Django Server Comparison: The Development Server, Mod_WSGI, uWSGI, and Gunicorn

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: *http://faq.asphosthelpdesk.com/article.php?id=252*

When it comes to Django Servers there are quite a few choices. It can be difficult to know which setup is right for you. While each setup is different, they each come with their own advantages and drawbacks. In this article, we will attempt to explore the most common Django Servers and point out the advantages and disadvantages associated with them.

One - The Django Development Server

The Django development server comes packaged with Django and can be run with the following command:

```
django-admin.py runserver
```

This launches a lightweight web server running by default using localhost on port 8000. You can modify this by passing something like the following in as an added parameter:

```
django-admin.py runserver 10.0.0.150:8001
```

This will make it so that the web server is serving up your Django application using the IP 10.0.0.150 on port 8001. While there are various other options you can pass in, the above is the most common.

The Django development server is just that, a development server. It was not meant for a production environment, as stated in the official Django documentation here. What it is meant for, however, is to be used as an easy way to test your application in a development environment without the added overhead of installing/configuring a full on web server.

The Django development server shines in that it is light weight and dead simple to use. It serves up your static files without needing to collect them to any specific location as well. The development server also restarts each time you save your files. This is helpful because most all web servers for Python cache your app to get rid of the need for startup times, but require a restart if the code changes so a new copy can be cached.

The obvious downside to the Django development server is that it is not production ready and should only be used for development purposes. It is not made to handle lots of requests or load at any given time. Other than this, it is a great option for those seeking a fast way to get up and running with Django.

Two - Mod_WSGI

Mod_wsgi is the most popular Python WSGI adapter module for Apache and is a highly recommended method if you are using Apache as your web server.

Once you have mod_wsgi installed it is fairly simple to implement. Just add the following line to your Apache VirtualHost entry:

```
WSGIScriptAlias /yourapp /opt/yourenv/yourapp.wsgi
```

The first argument you give to the WSGIScriptAlias directive is the URL mount point. So if this VirtualHost entry was for the domain mydomain.com then your application would be displayed at http://mydomain.com/yourapp.

If your Django app is located outside of the directories Apache is configured to be accessible to Apache, then you will need to add the following to your VirtualHost entry as well:

```
<Directory /opt/yourenv>
Order allow,deny
Allow from all
</Directory>
```

Note that if you want to mount your WSGI application at the root of your domain you can use "/", but this comes with a small caveat. When you do this your static files found in your DocumentRoot are no longer served by Apache, but rather by the WSGI application. To get around this simply map your static files using the "Alias" directive like so:

```
Alias /static/ /opt/yourenv/static/
```

Just like with the WSGIScriptAlias directive, the first argument is the mount point. The second option is the path to the static directory where your files live.

The positives of mod_wsgi are that it was built to integrate fairly seamlessly with Apache. This is great for anyone who uses Apache as their primary web server.

The down side to mod_wsgi is that it only works with Apache. So if you prefer or require another web server such as NGINX, Lighttpd, Cherokee, etc. then you're out of luck. It also isn't as light weight as some of the other options we will discuss.

Three - uWSGI

uWSGI is a server that implements the WSGI protocol in order to communicate with other web servers such as NGINX, Apache, Cherokee, etc. Its end goal is to handle the interpreting of your Python code, while a web server such as the ones mentioned in the previous sentence handles static file and other requests. uWSGI was written for Python so installation is as easy as running the following command:

```
sudo pip install uwsgi
```

The implementation of uWSGI for Django is fairly simple. The Django project site has great documentation on this subject and can be found here.

Once you have uWSGI installed and running as shown in the Django documentation, all that is left is to proxy to uWSGI from your main web server. How to do this will depend on your web server, but is typically very straightforward.

The plus side to using uWSGI is that it is very light weight, runs separately from your web server (so as not to overload your web server processes), and is relatively easy to set up.

A few of the cons are the overhead of having to set up another server apart from your web server, and needing to proxy from your main web server to uWSGI. If you don't mind getting your hands a little dirty in configuration though, uWSGI is a great option.

Four - Gunicorn

Gunicorn is a Python WSGI HTTP server much like uWSGI. I personally use Gunicorn, but that does not necessarily mean it's the best. I use Gunicorn for it's simple setup and easy integration with Django. It is installed much the same as uWSGI with the following command:

```
sudo pip install gunicorn
```

As with uWSGI, you must proxy to Gunicorn from your main web server. This is no easier to do for Gunicorn than it is for uWSGI.

Gunicorn is also very light weight. Whether it is faster than uWSGI is very much up for debate. Much of this has to do with how you configure Gunicorn or uWSGI. Both can reach very impressive levels of performance, though some have mentioned that Gunicorn works better under high load.

Drawbacks to Gunicorn are much the same as uWSGI, though I personally have found Gunicorn to be more easily configurable than uWSGI. It still isn't as quick or simple to configure and set up as using mod_wsgi with Apache, but on a performance level there is no comparison.

Summary

There are still many other options that we didn't cover like Flup, FastCGI, mod_python, etc.

Flup is not very widely used so I wouldn't recommend it. FastCGI is more for a shared hosting environment than something where you are not sharing web servers or resources. Mod_python for Apache does work, but in most any documentation you find mod_wsgi will be the recommended route as it simply works much better.

So which method is the winner? Well, that honestly depends on what you're after. If you like Apache and a simple setup and integration then by all means use mod_wsgi. If you don't use Apache then I would recommend either uWSGI or Gunicorn. Both have their merits and will work great for any Django application. And of course in a development environment, there is simply no reason not to use the built in Django development server!