Wagtail Cookiecutter Foundation Documentation

Release 0.1

Christopher Clarke

Oct 04, 2018

Getting Started

A cookiecutter template for Wagtail CMS was built using Zurb Foundation front-end framework. A demo of a default project generated from this cookiecutter is available at http://wagtail.chrisdev.com.

CHAPTER 1

Getting Started

Here is how we create a new Django project quickly while letting cookiecutter to do all the work.

To get started we assume the following dependencies

```
pip
virtualenv/pyvenv/virtualenvwrapper
PostgreSQL
Bower
```

Get Cookiecutter

```
$ pip install cookiecutter
```

Now run it against this repo:

```
$ cookiecutter https://github.com/chrisdev/wagtail-cookiecutter-foundation.git
```

You'll be prompted to provide some values for your project.

Enter the project

```
$ cd wagtail_project/
```

Create a git repo and push it there:

```
$ git init
$ git add .
$ git commit -m "first awesome commit"
$ git remote add origin git@github.com:cclarke/my_site.git
$ git push -u origin master
```

To create your project's virtual environment, install all pip dependencies, create the development database, run migrations and load initial data to database, install front-end dependencies and finally start the development server for you run

```
make develop_env
```

You can access your site at `http://localhost:8000`. The Admin back-end is available at `http://localhost:8000/admin/`.The default Admin username is *admin* and The default Admin password is *admin123*.

CHAPTER 2

Pre-Requisites

To get started we assume the following dependencies

- pip
- virtualenv/pyvenv/virtualenvwrapper
- PostgreSQL
- Bower

CHAPTER 3

Creating A Project

Let's pretend you want to create a Django project called "wagtail_project". Rather than using *startproject* and then editing the results to include your name, email, and various configuration issues that always get forgotten until the worst possible moment, get cookiecutter to do all the work.

First, get Cookiecutter

```
$ pip install cookiecutter
```

Now run it against this repo:

```
$ cookiecutter https://github.com/chrisdev/wagtail-cookiecutter-foundation.git
```

You'll be prompted for some values. Provide them, then a Django project will be created for you

```
Cloning into 'wagtail-cookiecutter-foundation'...
remote: Counting objects: 5849, done.
remote: Compressing objects: 100% (129/129), done.
remote: Total 5849 (delta 47), reused 0 (delta 0), pack-reused 5718
Receiving objects: 100% (5849/5849), 12.43 MiB | 2.64 MiB/s, done.
Resolving deltas: 100% (3291/3291), done.
Checking connectivity... done.
project_name [Wagtail Project]:
project_slug [wagtail_project]:
Select version_control_system:
    1 - git
    2 - hg
    Choose from 1, 2 [1]:
Select vcs_host:
    1 - bitbucket.org
    2 - github.com
    Choose from 1, 2 [1]:
your_bitbucket_or_github_user_name [chrisdev]:
author_name [Christopher Clarke]:
email [cclarke@chrisdev.com]:
description [A short description of the project.]:
```

```
timezone [UTC]:
production_host_name [wagtail.chrisdev.com]:
version [0.1.0]:
use_letsencrypt [y]:
use_celery [y]:
use_opbeat [n]:
use_django_cachalot [n]:
use_wagalytics_app [n]:
staging_host_name [wagtail-staging.chrisdev.com]:
use_vagrant_staging [True]:
deploy_user_name [django]:
database_user_name [django]:
django_admin_user [my_wagtail_admin]:
Select open_source_license:
    1 - MIT
    2 - BSD
    3 - Apache Software License 2.0
    4 - Not open source
    Choose from 1, 2, 3, 4 [1]:
```

Enter the project and take a look around:

```
$ cd wagtail_project/
$ ls
```

Create a git repo and push it there:

```
$ git init
$ git add .
$ git commit -m "first awesome commit"
$ git remote add origin git@github.com:cclarke/my_site.git
$ git push -u origin master
```

CHAPTER 4

Manual Installation

Firstly, open up a command line shell in your new projects directory.

1. **Create a virtual environment**

   Linux/Mac OSX: `pyvenv venv`

   Windows: `c:\Python34\python -m venv myenv`

   Python 2.7

   `pyvenv` is only included with Python 3.3 onwards. To get virtual environments on Python 2, use the `virtualenv` package:

   ```
   pip install virtualenv
   virtualenv venv
   ```

   Virtualenvwrapper

   virtualenvwrapper provides a set of commands which makes working with virtual environments much more pleasant. It also places all your virtual environments in one place.

   To install (make sure virtualenv is already installed):

   ```
   pip install virtualenvwrapper
   export WORKON_HOME=~/Envs
   source /usr/local/bin/virtualenvwrapper.sh
   mkvirtualenv venv
   ```

   Windows:

   ```
   pip install virtualenvwrapper-win
   mkvirtualenv venv
   ```

2. **Activate the virtual environment**

   Linux/Mac OSX: `source venv/bin/activate`

   Windows: `venv/Scripts/activate.bat`

Virtualenvwrapper: `workon venv`

3. **Install PIP requirements**

   `pip install -r requirements/dev.txt`

4. **Create the database*

   By default require `PostgreSQL` to be installed

```
createdb my_site
```

5. **Load the Initial Data** The cookiecutter comes with some pages already created for your convenience including the Homepage with a working `bx_slider` slide show, contact page, events and news/blog pages. To generate these pages run:

   `psql -d my_site -f ansible/roles/web/files/initial_data.sql`

   The default Admin username is *admin*

   The default Admin password is *admin123*

   To copy the media directory to project root.

   `./manage.py copy_media`

6. **Install Packages (Foundation, Font-Awesome etc.) using Bower package manager**

   We use bower for front-end dependency management. To install front dependencies use

   `bower install`

   This will install the supported version of Zurb Foundation, Font Awesome and bxSlider as well as their dependencies.

7. **Run the development server**

   `./manage.py runserver`

   Your site is now accessible at `http://localhost:8000`, with the admin backend available at `http://localhost:8000/admin/`.

CHAPTER 5

Styling your Project

The projects generated with this cookiecutter include a scss file, named `app.scss`. You can find the settings file under `<project_slug>/pages/static/css/`.

Every component includes a set of variables that modify the core structural or visual styles. If there's something you can't customize with a variable, you can just write your own CSS to add it.

We are now using colors $primary, $secondary, $tertiary, so changing the look of your site has now become simpler. Change these main colors can give your project a whole new look within seconds. These are the default scss colors of your project.

```
$primary: #3F51B5;
$secondary: #303F9F;
$tertiary: #FF4081;
$off-white : #F2F2F2;
$tundora: #444;
$mine-shaft: #333;
$dark-metal: #222;
```

By default your project is set:

- `$primary` - Topbar, Header Backgrounds and Links
- `$secondary` - Social Bar and Sub-headers
- `$tertiary` - Buttons and Hovers
- `$tundora` - Headers and Paragraph

For further documentation on using sass check out Foundation SASS.

CHAPTER 6

Template Tags

The listing of template tags can be found in `{{project_slug}}/utils/templatetags/{{project_slug}}_utils.py`. From there they can be modified.

Top Menu to Offcanvas

To switch to the regular top menu to the foundation offcanvas menu change `{% top_menu parent=site_root calling_page=self %}` to `{% offcanvas_top_menu parent=site_root calling_page=self %}` in the file `{{project_slug}}/pages/templates/base.html`.

Upcoming Events

The template tag `{% upcoming_events %}` is a feed of upcoming events in the order of upcoming dates by default. The count for the feed as well as the order can be changed in the utils file which can be found at `{{project_slug}}/utils/templatetags/{{project_slug}}_utils.py`.

Latest News

The template tag `{% latest_news %}` is a news feed of the most recent post. The count for this feed can be changed in the utils file which can be found at `{{project_slug}}/utils/templatetags/{{project_slug}}_utils.py`.

CHAPTER 7

App Modules

We provide a comprehensive suite of apps to use for building your site. We take a modular approach so that can pick and choose the functionality you want to include.

Blog

The Blog Page module allows for your daily blog post, articles or even news posts.

- Now supports Facebook sized feed images by default 1200x630
- Blog page redesigned for a simpler look and feel

Contact

This is your Contact Page with included contact form that supports unlimited fields.

- New side bar supporting google map images
- Contact card flexibility

Document Gallery

This module allows to organize documents into “folders” based on common tags. Organize documents in folders using wagtail tags

- Go to Documents section in Wagtail Admin.
- Add add a common tag name to all documents that you want to appear in the folder.
- Now create a Document Index page which displays all your Document folders.
- Create a Document Folder Child Page and enter the tags for the document that you want to appear in the folder.

- The Folder is now created with all the Documents matching the tags you want.

Events

The Event Page module allows for adding of your upcoming events with fields for date, location, time, cost and much more.

- New Event Feed design with scroll access which allows for multiple events to be presented in the feed.

Pages

This core module allows for different pages to be added including a Homepage and Standard Pages with multiple different template layouts.

- Standard Index now supports Feed Images
- Now supports template options for full page site of a standard page with a sidebar

People

The People Page module allows for biography pages for persons in your organization or team.

- New design to the people index with callout

Photo Gallery

The Photo Gallery module allows you to easily create Photo Galleries for your site Using the built in tagging functionality. Our photo gallery now uses Lightbox2 as clearing box has been removed from foundation. To create Photo Galleries simply:

- Go to Images section of the Wagtail Admin and click on Add an Image.
- Drag and drop images you want in your gallery and add common tag name to all the uploaded Images. You can also add same tag name to any existing images that you want to include in the gallery.
- Next create a Gallery Index Page which displays all your galleries.
- Add a Gallery Child Page and enter the tags of the Images that you want to appear in the Gallery. You can also choose a feed image so it can appear in Gallery index page.
- Your Gallery is now created with all images you want.

Products

The Product Page module allows for a store like look allowing you to display products with prices and also related products.

- New design to the product index with callout
- Redesign of the Product Page with carousel access to products

Utils

New updated share buttons with whatsapp added for mobile

CHAPTER 8

Grunt Tasks

Make sure that nodejs is installed. Then in the project root run

```
$ npm install
or
$ make node_modules
```

Image Compression

To compress images used in project's made with this cookiecutter run

```
$ grunt imagemin
```

You can also use make to run the above task

```
$ make compress_images
```

Using Browser Sync for browser testing

To use `browser-sync` for Time-saving synchronised browser testing:

```
$ grunt browser-sync
```

CHAPTER 9

Deploying to a VPS

Create a Virtual Server

Create a droplet at Digital Ocean selecting your size server, server name, server location and adding your ssh keys. Once you have create a droplet, deploying your site is really simple.

If you are using digital ocean run

```
make pre_task
```

Creating a Deployment User

Next we want to create a deployment user. To create the deploy user for the production

```
make deploy_user
```

Next copy your `id_rsa.pub` to the `ansible/{{cookiecutter.project_slug}}_keystore/` folder and change the file name to `authorized_keys`

In addition, to creating the deploy user, this make command will download the RSA `ssh` public key for the deployment user into your `ansible/{{cookiecutter.project_slug}}_keystore/` directory. Add this key as a deployment key on sites like github.com or bitbucket.org

Note: If your repository is private. Make sure to add `ssh` key to the repository level deploy keys. You can add deploy keys to the repository from the repository's settings tab.

Provisioning your Server and Deploying your Site

Now we want to populate your site with all the project's requirements

```
make provision
```

You may also choose to run the individual tasks using tag by making use of the ansible tags.

Install and configure various Ubuntu packages

```
ansible-playbook -i ansible/production ansible/provision.yml --tags packages
```

Add basic security (UFW and Fail2Ban):

```
ansible-playbook -i ansible/production ansible/provision.yml --tags secure
```

Install and configure PostgreSQL, set up the project's database

```
ansible-playbook -i ansible/production ansible/provision.yml --tags database
```

Install and Configure LetsEncrypyt and Nginx

```
ansible-playbook -i ansible/production ansible/provision.yml --tags nginx_le
```

Install VCS, Venv, Bower, Redis, Django, Load Initial Data, Gunicorn Celery

```
ansible-playbook -i ansible/production ansible/provision.yml --tags webapp
cp env.example ansible/wagtail_project_keystore/env.production
```

Once your site is up and running. You can push the changes to the live site using

```
make deploy
```

Populate .env With Your Environment Variables

Some of these services rely on environment variables set by you. There is an env.example file in the root directory of this project as a starting point. Add your own variables to the file, then move it to the `ansible/{{cookiecutter.project_slug}}_keystore/` folder and change the file name to `env.production`. After you have change the file name set the `DJANGO_DEBUG` to `off`.

CHAPTER 10

Deployment to PythonAnywhere

Overview

Full instructions follow, but here's a high-level view.

First time config:

1. Pull your code down to PythonAnywhere using a *Bash console* and setup a virtualenv
2. Set your config variables in the *postactivate* script
3. Run the *manage.py* `migrate` and `collectstatic` commands
4. Add an entry to the PythonAnywhere *Web tab*
5. Set your config variables in the PythonAnywhere *WSGI config file*

Once you've been through this one-off config, future deployments are much simpler: just `git pull` and then hit the "Reload" button :)

Getting your code and dependencies installed on PythonAnywhere

Make sure your project is fully commited and pushed up to Bitbucket or Github or wherever it may be. Then, log into your PythonAnywhere account, open up a **Bash** console, clone your repo, and create a virtualenv:

```
git clone <my-repo-url>  # you can also use hg
cd my-project-name
mkvirtualenv --python=/usr/bin/python3.5 my-project-name # or python2.7, etc
pip install -r requirements/production.txt  # may take a few minutes
```

Setting environment variables in the console

Generate a secret key for yourself, eg like this:

```
python -c 'import random; print("".join(random.SystemRandom().choice(
↪"abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*(-_=+)") for _ in range(50)))'
```

Make a note of it, since we'll need it here in the console and later on in the web app config tab.

Set environment variables via the virtualenv "postactivate" script (this will set them every time you use the virtualenv in a console):

```
vi $VIRTUAL_ENV/bin/postactivate
```

TIP: *If you don't like vi, you can also edit this file via the PythonAnywhere "Files" menu; look in the ".virtualenvs" folder.*

Add these exports

```
export DJANGO_SETTINGS_MODULE='<project_slug>.settings.production'
export PROD_DATABASE_URL='<see below>'
export DJANGO_SECRET_KEY='<secret key goes here>'
export DJANGO_ALLOWED_HOST_NAME='<www.your-domain.com>'
export EMAIL_HOST='email_host'
export EMAIL_FROM='support@host.com'
export EMAIL_USER='email_user'
export EMAIL_PASSWD='email_passwd'
```

Database setup:

Go to the PythonAnywhere **Databases tab** and configure your database.

- For Postgres, setup your superuser password, then open a Postgres console and run a *CREATE DATABASE my-db-name*. You should probably also set up a specific role and permissions for your app, rather than using the superuser credentials. Make a note of the address and port of your postgres server.
- For MySQL, set the password and create a database. More info here: https://help.pythonanywhere.com/pages/UsingMySQL
- You can also use sqlite if you like! Not recommended for anything beyond toy projects though.

Now go back to the *postactivate* script and set the `DATABASE_URL` environment variable:

```
export DATABASE_URL='postgres://<postgres-username>:<postgres-password>@<postgres-
↪address>:<postgres-port>/<database-name>'
# or
export DATABASE_URL='mysql://<pythonanywhere-username>:<mysql-password>@<mysql-
↪address>/<database-name>'
# or
export DATABASE_URL='sqlite:////home/yourusername/path/to/db.sqlite'
```

If you're using MySQL, you may need to run `pip install mysqlclient`, and maybe add `mysqlclient` to *requirements/production.txt* too.

Now run the migration, and collectstatic:

```
source $VIRTUAL_ENV/bin/postactivate
python manage.py migrate
python manage.py collectstatic
# and, optionally
python manage.py createsuperuser
```

Configure the PythonAnywhere Web Tab

Go to the PythonAnywhere **Web tab**, hit **Add new web app**, and choose **Manual Config**, and then the version of Python you used for your virtualenv.

NOTE: *If you're using a custom domain (not on *.pythonanywhere.com), then you'll need to set up a CNAME with your domain registrar.*

When you're redirected back to the web app config screen, set the **path to your virtualenv**. If you used virtualenvwrapper as above, you can just enter its name.

Click through to the **WSGI configuration file** link (near the top) and edit the wsgi file. Make it look something like this, repeating the environment variables you used earlier:

```
import os
import sys
path = '/home/<your-username>/<your-project-directory>'
if path not in sys.path:
    sys.path.append(path)

os.environ['DJANGO_SETTINGS_MODULE'] = '<project_slug>.settings.production'
os.environ['PROD_DATABASE_URL'] = '<as above>'
os.environ['DJANGO_SECRET_KEY'] = '<as above>'
os.environ['DJANGO_ALLOWED_HOST_NAME'] = '<as above>'
os.environ['EMAIL_HOST'] = '<as above>'
os.environ['EMAIL_FROM'] = '<as above>'
os.environ['EMAIL_USER'] = '<as above>'
os.environ['EMAIL_PASSWD'] = '<as above>'

from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

Back on the Web tab, hit **Reload**, and your app should be live!

NOTE: *you may see security warnings until you set up your SSL certificates. If you want to supress them temporarily, set DJANGO_SECURE_SSL_REDIRECT to blank. Follow the instructions here to get SSL set up: https://help.pythonanywhere.com/pages/SSLOwnDomains/*

Static files

Essentially you'll need an entry to match your `STATIC_URL` and `STATIC_ROOT` settings. There's more info here: https://help.pythonanywhere.com/pages/DjangoStaticFiles

Future deployments

For subsequent deployments, the procedure is much simpler. In a Bash console:

```
workon my-virtualenv-name
cd project-directory
git pull
python manage.py migrate
python manage.py collectstatic
```

And then go to the Web tab and hit **Reload**

TIP: *if you're really keen, you can set up git-push based deployments: https://blog.pythonanywhere.com/87/*

Inspired from Pythonanywhere deployment docs at https://cookiecutter-django.readthedocs.io/en/latest/deployment-on-pythonanywhere.html

CHAPTER 11

Deployment to Aldryn Cloud

Wagtail is now supported on Aldryn Cloud. The following steps will guide you to deploy Wagtail Cookiecutter Foundation projects on Aldryn Cloud.

Steps to Deploy

1. Create a Wagtail project (like `wagtail-aldryn`) on Aldryn using the guide - Get Started with Wagtail on Aldryn
2. Create a local development environment of project generated.
3. Suppose the project you want to deploy (generated with Wagtail Cookiecutter Foundation) is named as `wagtail-demo`.
4. Now we can copy the apps directories from `wagtail-demo` project directory to `wagtail-aldryn` project directory. You can move apps like `pages`, `blog`, `utils` etc. You are free to move any app but remember all static assets are present in `pages` app.
5. Copy the `media` directory to `wagtail-aldryn` project directory.
6. Open `requirements.in` file from `wagtail-aldryn` project. Add the following requirements:

```
wagtailfontawesome==1.0.5
celery==3.1.23
django-cachalot==1.2.1
django-compressor==2.1
django-environ==0.4.0
django-foundation-formtags==0.0.6
django-wagtail-feeds==0.0.3
django-libsass==0.7
django-redis==4.4.4
elasticsearch==2.4.0
libsass==0.11.1
```

For updated list, copy the requirements from your `wagtail-demo` project.

7. Open `settings.py` file from `wagtail-aldryn` project. Add the following:

```
INSTALLED_APPS.extend([
    # add your project specific apps here
    'compressor',
    'foundation_formtags',
    'wagtail_feeds',

    'utils',
    'pages',
    'blog',
    'events',
    'contact',
    'people',
    'photo_gallery',
    'products',
    'documents_gallery',
])

COMPRESS_PRECOMPILERS = (
    ('text/x-scss', 'django_libsass.SassCompiler'),
)
```

8. Importing Database from `wagtail-demo` project.

- Create db dump of `wagtail-demo` project.
- Wagtail Cookiecutter Foundation and Aldryn projects use PostgreSQL as a database management system (DBMS). To transfer your existing Postgres data dumps into the `wagtail-aldryn` project, the commands to do so could look like following example:

```
docker exec <container_id> dropdb -U postgres db --if-exists

docker exec <container_id> createdb -U postgres db

docker exec <container_id> psql -U postgres --dbname=db -c "CREATE EXTENSION IF␣
↪NOT EXISTS hstore"

docker run --rm -v /path/to/dump:/app/tmp/db_dump --link <container_id>:postgres␣
↪postgres:9.4 /bin/bash -c pg_restore -h postgres -U postgres -F /app/tmp/db_
↪dump --dbname=db -n public --no-owner --exit-on-error
```

Note: `<container_id>` is usually something like: `projectslug_db` (if you're unsure, open the docker-compose.yaml file and check.

9. Test locally if everything works.
10. Push changes to Server

To push code changes to the test server, run for example:

```
git add .
git commit -m "Your commit message"
git push
```

To push DB changes, run:

```
aldryn project push db
```

To push media file changes, run:

```
aldryn project push media
```

To deploy the test server, run:

```
aldryn project deploy
```

Useful links:

- https://support.divio.com/hc/en-us/articles/209053809-Get-Started-with-Wagtail-on-Aldryn
- https://support.divio.com/hc/en-us/articles/208393155-Moving-your-Django-projects-to-Aldryn
- https://support.divio.com/hc/en-us/articles/208243415

CHAPTER 12

Project Settings

This project relies extensively on environment settings which **will not work with Apache/mod_wsgi setups**. It has been deployed successfully with both Gunicorn/Nginx and even uWSGI/Nginx.

For configuration purposes, the following table maps environment variables to their Django setting:

Environment Variable	Django Setting	Development Default	Production Default
DJANGO_ADMIN_URL	n/a	r'^admin/'	raises error
DJANGO_CACHES	CACHES (default)	locmem	redis
DJANGO_DATABASES	DATABASES (default)	See code	See code
DJANGO_DEBUG	DEBUG	True	False
DJANGO_SECRET_KEY	SECRET_KEY	CHANGEME!!!	raises error
DJANGO_SECURE_BROWSER_XSS_FILTER	SECURE_BROWSER_XSS_FILTER	n/a	True
DJANGO_SECURE_SSL_REDIRECT	SECURE_SSL_REDIRECT	n/a	True
DJANGO_SECURE_CONTENT_TYPE_NOSNIFF	SECURE_CONTENT_TYPE_NOSNIFF	n/a	True
DJANGO_SECURE_FRAME_DENY	SECURE_FRAME_DENY	n/a	True
DJANGO_SECURE_HSTS_INCLUDE_SUBDOMAINS	SECURE_HSTS_INCLUDE_SUBDOMAINS	n/a	True
DJANGO_SESSION_COOKIE_HTTPONLY	SESSION_COOKIE_HTTPONLY	n/a	True
DJANGO_SESSION_COOKIE_SECURE	SESSION_COOKIE_SECURE	n/a	False
DJANGO_DEFAULT_FROM_EMAIL	DEFAULT_FROM_EMAIL	n/a	"your_project_name <noreply@your_domain_name>"
DJANGO_SERVER_EMAIL	SERVER_EMAIL	n/a	"your_project_name <noreply@your_domain_name>"
DJANGO_EMAIL_SUBJECT_PREFIX	EMAIL_SUBJECT_PREFIX	n/a	"[your_project_name] "
DJANGO_ALLOWED_HOSTS	ALLOWED_HOSTS	['*']	['your_domain_name']

The following table lists settings and their defaults for third-party applications, which may or may not be part of your project:

Environment Variable	Django Setting	Development Default	Production Default
DJANGO_AWS_ACCESS_KEY_ID	AWS_ACCESS_KEY_ID	n/a	raises error
DJANGO_AWS_SECRET_ACCESS_KEY	AWS_SECRET_ACCESS_KEY	n/a	raises error
DJANGO_AWS_STORAGE_BUCKET_NAME	AWS_STORAGE_BUCKET_NAME	n/a	raises error
DJANGO_SENTRY_DSN	SENTRY_DSN	n/a	raises error
DJANGO_SENTRY_CLIENT	SENTRY_CLIENT	n/a	raven.contrib.django.raven_compat.DjangoClient
DJANGO_SENTRY_LOG_LEVEL	SENTRY_LOG_LEVEL	n/a	logging.INFO
DJANGO_MAILGUN_API_KEY	MAILGUN_ACCESS_KEY	n/a	raises error
DJANGO_MAILGUN_SERVER_NAME	MAILGUN_SERVER_NAME	n/a	raises error
NEW_RELIC_APP_NAME	NEW_RELIC_APP_NAME	n/a	raises error
NEW_RELIC_LICENSE_KEY	NEW_RELIC_LICENSE_KEY	n/a	raises error
DJANGO_OPBEAT_APP_ID	OPBEAT['APP_ID']	n/a	raises error
DJANGO_OPBEAT_SECRET_TOKEN	OPBEAT['SECRET_TOKEN']	n/a	raises error
DJANGO_OPBEAT_ORGANIZATION_ID	OPBEAT['ORGANIZATION_ID']	n/a	raises error

Other Environment Settings

DJANGO_ACCOUNT_ALLOW_REGISTRATION (=True) Allow enable or disable user registration through *django-allauth* without disabling other characteristics like authentication and account management. (Django Setting: ACCOUNT_ALLOW_REGISTRATION)

CHAPTER 13

Using Ansible

This cookiecutter also comes with a suite of Ansible play books and roles for provisioning your servers and deploying the site. We also support the creation of a Vagrant based staging server to "stage" your site locally and allow you to tweak and experiment with different deployment configurations. By default these play books assume that all your application components `django`, `PostgreSQL`, `redis` and so on are deployed to a single server. However, we can easily change the Vagrant staging configuration to simulate more complex deployments including using a separate Database server, multiple upstream `wsgi` servers and so on.

Ansible Keystore

In our project layout, the `keystore` folder located at `ansible/cookiecutter.project_slug.keystore` is utilized for storing authorized keys, deployment key and the production environment settings. The various provisioning and deployment playbooks expect these files to be in the keystore directory and would fail if they were not present. In case you are wondering, the included `.gitignore` and `.hgignore` should ensure that that no file in this directory can accidentally be added to your VCS. However, it is important to verify that this is indeed the case. The files are as follows:

- *Authorized Keys (authorized_keys)* - public key of the developers for e.g. `id_rsa.pub`. You can concatenate keys for one or more developers and name as `authorized_keys`
- *Deployment Key (host_name-deploy_user-deploy_keys.pub)* - This is the Public key of the Deployment user generated running the command `make deploy`
- *Production Environment Settings (production.env)* - The Django production enviroment settings used in the project.

Ansible Playbooks

To provision you servers run

```
cd ansible
#list the available tags
ansible-playbook -i staging provision.yml --list-tags

#Run all the plays

ansible-playbook -i staging provision.yml

#Install and configure various Ubuntu packages

ansible-playbook -i ansible/production ansible/provision.yml --tags packages

#Add basic security (UFW and Fail2Ban)

ansible-playbook -i ansible/production ansible/provision.yml --tags secure

#Install and configure PostgreSQL, set up the project's database

ansible-playbook -i ansible/production ansible/provision.yml --tags database

#Install and Configure LetsEncrypyt and Nginx

ansible-playbook -i ansible/production ansible/provision.yml --tags nginx_le

#Install VCS, Venv, Bower, Redis, Django, Load Initial Data, Gunicorn Celery

ansible-playbook -i ansible/production ansible/provision.yml --tags webapp
```

To deploy changes to production

```
make deploy
```

To make changes to your project settings edit env.production and also settings/production. Then run:

```
ansible-playbook -i ansible/production ansible/update_env.yml
```

Ansible Variables

Group Variables

Variable	Default
project_slug	ssh://hg@bitbucket.org/username/wagtail_project
project_title	Wagtail Project
project_name	Wagtail Project
virtualenvs_dir	/home/django/virtualenvs/
sites_dir	/usr/local/sites
nginx_root_dir	/etc/nginx/sites-available
gunicorn	127.0.0.1:2015
deploy_user	django
keystore_path	project_slug/ansible/project_slug_keystore
vcs	*hg* - mecurial *git* - git

Host Variables

Variable	Default
use_ssl	True if this is going to be a production deployment
DJANGO_SETTINGS_MODULE	wagtail_project.settings.production
HOST_NAME	`server_name`
DB_USER	django
DJANGO_ADMIN:	django_admim_user_name
DJANGO_ADMIN:	your_django_admin_password
DB_PASSWD	The database password you must set a value for this
DB_HOST	localhost
DB_NAME	cookiecutter.project_slug_db
EMAIL_HOST	The SMTP email host name e.g. `smtp.mandrillapp.com`
EMAIL_FROM	support@chrisdev.com
EMAIL_USER	The email user
EMAIL_PASSWD	The email password

CHAPTER 14

Make Commands

The easiest way to get started is to use the built in `make` commands. Your project contains a Makefile that allows you to setup your development environment with a single command. This command will create your project's virtual environment, install all pip dependencies, create the development database, run migrations and load initial data to database, install front-end dependencies and finally start the development server for you.

To do this run

```
make develop_env
```

You can access your site at `http://localhost:8000`. The Admin back-end is available at `http://localhost:8000/admin/`.The default Admin username is *admin* and The default Admin password is *admin123*.

Make command line

Create the virtualenv for the project

```
make virtualenv
```

Install the requirements to the virtualenv

```
make requirements
```

Create a PostgreSQL database for the project. It will have the same name as the project

```
make db
```

Run the migrations

```
make migrate
```

Populate the site with initial page structure

```
make initial_data
```

Copy the media(images and documents) to project root

```
make copy_media
```

Install all front-end dependencies with bower

```
make bower
```

Start the standard Django dev server

```
make runserver
```

Start Server with livereload functionality

```
make livereload
```

Run your unit tests

```
make test
```

Run your functional tests

```
make func_test
```

Install Node modules:

```
make node_modules
```

Minify Images used in site

```
make compress_images
```

Generate a static site from the project

```
make static_site
```

For Staging

For staging run / a Vagrant based staging server

```
make deploy_user DEPLOY_ENV=staging
```

This will create the deploy user for the production server. For staging run

```
make deploy_user DEPLOY_ENV=staging
```

If you want to use the Vagrant based staging server first ensure that the Vagrant VM is running

```
cd /my_project/ansible
vagrant up
```

Then create the deployment user return to the project root and run:

```
make deploy_user DEPLOY_ENV=vagrant
```

When prompted for the password, enter “vagrant”.

If you get the following error

```
fatal: [staging.example.org] => {'msg': 'FAILED: Authentication failed.', 'failed':␣
↪True}``
```

For the staging server run

```
make provision DEPLOY_ENV=staging
```

To provision the Vagrant based staging server run

```
make provision DEPLOY_ENV=vagrant
```

CHAPTER 15

Vagrant for Development

Alternatively you may prefer to use Vagrant to run your project locally in its own virtual machine. This will allow you to use PostgreSQL, Elasticsearch Redis etc. in development without having to install them on your host machine. To install Vagrant, see: Installing Vagrant

To setup the Vagrant box, run the following commands

```
vagrant up # This may take some time on first run
vagrant ssh
# within the ssh session
bower install
dj createsuperuser
djrun
```

If you now visit http://localhost:8000 you should see the default wagtail foundation site

You can browse the Wagtail admin interface at: http://localhost:8000/admin

You can read more about how Vagrant works at: https://docs.vagrantup.com/v2/

Vagrant based Staging Server

Start by changing to the `ansible` directory and bringing up vagrant based the staging server.

```
cd /my_project/ansible
vagrant up
```

Because of the way Vagrant is setup we need to run a special play book to copy your `ssh` public key (`id_rsa.pub`) to the root account on the Vagrant staging machine i.e. to `authorized_keys`.

```
ansible-playbook -c paramiko -i staging vagrant_staging_setup.yml --ask-pass --sudo -
↪u vagrant
```

When prompted for the password, enter "vagrant"

If you get the following error

```
fatal: [staging.example.org] => {'msg': 'FAILED: Authentication failed.', 'failed':␣
↪True}``
```

The you may have to remove the entry (IP Address 192.168.33.10) for the staging server from your `~/.ssh/known_hosts` file.

If you are using Vagrant staging you also need to make an entry into your `/etc/hosts` file for example.

CHAPTER 16

Contributing

Contributions are always welcome to improve this project. If you think you've found a bug or are interested in contributing fork this project and send the pull request. After review, your pull request will be merged. We are always happy to receive pull requests. If you identify any issue, please raise it in the issues section.

CHAPTER 17

Authors

Development Leads

- Christopher Martin Clarke (@chrisdev)

Core Committers

- Lendl Smith (@ilendl2)
- Parbhat Puri (@parbhat)

Contributors

- Fygul Hether (@fygul)
- Matt Westcott (@gasman)

Index

P