Django Runserver Is Not Your Production Server

You've built your Django web app and are working on deploying it.

You've been running your app locally with `python manage.py runserver`. That's a fine command, built for development convenience, but it's not meant to be used as part of a production setup.

The docs are very adamant about this:

> *DO NOT USE THIS SERVER IN A PRODUCTION SETTING. It has not gone through security audits or performance tests.*

So the server started with `runserver` is not guaranteed to be performant (it's very slow), and it hasn't been built with security concerns in mind. Not a good fit for production use.

So, what's the right way to approach this?

A Production Stack

You want to only use tech in production, which is reliable, well tested and has been around for a while.

A production setup usually consists of multiple components, each designed and built to be really good at one specific thing. They are fast, reliable and very focused.

When a request arrives at your server, it should be passed to a dedicated **web server**. Nginx is an example for a good web server.

This is an application, which is great at serving static files from disk (your css and js files for example) and handling multiple requests at once. If the request is not for a static file, but should be processed by your application, the webserver is configured to pass this request to the next component.

The next component is an **application server**. It gets those fancy requests and uses them to construct Python objects which are usable by Django. **WSGI** is a specification which people agreed on, which describe how that happens. Gunicorn is an example for a WSGI server.

> *Curious about how Nginx and Gunicorn work together? Read more here*

How Does Django Fit In?

Your Django app does not actually *run* as you would think a server would - waiting for requests and reacting to them. Your project provides a `uwsgi.py` file, which contains a function to be called by the application server. This function gets a Python object representing the incoming request.

This function calls your code, and produces a response object which is passed to the WSGI server. There the response is translated into a HTTP response and is passed back to the web server, which finally delivers it to the user.

In Conclusion

If you want to run Django in production, be sure to use a production-ready web server like Nginx, and let your app be handled by a WSGI application server like Gunicorn.

If you plan on running on Heroku, a web server is provided implicitly. You don't have to take care of it. You just need to specify a command to run your application server (again, Gunicorn is fine) in the Procfile.

See Also

- 11 Django Real-World Challenges Your Tutorial Didn't Mention
- Django Systemd Crashcourse

> Gunicorn and Nginx in a Nutshell

> What Is Gunicorn, and What Does It Do?

> AWS Services You Should Know When Deploying Your Django App