu, please refer to our article: [How to Install Nginx on Ubuntu 12.04](#).

## Serving Python Web Applications with Gunicorn

In this section, we will see how a WSGI application works with Gunicorn. This process consists of providing the server with a *WSGI application callable* (e.g. `application = ( . . )`) as the point of entry.

### WSGI

WSGI in a nutshell is an interface between a web server and the application itself. It exists to ensure a standardized way between various servers and applications (frameworks) to work with each other, allowing interchangeability when necessary (e.g. switching from development to production environment), which is a must have need nowadays.

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



Sign Up



**Note:** If you are interested in learning more about **WSGI** and **Python web servers**, check out our article: [A Comparison of Web Servers for Python Based Web Applications](#).

## WSGI Application Object (Callable): wsgi.py

As mentioned above, web servers running on WSGI need an application object (i.e. your application's).

With most frameworks and applications, this consists of:

- A **wsgi.py** to contain and provide an application object (or callable) to be used by the server.

We will begin with creating an exemplary wsgi.py to use with Gunicorn.

You can choose any name instead of “wsgi.py”. However, these are the ones that are commonly used (e.g. by Django).

Let's begin with creating a wsgi.py file to contain a basic WSGI application.

**Run the following command to create a wsgi.py using the text editor nano:**

```
nano wsgi.py
```

Let's continue with moving (copy/paste) the basic WSGI application code inside (which should be replaced with your own application's callable for production):

```
def application(env, start_response):  
    start_response('200 OK', [('Content-Type', 'text/html')])  
    return ["Hello!"]
```

This is the file that is included by the server and each time a request comes, the server uses this application callable to run the application's request handlers (e.g. controllers) upon parsing the URL (e.g. *mysite.tld/controller/method/variable*).

After placing the application code in, press CTRL+X and then confirm with Y to save this file inside the “my\_app” folder alongside the virtual environment and the app module containing your actual application.

**Note:** This WSGI application is the most basic example of its kind. You will need to replace

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



Sign Up

Once we are done, this is how your main application deployment directory should look like:

```
my_app          # Main Folder to Contain Everything Together
|
|== my_app_venv # V. Env. folder with the Python Int.
|== app         # Your application module
|
|--- wsgi.py    # File containing application callable
|..
|.
|.
```

## Running the server

To start serving your application, you just need to execute:

```
gunicorn [option] [option] .. [wsgi file]
```

Run the following to start the server:

```
gunicorn -b 0.0.0.0:8080 wsgi
```

This will run the server on the foreground. If you would like to stop it, press CTRL+C.

To run the server in the background, run the following:

```
gunicorn -b 0.0.0.0:8080 wsgi &
```

When you run an application in the background, you will need to use a process manager (e.g. htop) to kill (or stop) it.

## Configuring and Optimising Gunicorn

**Note:** Below are some of the most commonly used configuration and optimization settings. To see about all available options, check out the official documentation for [Gunicorn configuration overview](#).

As mentioned earlier, Gunicorn is highly configurable and it is very easy to modify all

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



Sign Up

[!] **Important:** All settings and configuration options listed below are to be chained (put one after the other) to launch gunicorn and to server your application. You cannot modify any of the options after launching the server. Whichever option or option(s) you use, they must be followed by the wsgi file containing the point of entry to your application.

### Example:

```
# Simply running the server (as shown above):  
gunicorn -b 0.0.0.0:8080 wsgi
```

```
# Running the server with five workers:  
gunicorn -b 0.0.0.0:8080 --workers=5 wsgi
```

### Worker Count

In general, it is considered (and accepted) that applications are rather I/O bound than CPU bound. What this means is, the bottleneck is not caused by the processing power your virtual server has, but instead by the disks. The idea is: when a worker is busy with the disk operations, another still utilizes the CPU dealing with requests.

Therefore, the suggested amount of workers to have is usually set around with the following simple formula:

```
# (2 Workers * CPU Cores) + 1  
# -----  
# For 1 core  -> (2*1)+1 = 3  
# For 2 cores -> (2*2)+1 = 5  
# For 4 cores -> (2*4)+1 = 9
```

You can specify the amount of workers by passing the argument `--workers=[n]`.

Usage:

```
# Example: gunicorn --workers=[number of workers]  
gunicorn --workers=5
```

**Note:** The above scenario does not strictly apply to non-blocking workers.

### Socket Settings

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



Sign Up

```
# Example: gunicorn -b [address:port]
gunicorn -b 127.0.0.1:8080
```

**Note:** When an application is set to listen for incoming connections on `127.0.0.1`, it will only be possible to access it locally. If you use `0.0.0.0`, however, it will accept connections from *the outside* as well.

## Selecting The Worker Type

Gunicorn, as we have discussed, offered the possibility to work with various type of workers.

For the majority of deployments, the standard worker type - sync - will be sufficient and comes by default, therefore not requiring any explicit setting.

For other workers, please note that you will need to install them as dependencies.

Usage:

```
# Example: gunicorn -k [worker]
gunicorn -k sync    # or;
gunicorn -k gevent # ..
```

Available types:

- sync
- eventlet
- gevent
- tornado

## Number of Simultaneous Connections for Eventlet / Gevent

To modify number of simultaneous connections for Eventlet and Gevent workers:

```
# Example: gunicorn -k [worker] --worker-connections [number]
gunicorn -k gevent --worker-connections 1001
```

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



Sign Up

If you would like to set explicitly the file to write access logs:

```
# Example: gunicorn --log-file [file]
gunicorn --log-file error_logs.log
```

By default, this option is set to *None*.

## Error Logs

In order to specify a file to write error logs to, use this setting.

```
# Example: gunicorn --access-logfile [file]
gunicorn --access-logfile acclogs
```

## Log Level

This is used to set the granularity of error log outputs. Possible options are:

- debug
- info
- warning
- error
- critical

Usage:

```
# Example: gunicorn --log-level [level]
gunicorn --log-level error
```

## Process Naming

If you are using utilities such as *top* to monitor your processes, you can use this setting to give them a more meaningful name which should help you with the task.

```
# Example: gunicorn --name [name]
gunicorn --name my_application
```

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



Sign Up

# Configuring Nginx

After learning about configuring and running Gunicorn, we now need to do the same with Nginx to talk with the Gunicorn server running the application. For this, we need to modify Nginx's configuration file: `nginx.conf`

**Run the following command to open up “`nginx.conf`” and edit it using nano text editor:**

```
sudo nano /etc/nginx/nginx.conf
```

Afterwards, you can replace the file with the following example configuration to get Nginx work as a reverse-proxy, talking to your application.

**Note:** To learn about incorporating SSL support, please read this article first: [Creating an SSL certificate on Nginx.](#)

**Example configuration for web applications:**

```
worker_processes 1;

events {

    worker_connections 1024;

}

http {

    sendfile on;

    gzip                on;
    gzip_http_version 1.0;
    gzip_proxied        any;
    gzip_min_length     500;
    gzip_disable        "MSIE [1-6]\.";
    gzip_types          text/plain text/xml text/css
                        text/comma-separated-values
                        text/javascript
                        application/x-javascript
                        application/atom+xml;
```

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



Sign Up

```
server 127.0.0.1:8080;
# server 127.0.0.1:8081;
# ..
# .

}

# Configuration for Nginx
server {

    # Running port
    listen 80;

    # Settings to serve static files
    location ^~ /static/ {

        # Example:
        # root /full/path/to/application/static/file/dir;
        root /app/static/;

    }

    # Serve a static file (ex. favico)
    # outside /static directory
    location = /favico.ico {

        root /app/favico.ico;

    }

    # Proxy connections to the application servers
    # app_servers
    location / {

        proxy_pass            http://app_servers;
        proxy_redirect         off;
        proxy_set_header       Host $host;
        proxy_set_header        X-Real-IP $remote_addr;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header        X-Forwarded-Host $server_name;

    }

}

}
```

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



Sign Up

When you are done modifying the configuration, press CTRL+X and confirm with Y to save and exit. You will need to restart Nginx for changes to come into effect.

**Run the following to restart Nginx:**

```
sudo service nginx stop
sudo service nginx start
```

**Note:** To learn more about Nginx, please refer to our article: [How to Configure Nginx Web Server on a VPS.](#)

## Miscellaneous Tips and Suggestions

### Firewall:

- [Setting up a firewall using IP Tables](#)

### Securing SSH:

- [How To Protect SSH with fail2ban on Ubuntu](#)
- [How To Protect SSH with fail2ban on CentOS 6](#)

### Creating Alerts:

- [How To Send E-Mail Alerts on a CentOS VPS for System Monitoring](#)

### Monitor and Watch Server Access Logs Daily:

- [How To Install and Use Logwatch Log Analyzer and Reporter](#)

Submitted by: [O.S. Tezer](#)

♥ Upvote (14)

✚ Subscribe

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



Sign Up



## Write about Chef - We'll donate to a tech nonprofit

Partner with us to publish an article on Chef and we'll donate up to \$300 to a charity of your choice.

[LEARN MORE](#)

---

### Related Tutorials

How to Deploy Python WSGI Applications Using a CherryPy Web Server Behind Nginx

How to Install and Configure a LEMP Stack using Software Collections on CentOS 7

How To Install and Secure phpMyAdmin with Nginx on Ubuntu 16.04

How to Block Unwanted SSH Login Attempts with PyFilter on Ubuntu 16.04

How To Host a Website Using Cloudflare and Nginx on Ubuntu 16.04

---

## 8 Comments


Leave a comment...

[Log In to Comment](#)


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




[Sign Up](#)

^  How old is this article?  
0

---

^ [etel](#) MOD January 9, 2014  
 0 This article was published on December 12th

---

^ [spaesani](#) July 18, 2014  


1 Ok so we can start a python app from the command line (say for practice):  
gunicorn [option] [option] .. [wsgi file]

and we can start the gunicorn server in the background (say for actually handling python app requests):

```
gunicorn -b 0.0.0.0:8080 wsgi ^&
```


and we can configure nginx to pass requests for ? which mime type ? to an application server at 127.0.0.1 ???

```
upstream app_servers {  
server 127.0.0.1:8080;  
# server 127.0.0.1:8081;
```


I may be off here but it seems the nginx config file example is inaccurate and lacks comments that are specific to the focus of the tutorial: handling python file/app requests.

Correct me, the tutorial or whomever you so please if any are wrong but by all means, take your time :).


---

^ [rossk](#) December 23, 2014  
 0 What's the best way to get gunicorn to automatically start up?

---

^ [yansong10101](#) February 19, 2015  
 1 it doesn't work

---

^ [Lanti](#) April 9, 2015  


0 I got the following error on Win7 when I try to run the gunicorn server inside virtualenv. Somebody can help?

```
(myappvenv) C:\www\python\myapp> gunicorn -b 0.0.0.0:8080 wsgi
```

```
Traceback (most recent call last):
```

```
File "C:\Python34\Lib\runpy.py", line 170, in _runmoduleasmain
```

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



Sign Up

```
exec(code, run_globals)
File "C:\www\python\myapp\myappenv\Scripts\gunicorn.exe_main.py", line 5, in <module>
File "C:\www\python\myapp\myappenv\lib\site-packages\gunicorn\app\wsgiapp.py", line 10, in
<module>
from gunicorn.app.base import Application
File "C:\www\python\myapp\myappenv\lib\site-packages\gunicorn\app\base.py", line 12, in
<module>
from gunicorn import util
File "C:\www\python\myapp\myappenv\lib\site-packages\gunicorn\util.py", line 9, in <module>
import fcntl
ImportError: No module named 'fcntl'
```

---

 [shiweistg](#) June 15, 2017

0 [@Lanti](#) Looks like Gunicorn is not supported on Windows yet. See <https://github.com/benoitc/gunicorn/issues/1005> and <https://github.com/benoitc/gunicorn/issues/524>

---

 [shiweistg](#) June 15, 2017

0 I think the example codes for access log and error log configurations are mixed up.  
This is what I see:

#### Access Logs

If you would like to set explicitly the file to write access logs:

```
# Example: gunicorn --log-file [file]
gunicorn --log-file error_logs.log
By default, this option is set to None.
```

#### Error Logs

In order to specify a file to write error logs to, use this setting.

```
# Example: gunicorn --access-logfile [file]
gunicorn --access-logfile acclogs
```

---

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



Sign Up



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



Copyright © 2018 DigitalOcean™ Inc.

[Community](#) [Tutorials](#) [Questions](#) [Projects](#) [Tags](#) [Newsletter](#) [RSS](#) 

[Distros & One-Click Apps](#) [Terms, Privacy, & Copyright](#) [Security](#) [Report a Bug](#) [Write for DOnations](#) [Shop](#)

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



Sign Up