Deploying with Fabric

Fabric is a tool for Python similar to Makefiles but with the ability to execute commands on a remote server. In combination with a properly set up Python package (Larger Applications) and a good concept for configurations (Configuration Handling) it is very easy to deploy Flask applications to external servers.

Before we get started, here a quick checklist of things we have to ensure upfront:

- Fabric 1.0 has to be installed locally. This tutorial assumes the latest version of Fabric.
- The application already has to be a package and requires a working `setup.py` file (Deploying with Setuptools).
- In the following example we are using *mod_wsgi* for the remote servers. You can of course use your own favourite server there, but for this example we chose Apache + *mod_wsgi* because it's very easy to setup and has a simple way to reload applications without root access.

Creating the first Fabfile

A fabfile is what controls what Fabric executes. It is named `fabfile.py` and executed by the *fab* command. All the functions defined in that file will show up as *fab* subcommands. They are executed on one or more hosts. These hosts can be defined either in the fabfile or on the command line. In this case we will add them to the fabfile.

This is a basic first example that has the ability to upload the current source code to the server and install it into a pre-existing virtual environment:

```
from fabric.api import *

# the user to use for the remote commands
env.user = 'appuser'
# the servers where the commands are executed
env.hosts = ['server1.example.com', 'server2.example.com']

def pack():
    # build the package
    local('python setup.py sdist --formats=gztar', capture=False)

def deploy():
    # figure out the package name and version
    dist = local('python setup.py --fullname', capture=True).strip()
    filename = '%s.tar.gz' % dist

    # upload the package to the temporary folder on the server
```

```
    put('dist/%s' % filename, '/tmp/%s' % filename)

    # install the package in the application's virtualenv with pip
    run('/var/www/yourapplication/env/bin/pip install /tmp/%s' % filena

    # remove the uploaded package
    run('rm -r /tmp/%s' % filename)

    # touch the .wsgi file to trigger a reload in mod_wsgi
    run('touch /var/www/yourapplication.wsgi')
```

Running Fabfiles

Now how do you execute that fabfile? You use the *fab* command. To deploy the current version of the code on the remote server you would use this command:

```
$ fab pack deploy
```

However this requires that our server already has the `/var/www/yourapplication` folder created and `/var/www/yourapplication/env` to be a virtual environment. Furthermore are we not creating the configuration or `.wsgi` file on the server. So how do we bootstrap a new server into our infrastructure?

This now depends on the number of servers we want to set up. If we just have one application server (which the majority of applications will have), creating a command in the fabfile for this is overkill. But obviously you can do that. In that case you would probably call it *setup* or *bootstrap* and then pass the servername explicitly on the command line:

```
$ fab -H newserver.example.com bootstrap
```

To setup a new server you would roughly do these steps:

1. Create the directory structure in `/var/www`:

   ```
   $ mkdir /var/www/yourapplication
   $ cd /var/www/yourapplication
   $ virtualenv --distribute env
   ```

2. Upload a new `application.wsgi` file to the server and the configuration file for the application (eg: `application.cfg`)
3. Create a new Apache config for `yourapplication` and activate it. Make sure to activate watching for changes of the `.wsgi` file so that we can automatically reload the ap-

plication by touching it. (See mod_wsgi (Apache) for more information)

So now the question is, where do the `application.wsgi` and `application.cfg` files come from?

The WSGI File

The WSGI file has to import the application and also to set an environment variable so that the application knows where to look for the config. This is a short example that does exactly that:

```
import os
os.environ['YOURAPPLICATION_CONFIG'] = '/var/www/yourapplication/applic
from yourapplication import app
```

The application itself then has to initialize itself like this to look for the config at that environment variable:

```
app = Flask(__name__)
app.config.from_object('yourapplication.default_config')
app.config.from_envvar('YOURAPPLICATION_CONFIG')
```

This approach is explained in detail in the Configuration Handling section of the documentation.

The Configuration File

Now as mentioned above, the application will find the correct configuration file by looking up the `YOURAPPLICATION_CONFIG` environment variable. So we have to put the configuration in a place where the application will able to find it. Configuration files have the unfriendly quality of being different on all computers, so you do not version them usually.

A popular approach is to store configuration files for different servers in a separate version control repository and check them out on all servers. Then symlink the file that is active for the server into the location where it's expected (eg: `/var/www/yourapplication`).

Either way, in our case here we only expect one or two servers and we can upload them ahead of time by hand.

First Deployment

Now we can do our first deployment. We have set up the servers so that they have their virtual environments and activated apache configs. Now we can pack up the application and deploy it:

```
$ fab pack deploy
```

Fabric will now connect to all servers and run the commands as written down in the fabfile. First it will execute pack so that we have our tarball ready and then it will execute deploy and upload the source code to all servers and install it there. Thanks to the `setup.py` file we will automatically pull in the required libraries into our virtual environment.

Next Steps

From that point onwards there is so much that can be done to make deployment actually fun:

- Create a *bootstrap* command that initializes new servers. It could initialize a new virtual environment, setup apache appropriately etc.
- Put configuration files into a separate version control repository and symlink the active configs into place.
- You could also put your application code into a repository and check out the latest version on the server and then install. That way you can also easily go back to older versions.
- hook in testing functionality so that you can deploy to an external server and run the test suite.

Working with Fabric is fun and you will notice that it's quite magical to type `fab deploy` and see your application being deployed automatically to one or more remote servers.