fab$_supportDocumentation$

Release 0.2.6

Humphrey Drummond

Nov 01, 2019

Contents

1 fab_support **3**
1.1 Stages . . . 3

2 Django configuration **5**
2.1 Features . . . 5
2.2 Levels of fabfile in this module . . . 5
2.3 Credits . . . 6

3 Installation **7**
3.1 Stable release . . . 7
3.2 From sources . . . 7

4 Usage **9**

5 General variables env['stages'] **11**

6 Environment variables env['stages']['stage_x']['ENV'] **13**
6.1 DJANGO_ALLOWED_HOSTS . . . 14

7 Contributing **15**
7.1 Types of Contributions . . . 15
7.2 Get Started! . . . 16
7.3 Pull Request Guidelines . . . 17
7.4 Tips . . . 17

8 Credits **19**
8.1 Development Lead . . . 19
8.2 Contributors . . . 19

9 History **21**
9.1 0.2.1 (2018-05-11) . . . 21
9.2 0.2.0 (2018-04-20) . . . 21
9.3 0.1.0 (2018-02-04) . . . 21

10 Indices and tables **23**

Contents:

CHAPTER 1

fab_support

Code to implement staging in Fabric and recipes for using that staging for pelican deployments and Django to Heroku. It supports a local .env file importing for storing secrets that you don't want to store in git.

1.1 Stages

Stages are the different stages of development of an application. So they might go from:

test -> uat -> production -> old production

Fig. 1: Different stages of a single project

I have create a fab-support.py which does the heavy lifting of creating each environment. The aim is that this should be hardly any more than the use of fabric and much simpler than the use of a full featured build Salt or Ansible. This is really only if you fit one of the use cases. Like Ansible this is a simple single master deployment system.

Suitable use cases:

- **Deployment of Pelican static website**
  - Deployment to local file system for use with a file server
  - Deployment to local for a file based browser
  - Deployment to S3
- **Simple Django to Heroku where you have at a minimum two stages eg UAT and Production.**
  - Copes with Postgres database
  - Static data in AWS

In the root fabfile create a dictionary like this which documents how to deploy each stage:

```
from fabric.api import env

# Definition of different environments to deploy to
env['stages'] = {
    'localsite': {
        'comment': 'stage: For serving locally on this computer via mongoose. ',
        'config_file': 'local_conf.py',
        'destination': 'C:/Sites/local.drummond.info',
        'copy_method': copy_file,
        'SITEURL': 'http://localhost:8042',
    },
    'production': {
        'comment': 'stage: For serving on local file server',
        'destination': '//10.0.0.1/web/www.drummond.info',
        'config_file': 'local_conf.py',
        'copy_method': copy_file,
        'SITEURL': 'http://www.drummond.info',
},
}
```

Then the deployment by Pelican is pretty standardised eg build deploy and you have commands such as:

fab localsite deploy

I think it was inspired by BreytenErnsting. This is then reiplmeneted using the standard env environment and support in Fabric.

- Free software: MIT license
- Documentation: https://fab-support.readthedocs.io.

CHAPTER 2

Django configuration

The Django configuration includes the following features:

- deployment to Heroku
- Celery support with aqmp
- Log trapping support with Papertrail

2.1 Features

Runs on Windows. If it is getting to complex then it should probably be ported to Ansible or Salt.

2.2 Levels of fabfile in this module

In this module I use three levels of fabfile.py:

- At the project root
- at the /tests root
- at a test/demo level

This can get confusing, however they operate at different levels. The project is about project operations eg releasing to fab_support to pypi.

The tests level is then about managing the tests. Some of these tests use fab support and a fabfile.py which gives you the third level of nesting.

2.2.1 Project level fabfile

This is used to do work on the distribution:

- Make deocumentation
- build wheels
- deploy wheels to the package manager

2.2.2 At the tests level

This is used to run local commands. Often the commands will be run from the test fab file level and then *lcd* to the demo level.

2.2.3 At the tests/demo level

This is a model fabric file- however it is not like a normal one in that fab_support is not installed in the environment and in fact is located at *../../fab_support*.

2.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template. Thanks Audrey

CHAPTER 3

Installation

3.1 Stable release

To install fab_support, run this command in your terminal:

```
$ pip install fab_support
```

This is the preferred method to install fab_support, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

3.2 From sources

The sources for fab_support can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/drummonds/fab_support
```

Or download the tarball:

```
$ curl  -OL https://github.com/drummonds/fab_support/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 4

Usage

To use fab_support in a project add:

```
import fab_support
```

You will then need a fabfile.py like this:

```
from fabric.api import env
import os
import time


import fab_support


# Operating System Environment variables have precedence over variables defined in␣
↪the .env file,
# that is to say variables from the .env files will only be used if not defined
# as environment variables.
# Load operating system environment variables and then prepare to use them
os_env = environ.Env()
env_file = '.env'
print('Loading : {}  See {} for more info'.format(env_file, __name__))
os_env.read_env(env_file)


# Definition of different environments to deploy to
env['stages'] = {
    'local': {
        'comment': 'FAC Galleria local version',
        'ENV': {
            'DJANGO_SECRET_KEY' : os.getenv('DJANGO_SECRET_KEY_LOCAL'),
        },
        'GIT_BRANCH': 'test'
    },
```

(continues on next page)

(continued from previous page)

```
    'uat': {
        'comment' : 'www.drummonds.net UAT',
        'ENV': {
            'DJANGO_SECRET_KEY' : os.getenv('DJANGO_SECRET_KEY_LOCAL'), # same as␣
↪local
        },
        'DJANGO_SETTINGS_MODULE' : 'drummonds_net.settings.production',  # UAT same␣
↪as production
        'HEROKU_APP_NAME' : 'drummonds-uat',
        'HEROKU_PROD_APP_NAME': 'drummonds-prod',
        'HEROKU_POSTGRES_TYPE' : 'hobby-basic',  # Need more than 10,000 rows allows␣
↪to 10M rows but costs $9 a month
        'PRODUCTION_URL' : 'uat.drummonds.net',
        'DJANGO_SECRET_KEY' : os.getenv('DJANGO_SECRET_KEY'),
        'DJANGO_ALLOWED_HOSTS' : '.herokuapp.com,.drummonds.net',
        'DJANGO_SENTRY_DSN' : 'https://b1a_very:_secrete6b34d18@sentry.io/1170320',
        'GIT_BRANCH' : 'uat'
    },
    'prod': {
        'comment' : 'www.drummonds.net production',
        'ENV': {
            'DJANGO_SECRET_KEY' : os.getenv('DJANGO_SECRET_KEY_PROD'),
        },
        'DJANGO_SETTINGS_MODULE' : 'drummonds_net.settings.production',  # UAT same␣
↪as production
        'HEROKU_APP_NAME' : 'drummonds-prod',
        'HEROKU_POSTGRES_TYPE' : 'hobby-basic',  # Need more than 10,000 rows allows␣
↪to 10M rows but costs $9 a month
        'PRODUCTION_URL' : 'www.drummonds.net',
        'DJANGO_SECRET_KEY' : os.getenv('DJANGO_SECRET_KEY'),
        'DJANGO_ALLOWED_HOSTS' : '.herokuapp.com,.drummonds.net',
        'DJANGO_SENTRY_DSN' : 'https://b1a_very:_secrete6b34d18@sentry.io@sentry.io/
↪1125110',
        'GIT_BRANCH' : 'master'
    },
}
```

CHAPTER 5

General variables env[‘stages’]

As shown above this is a list of stages. Each stage is a dictionary which has general variables and also an ENV dictionary which has all the environment variables that are to be passed through to the final run time environment.

The documentation breaks down the general definitions here (those that have a meaning in fab_support) and any environment variables that have a special meaning in the next section.

Name	Default	Comments
comment		Identifies which stage this is - used internally eg fab fab_support.list_stages
GIT_BRANCH	master	Which GIT branch to use when building deployment, Required for Heroku deployement when you want to deploy a different branch than master[1].
GIT_PUSH	‘’	For specialised GIT push eg using a subtree ‘git subtree push –prefix tests/demo_django_postgres heroku master’
GIT_PUSH_DIR	‘.’	Local directory to run git push from eg ‘../..’
HEROKU_APP_NAME	fab-support-test-app	Name must start with a letter and can only contain lowercase letters, numbers, and dashes. The production name should end in *prod* for additional protection[3].
HEROKU_PROD_APP_NAME	fab-support-app-prod	Used to identify where to copy the production data from. Essential for all builds.
HEROKU_OLD_PROD_APP_NAME	fab-support-app-old_prod	Name of production (prod) after promoting uat to prod.
HEROKU_POSTGRES_TYPE	hobby-dev	free to 10K rows, hobby-basic allows to 10M rows but costs $9 a month
PRODUCTION_URL	‘’	This is where the production URL should be hosted. empty string if no remote URL[2].
USES_CELERY	False	If True then will set up on Heroku a scaling worker

[1] Heroku uses the local git repository to push from by default. So GIT_BRANCH will be the branch in the local repository

[3] This name must be globally distinct for heroku.

[2] This controls the heroku routing layer which is external to the Django routing layer. The DJANGO_ALLOWED_HOSTS is internal to the Django application and must also match the URL

CHAPTER 6

Environment variables env[‘stages’][‘stage_x’][‘ENV’]

These are the variables that are set in the .env and are carried through to the development environments. stage_x might be *uat* or *prod* etc. For heroku this will then involve the commmand line command like this *heroku config:set DJANGO_SECRET=very_secret*.

A common pattern is to use a single .env file to store all the secrets and then to use this dictionary to allocate the secrets to the same environment variable in different stages eg:

```
# Not a complete file but for illustration
env['stages'] = {
    'local': {
        'ENV': {
            'DJANGO_SECRET_KEY' : os.getenv('DJANGO_SECRET_KEY_LOCAL'),
        },
    },
    'uat': {
        'ENV': {
            'DJANGO_SECRET_KEY' : os.getenv('DJANGO_SECRET_KEY_UAT'),
        },
    },
    'prod': {
        'ENV': {
            'DJANGO_SECRET_KEY' : os.getenv('DJANGO_SECRET_KEY_PROD'),
        },
    },
}
```

If an environment variable is listed here it is because fab_support provides a default or takes some other action with it.

Name	Default	Comments
DJANGO_SETTINGS_MODULE	{{app_name}}	Two scoops config.settings.test or config.settings.production[4].
DJANGO_ALLOWED_HOSTS	Set	Will by default allow the app name setup. See *DJANGO_ALLOWED_HOSTS* for more details.
PYTHONHASHSEED	random	Heroku default

6.1 DJANGO_ALLOWED_HOSTS

This pattern was defined by Python django cookiecutter project and is the definition of a environment variable so that [*ALLOWED_HOSTS*]_ which is a standard Django setting. Then in the settings file you would have code like this:

```
ALLOWED_HOSTS = env.list('DJANGO_ALLOWED_HOSTS', default=['{{ cookiecutter.domain_
↪name }}'])
```

Defaults by application type.

Application	Default
Heroku	f'{HEROKU_APP_NAME}.herokuapp.com'

[4] Heroku needs to know what the settings module is and so the name is not passed like a simple environment variable.

CHAPTER 7

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/drummonds/fab_support/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

7.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

7.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

7.1.4 Write Documentation

fab_support could always use more documentation, whether as part of the official fab_support docs, in docstrings, or even on the web in blog posts, articles, and such.

7.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/drummonds/fab_support/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

7.2 Get Started!

Ready to contribute? Here's how to set up *fab_support* for local development.

1. Fork the *fab_support* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/fab_support.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv fab_support
   $ cd fab_support/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 fab_support tests
   $ python setup.py test or py.test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

7.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/drummonds/fab_support/pull_requests and make sure that the tests pass for all supported Python versions.

7.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_fab_support
```

CHAPTER 8

Credits

8.1 Development Lead

- Humphrey Drummond <hum3@drummonds.net>

8.2 Contributors

None yet. Why not be the first?

CHAPTER 9

History

9.1 0.2.1 (2018-05-11)

- Updating pelican comamnds to the parameter method of passing stage.

Note tests were failing to a non obvious cause. This was Heroku CLI needed updating to the latest version. I manually upgraded.

9.2 0.2.0 (2018-04-20)

- Change the way environment variables are passed through.

In version 0.1 only the following variables were considered env variables:

'DJANGO_SECRET_KEY', 'DJANGO_ADMIN_URL', 'DJANGO_AWS_ACCESS_KEY_ID', 'DJANGO_AWS_SECRET_ACCESS_KEY', 'DJANGO_AWS_STORAGE_BUCKET_NAME', 'DJANGO_MAILGUN_API_KEY', 'DJANGO_SERVER_EMAIL', 'MAILGUN_SENDER_DOMAIN', 'DJANGO_ACCOUNT_ALLOW_REGISTRATION', 'DJANGO_SENTRY_DSN', 'XERO_CONSUMER_SECRET', 'XERO_CONSUMER_KEY'

Now there is an 'ENV' list of variables that allows any variables to be passed through and also for them to renamed on the way from the file .env

9.3 0.1.0 (2018-02-04)

- First release on PyPI.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search