depl Documentation

Release 0.0.1

depl contributors

December 19, 2013

Contents

Release v0.0.1. (*Installation*)

`depl` is a pre-alpha prototype. **We want to start a discussion about how developers deploy.** Please tell us why you think depl is not the right tool for you!

Deploying stuff is hard, managing nginx and postgres painful. Why don't we solve that problem? Simple deployments should not involve tweaking configurations that you have never dealt with before. `depl` solves this.

Let's deploy Django to a live server! Create a file called `depl.yml`:

```
deploy:
    - django:
        settings: settings_prod
        ssl:
            key: /path/to/key
            certificate: /path/to/cert
hosts:
    - myhost.com
```

And run `depl deploy` in that same directory. After this two step process your website will be running on `https://myhost.com` and `http://myhost.com`. It only works on Ubuntu servers for now (maybe also Debian), but will work on other systems in the future as well.

`depl` would connect to `myhost.com` by SSH, install all necessary software packages and initialize your project. In our Django example this would involve installing `nginx`, `uwsgi`, `postgresql` (if required by Django's `settings.py`). It would install all Python dependencies and run on your production settings.

Currently this is also working for Meteor and WSGI (which includes flask).

CHAPTER 1

Why depl and not ansible, puppet, chef, docker or vagrant?

Have you ever tried those tools? ansible, puppet and chef are really complicated to understand. They are made for big deployments, for large scale operation engineers. You can read the blog posts below, they explain a bit more. Generally depl solves the use case for all small deployments on your own server, those tools do something different.

Docker and Vagrant are completely different tools. They make controlling VMs easy. They do not install your software. However, depl could be used in combination with either of them.

Blog Posts talking about depl

- The current state of deploying Django applications!
- Platform as a Service: A Market Analysis!

CHAPTER 3

Docs

3.1 Installation and Configuration

3.1.1 The preferred way

On any system you can install depl directly from the Python package index using pip:

```
sudo pip install depl
```

If you want to install the current development version (master branch):

```
sudo pip install -e git://github.com/davidhalter/depl.git#egg=depl
```

3.1.2 Manual installation from a downloaded package

If you prefer not to use an automated package installer, you can download a current copy of depl and install it manually.

To install it, navigate to the directory containing `setup.py` on your console and type:

```
sudo python setup.py install
```

3.2 Usage

3.2.1 `depl.yml` files

The most important thing in your depl application is that you create a depl.yml file that contains at least your deploy settings, e.g. deploying Django could look like

```
deploy:
    - django:
        settings: settings.prod
```

On the command line you can use `depl deploy myserver.com` to deploy your project. You can also add your `myserver.com` *to your settings*.

There is extensive documentation for all supported *Web Frameworks* and *Services*.

3.2.2 Command Line Options

Generally the command line interface looks like this:

```
Usage:
  depl (deploy|remove) [-c=<file>] [-p=<file>] [<host>...]
  depl run [-c=<file>] [-p=<file>] <command> [<host>...]
  depl -h | --help

Options:
  -c, --config=<file>   Deploy configuration file [default: depl.yml]
  -p, --pool=<name>     Define a pool that is going to be deployed.
```

Only the `deploy` and `run` commands are working. The `remove` command is already reserverd.

3.2.3 Hosts / Pools

While it's possible to easily define hosts (so you don't always have to use them while using `depl deploy myhost.com`):

```
deploy:
    - redis
hosts:
    - myhost.com:
        password: "the answer is 42"
    - host_with_key.com  # it is always prefered to use keys.
```

It's also possible to group hosts with pools:

```
deploy:
    - postgresql
    - django:
        port: 8080
hosts:
    - databases.com
        password: "pg rocks"
    - i_dont_like_long_urls.myhost.webservers.com
        id: webservers

pools:
    db:
        deploy: [postgresql]
        hosts: [databases.com]
    web:
        deploy: [django]
        hosts: [webservers]
```

I'm not sure if anybody is ever going to use that, though.

3.2.4 Extend other depl modules

Just use the extends identifier:

```
deploy:
    - redis
extends:
```

```
    - foo.yml
    - bar.yml
```

Files are automatically merged - it works like multiple inheritance.

3.2.5 Custom Scripts

There's two ways of writing custom scripts within depl. The first one is `sh` , a simple shell script (which shell depends currently on fabric):

```
deploy:
    - sh: |
        echo "Hello World"
```

The second one would be `fab`. Fab basically resembles the fab files of fabric. It's writing Python with an `from fabric.api import *` in front. This is also the way how depl scripts work internally:

```
deploy:
    - fab: |
        with warn_only():
            sudo('echo "Hello World"')
```

3.3 Web Frameworks

Currently we support the following frameworks/languages:

- *Django*
- *Meteor*
- *Python WSGI & Flask & others...*

Support for Rails, Java, Scala and whatever else is hopefully coming soon! We need experts, to tell us at least which tools we should use to deploy. So if you want to see your web framework, just join open an issue and start a discussion!

3.3.1 General Web Options & SSL

All are web frameworks (at least until now) are using nginx as a reverse proxy. This makes it possible to use a multitude of ports and frameworks on the same server.

There's a general set of web options, basically every web host has these options (read the grammar file!!!):

```
deploy:
    - meteor:
        url: localhost
        port: 80  # 0 to disable
        path: .
        redirect: null  # 'https' to redirect to the https port

        ssl:
            port: 443  # to disable
            redirect: null  # 'http' to redirect to the http port
            certificate: null  # a file path
            key: null  # a file path
```

As you can see, there's a way to play with the ports and specify an url (to host multiple domains on the same server). There's also a way to specify an ssl key/certificate. By default if not specified, depl generates self-signed certificates for you (obviously only working with a warning in the browser).

A typical django ssl configuration with ssh might look like this:

```
deploy:
    - django:
        redirect: https
        ssl:
            certificate: ~/.private/ssl/cert.crt
            key: ~/.private/ssl/ssl.key
```

Specifying the `path` allows a different source and project directory.

3.3.2 Django

3.3.3 Meteor

3.3.4 Python WSGI & Flask & others...

3.4 Services

Currently we support the following services:

- *PostgreSQL*
- *Redis*
- *MongoDB*

We need experts to increase our services and features. So please open an issue and start a discussion!

3.4.1 PostgreSQL

3.4.2 Redis

3.4.3 MongoDB

3.5 The Plugin API

There's no plugin API yet. We're focusing our efforts on improving the core.

The plugin api will look something like this:

```
plugins:
    - depl_my_django:
        path: depl_plugins/my_django
    - depl_sinatra  # just a plugin in python path

deploy:
    - my_django
    - my_sinatra
```

Obviously we would still need to define the exact way of how to write that plugin. But as you can see it's probably going to be pretty easy to write a plugin (just create a new python package).

3.6 depl Development

Note: This documentation is for depl developers who want to improve depl itself, but have no idea how depl works.

3.6.1 Introduction

`depl` is currently a very small program, so it should still be very easy to understand if you read the code. depl is built on fabric and there's no way to change things without knowing a little bit about fabric. If you happen to know fabric most depl code is really easy to understand.

The only thing that really deserves to be mentioned is the internal configuration verifier. There's a grammar file called `grammar.yml`, that is being compared to your own project-based `depl.yml` file. This grammar file compares all the types and if raises an error if something is wrongly formatted.

3.6.2 Testing

We heavily rely on pytest for testing. Testing `depl` correctly involves a lot of integration tests. Therefore I recommend you to use a virtual machine for testing. Alternatively you can just create a pull request, which travis automatically tests.

`depl` will open a lot of ports and testing might even create security holes on your computer - so really - use a VM.

You can run tests like this:

```
sudo pip install tox
sudo aptitude install libpq-dev python-dev

tox
```

The goal is to keep at least 90% testing coverage.

3.6.3 Contributing

See CONTRIBUTING.md.

CHAPTER 4

Resources

- Source Code on Github
- Travis Testing
- Python Package Index
- Test Coverage