Deploying Django Documentation

Release 0.1.1

Ben Lopatin

Sep 27, 2017

Contents

Moving from Django's *runserver* management command on your laptop to a live webserver can be one of the trickiest steps for newcomers to Django. And not just newbies. There is no one right way to deploy your application, and the rest of the community benefits from learning about better practices.

What follows is an attempt to collect and distill some of these practices in a digestible and applicable way.

This guide is primarily for people deploying to their own servers. Much of the material will apply if you are deploying to a platform as a service, but this is not primarily a PaaS deployment guide. It also assumes you are deploying to a Linux or otherwise Unix like system (unless someone wants to contribute to a Windows guide of course!). This guide should provide both a roadmap to people getting started and a reference for more experienced developers.

CHAPTER 1

Proposed outline

Getting started

From runserver to N-tier architectures

This guide aims to address two questions:

1. How do you serve a Django app in production?
2. How do you get updates from your computer to your server

There are lots of related topics and advanced methods, and while this guide will cover some and mention many, the primary goal is to give people a solid footing just answering these two primary questions.

A production server

You might have noticed this in the Django docs with regard to the runserver management command:

> DO NOT USE THIS SERVER IN A PRODUCTION SETTING

It can be confusing at first, but what's really meant is this:

> DO NOT USE THIS SERVER IN A PRODUCTION SETTING

It usually makes more sense the second time around. This is an unoptimized, minimally tested tool available solely for making your application available in development.

In production what you want is a dedicated WSGI server, one designed and configurable for a production environment and controlled by some kind of process manager.

Moving code to production

You'll need to move your code from your computer, from your repository, to the production server. More, you'll want to do this periodically as you fix bugs, release new features, etc. You could certainly use FTP to move code from one

place to another, but this has some serious downsides. You can't distinguish between releases. There's way too much manual intervation required.

What you really want is a way to smoothly move code, run deployment tasks, and ensure the server uses your new code. You can do better than FTP.

WSGI servers

For each:

- overview
- pros/cons, benefits/drawbacks
- recipes for using
- add'l references

mod_wsgi

gunicorn

uwsgi

Twisted

Web servers

Apache

Nginx

Django Settings management

What *not* to include

Passwords, auth keys, etc.

Multiple settings files

Per environment/host settings files

Using the system environment

A distinct or compatible option

Platform as a Service

Heroku

dotCloud

Gondor

Going live

6. Process management
7. Releases
8. Python dependencies
9. Static assets

Taking control

10. Logging & exceptions
11. Backing services
12. Securing your Django deployment

Advanced deployment options

13. Configuration management
14. Packaging your application
15. Application containers

Deployment recipes

Sample configurations, from Nginx configuration, gunicorn conf files, Fabric scripts, etc.

CHAPTER 2

License

CHAPTER 3

Contributing

Contributions are welcome, whether new content, technical corrections, or just typo fixes. Just as an open source coding project benefits from a consistent coding style, so does the guide benefit from a consistent writing style and voice.

CHAPTER 4

Indices and tables

- genindex
- search