django-layout Documentation

Release 0.1

Joseph Mornin

October 27, 2016

Contents

Contents:

CHAPTER 1

Introduction

This is a reusable Django project template. These instructions explain how to configure development, staging, and production environments.

This setup uses:

- Ubuntu Server 12.04.03 LTS 64-bit
- Python 2.7.3
- Django 1.6.1

See `requirements.txt` for other dependencies. You can modify this setup, but I haven't tested other configurations.

1.1 Project Settings

By default, Django projects use a single settings file. This is fine for development environments, but in production you'll want different settings (for instance, to set `Debug = False`).

In this configuration, project-wide settings live in `default.py` and settings for specific enviroments (development, staging, and production) live in `development.py` and `production.py`. The instructions below explain how to configure settings for each environment.

CHAPTER 2

Development

These instructions assume your project is called `project_name`. You can call it whatever you want, but be sure to replace `project_name` with your project's name.

2.1 Configure Virtualenv

The project should live inside a virtual environment. Virtualenvs help keep projects clean and portable.

First, install and configure virtualenv and virtualenvwrapper.

Next, configure your local virtualenv:

```
$ mkvirtualenv -p python2.7 project_name
$ cd /path/to/virtualenvs/project_name
```

2.2 Create New Django Project

Make sure you're in directory and that the virtualenv is activated.

First, install Django 1.6.1:

```
$ pip install Django==1.6.1
```

Next, create a new Django project based on the `django-layout` template (replacing `project_name` at the end of the command with the name of your project):

```
$ django-admin.py startproject --template=https://github.com/morninj/django-layout/archive/master.zip
```

To keep your virtualenv organized, rename the project directory as `src`:

```
$ mv project_name src
$ cd src
```

Install dependencies:

```
$ pip install -r requirements.txt
```

2.3 Initialize Repository

The contents of `src` should be under version control. For Git, run:

```
$ git init
$ git add * .gitignore
$ git commit -m "Create new Django project"
```

To store your code on GitHub, create a new GitHub repository and then run:

```
$ git remote add origin git@github.com:your_username/repo_name.git
$ git push -u origin master
```

This is also the spot to add a `README`.

2.4 Configure Settings

Go to the settings folder:

```
$ cd project_name/project_name
```

Project-wide settings live in `default.py`. Edit that file to match your configuration. The defaults should be fine for most configurations.

Development settings live in `development.py`. The defaults should work for most development environments.

Next, activate the development settings:

```
$ cp settings.sample.py settings.py
```

Create the local database:

```
$ cd ..
$ python manage.py syncdb
```

Run the development server:

```
$ python manage.py runserver
```

Your project should now be available at `http://127.0.0.1:8000/`.

CHAPTER 3

Staging Configuration

First, commit all local changes to the repository.

Next, launch a new staging server. On Amazon Web Services, for instance, launch a new EC2 instance. Remember that this configuration uses Ubuntu Server 12.04.03 LTS 64-bit (though other Debian-based Linux distributions should work fine).

The staging server will be almost identical to the production server. Most of the settings below will also apply in production.

3.1 Edit Configuration Settings

Change to the settings directory:

```
$ cd /path/to/virtualenvs/project_name/src/project_name/project_name
```

Edit the settings in `production.py` to match the settings for your staging server. For instance, you may want to add credentials for a database server. Also, don't forget to add your staging and production domains to `ALLOWED_HOSTS`–e.g.:

```
ALLOWED_HOSTS = ['staging.example.com', 'www.example.com']
```

Next, change to the `conf` directory:

```
$ cd ../conf
```

Edit the following files and make sure the values are correct:

- `production.py`: production deployment configuration
- `nginx.staging.conf`: nginx virtual host configuration
- `launch.sh`: a shell script to launch the Gunicorn server
- `livesite.conf`: an Upstart configuration to launch Gunicorn on boot

Commit your changes and push them to the repository.

3.2 Configure Staging Server

Once you've specified the settings above, Fabric will automatically configure the server environment. To configure the staging server, run:

```
$ cd .. # you should now be in the same directory as fabfile.py
$ fab configure_staging
```

Fabric will show the output of each command. You may be prompted for passwords (e.g., to log into the server or to clone the repository).

3.3 Verify

Navigate to the staging server address. You should see `Hello, world!`

If you see "Bad Request (400)", it's probably because `ALLOWED_HOSTS` is set incorrectly. Make sure your domain is in `ALLOWED_HOSTS` in `src/project_name/project_name/production.py`.

CHAPTER 4

Production Configuration

CHAPTER 5

Workflow

5.1 Development

```
$ cd /path/to/virtualenv/
$ workon project_name
$ cd src/project_name
$ git pull origin master # Pull changes
$ python manage.py runserver
... # Make changes
$ git commit -am "Description of changes"
$ git push origin master
```

When adding an app, create the initial schema migration with South:

```
$ python manage.py app_name --initial
```

When updating an app's models, use South to migrate the database schema:

```
$ python manage.py schemamigration app_name --auto
$ python manage.py migrate app_name
```

5.2 Staging

```
$ fab deploy_staging
```

5.3 Production

TODO

CHAPTER 6

Credits

Joseph Mornin is the main author.

This project builds on parts of Lincoln Loop's django-layout.

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*