django-fabric Documentation

Release 1.5.0

Rolf Erik Lekang

July 18, 2015

Contents

CHAPTER 1

Quickstart

django-fabric is written to make writing fabfiles for django projects easier and faster. It contains the basic stuff one would expect from a django setup with git and virtualenv. The code expects the project to have a certain structure as seen below. It is possible to customize the activation of the virtualenvironment. .. code-block:

```
project-dir/
  venv/ # virtualenv
  project-package/
  manage.py
  fabfile.py
```

1.1 Installation

Run `pip install django-fabric`

1.2 Usage

There is two options to get get a basic setup, both will make you able to run `fab deploy:prod` and `fab test`.

1.2.1 Init script

There is a init script that will guide you through the generation of a basic fabfile that utilises django-fabric. Run it with the command .. code-block:

```
django-fabric-init
```

1.2.2 Basic manual setup

Create a `fabfile.py` in your project directory. You can see example of a fabfile below. If you run into problems with settings where fabric cannot locate settings add `sys.path.append(os.path.dirname(__file__))` to your fabfile.

Here is an example of an fabfile .. code-block:

```
from fabric.decorators import task
from fabric.state import env
from django_fabric import App

env.user = 'web'
env.hosts = ['server1.example.com']

site = App(
    project_paths={
        'prod': '/var/www/example_site',
    },
    urls={
        'prod': 'http://example.com'
    },
    restart_command={
        'prod': 'restart prod'
    },
    project_package='example',
    test_settings='example.settings.test',
)

deploy = task(site.deploy)
test = task(site.test)
```

CHAPTER 2

Testing

We all get a little nervous the first time we run a deploy script on production. Well, don't do it without proper testing first. django-fabric is used as deployment tool in several organisations. However, you should feel a lot more safe that you configured it correctly if you test it first.

One way to test it is to run a deployment of a staging environment before you deploy to your production environment.

2.1 The testing mixin

There is one way to do a no-operation run of the deployment, wich mean that every command that will be ran on your servers will be printed instead. This will give you a way to visually confirm the shell commands before running them on a server. The example below shows how to use the mixin.

```
from fabric.contrib import django

from django_fabric import App
from django_fabric.test_helpers import TestMixin

class TestApp(TestMixin, App):
    project_package = 'package'
    project_paths = {
        'prod': 'path-to-prod'
    }
    restart_command = {
        'prod': 'restart prod'
    }
```

CHAPTER 3

Notifications

It is always great to notify your team that you are deploying. django-fabric makes it easy to do that automatically.

3.1 Built in notification mixins

There are some built in mixins. To use them add them to your class in your fabfile and make sure you add the attributes necessary. They should have defaults for values that are not used to authenticate you with the given service.

class `django_fabric.notifications.`**IrcNotifyMixin**
A mixin that notifies given channels on irc.

SERVER
Default: 'irc.freenode.org'
The irc server you want to connect to.

PORT
Default: 6667
The port of the irc server.

NICK
Default: 'django-fabric'
The nick that should appear on irc when the notification is sent.

ROOMS
Default: []
List of rooms to notify, should be a list of strings.

TIMEOUT
Default: 25
The time in seconds before the irc connection times out.

class `django_fabric.notifications.`**SlackNotifyMixin**
A mixin that notifies a channel on the Slack. Requires to set the attribute URL.

CHANNEL
Default: '#general'
The channel to post the notification in.

NICK
Default: 'django-fabric'
The nick that should appear in Slack when the notification is sent.

URL
The Slack POST URL. Can be found at slack.com/services/new/incoming-webhook.

class `django_fabric.notifications.`**HipChatNotifyMixin**

A mixin that notifies a room on HipChat. Requires to set the attribute ROOM and HIPCHAT_TOKEN.

ROOM

The room to post the notification in.

NOTIFY
Default: False

Whether or not this message should trigger a notification for people in the room (change the tab color, play a sound, etc).

COLOR
Default: 'yellow'

Background color for message. Valid values: yellow, red, green, purple, gray, random

3.2 Build your own notification mixin

If we do not support your chat service, bot or whatever you want to notify it should not be a problem. It is pretty easy to create your own notification mixin. Just create a class that inherit from the `Notifier` class and overwrite the methods you need to customize. Remember you must at least override send_notification. If you think your notification mixin can be useful for others a pull-request is appreciated.

class `django_fabric.notifications.`**Notifier**

notification_message_context(self, instance):

Provides the context used in `pre_deploy_notify()` and `post_deploy_notify()`.

pre_deploy_notify(self, instance):

The method that sends notification before deployment. This should generate the message and call `send_notification`.

post_deploy_notify(self, instance):

The method that sends notification after deployment. This should generate the message and call `send_notification`.

send_notification(self, message):

This method actually sends the notification. The logic that talks to the service should be put here. This method needs to be implemented in the subclass or it will raise a NotImplementedError.

CHAPTER 4

Advanced usage

To be able to use different mixins or override som methods in the App class it is necessary to subclass it. If you use this approach in your fabfile it is possible to move your values out of the init call as seen in the example below, if you want to.:

```
from fabric.decorators import task
from fabric.state import env
from django_fabric import App

env.user = 'web'
env.hosts = ['server1.example.com']

class Site(App):
    project_package = 'package'
    project_paths = {
        'prod': 'path-to-prod'
    }
    restart_command = {
        'prod': 'restart prod'
    }

site = Site()

deploy = task(site.deploy)
test = task(site.test)
```

4.1 Need to use custom commands on the server?

Need to `su` to a specific user or something similar. No problem! Just override the method `App.run(command)`, but there are a few things to remember.

- Add `with quiet():` context manager around your code if you want to hide the output from fabric and only show the output from django-fabric.
- Return the fabric run command. This is used to determine the output of the command several places.

CHAPTER 5

Quickstart

django-fabric is written to make writing fabfiles for django projects easier and faster. It contains the basic stuff one would expect from a django setup with git and virtualenv. The code expects the project to have a certain structure as seen below. It is possible to customize the activation of the virtualenvironment. .. code-block:

```
project-dir/
  venv/ # virtualenv
  project-package/
  manage.py
  fabfile.py
```

5.1 Installation

Run `pip install django-fabric`

5.2 Usage

There is two options to get get a basic setup, both will make you able to run `fab deploy:prod` and `fab test`.

5.2.1 Init script

There is a init script that will guide you through the generation of a basic fabfile that utilises django-fabric. Run it with the command .. code-block:

```
django-fabric-init
```

5.2.2 Basic manual setup

Create a `fabfile.py` in your project directory. You can see example of a fabfile below. If you run into problems with settings where fabric cannot locate settings add `sys.path.append(os.path.dirname(__file__))` to your fabfile.

Here is an example of an fabfile .. code-block:

```
from fabric.decorators import task
from fabric.state import env
from django_fabric import App

env.user = 'web'
env.hosts = ['server1.example.com']

site = App(
    project_paths={
        'prod': '/var/www/example_site',
    },
    urls={
        'prod': 'http://example.com'
    },
    restart_command={
        'prod': 'restart prod'
    },
    project_package='example',
    test_settings='example.settings.test',
)

deploy = task(site.deploy)
test = task(site.test)
```

C

H

I

N

P

R

S

T

U