Comune di Zola Predosa

Fabric, miglior amico dei programmatori web pigri e annoiati

EuroPython 2013 - Florence, July 2

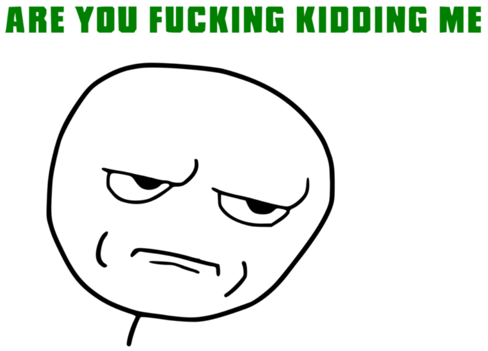
Simone Dalla @simodalla

CIO of Comune di Zola Predosa (Bologna).

Pythonista and Django programmer.

I use Python into my work environment for..... ALL!

You are a Web Programming (+1), you use Python (+1), you use Django (+1). Very good.
But your are lazy and bored by the deployment of your web application. Every time you must downloading the latest version of code from the VCS, needs to restart Apache, needs to update the new PsycoPg2 package on virtualenv from pypy, to back up the database, to restart your asynchronous task,queue,job with Celery. And and repeat the deployment on 5 web server??



Ok. My name is **Fabric***, I will make you save time, money and you will be less bored and by now I'll be your new best friend.*

Comune di
Zola Predosa

What am I?

A Python library/tool that is designed to use SSH to execute:

- system administration tasks
- deployment tasks

on one or more remote machines.

Why I'll be your best friend?

I'm a simple fire-and-forget tool that will make your life so much simpler.
Not only can you run simple tasks via SSH on multiple machines, but since you’re using Python code to execute items, you can combine it with any arbitrary Python code to make robust, complex, elegant and pythonic applications for deployment or administration tasks.

Installation

I require Python 2.5 or later and you use pip (or easy_install)

```
$ pip install fabric
```

on most systems, you can use your system’s package manager (apt-get, install, and so on) to install me

```
$ sudo apt-get install fabric
```

If you’re feeling froggy, you can check out the Git repository and hack away at the source code

https://github.com/fabric/fabric

The basics

- need a "fabfile" (typically a file called fabfile.py)
- from fabric.api import *

```
from fabric.api import *

def host_type():
    run('uname -s')
```

```
$ fab -u root -H server1.local host_type
```

Comune di Zola Predosa

"Base" operations

- ***get(remote_path, local_path=None) -*** get allows you to pull files from the remote machine to your local machine. The remote path is the path of the file on the remote machine that you are grabbing, and the local path is the path to which you want to save the file on the local machine.

- ***put(local_path, remote_path, use_sudo=False, mirror_local_mode=False, mode=None)*** — this is the opposite command of get, although you are given more options when putting to a remote system than getting. The local path can be a relative or absolute file path, or it can be an actual file object. If use_sudo=True is specified, Fabric will put the file in a temporary location on the remote machine, then use sudo to move it from the temporary location to the specified location. If you want the file mode preserved through the copy, use mirror_local_mode=True; otherwise, you can set the mode using mode.

Comune di Zola Predosa

"Base" operations

- ***open_shell(command=None)*** - this function is mostly for debugging purposes. It opens an interactive shell on the remote end, allowing you to run any number of commands.
- ***local(command, capture=False)*** - the local function allows you to take action on the local host in a similar fashion to the Python subprocess module (in fact, local is a simplistic wrapper that sits on top of the subprocess module). Simply supply the command to run and, if needed, whether you want to capture the output. If you specify capture=True, the output will be returned as a string from local; otherwise, it will be output to STDOUT.
- ***reboot(wait=120)*** - reboot does exactly what it says: reboots the remote machine. By default, reboot will wait 120 seconds before attempting to reconnect to the machine to continue executing any following commands.

"Base" operations

- ***prompt(text, key=None, default='', validate=None)***
  - in the case when you need to supply a value, but don’t want to specify it on the command line for whatever reason, prompt is the ideal way to do this.

- ***require(*keys, **kwargs)*** - require forces the specified keys to be present in the shared environment dict in order to continue execution. If these keys are not present, Fabric will abort. Optionally, you can specify used_for to indicate what the key is used for in this particular context

"Base" operations

- ***run(command, shell=True, pty=True, combine_stderr=True, quiet=False, warn_only=False, stdout=None, stderr=None)***

  this and sudo are the two most used functions in Fabric, because they actually execute commands on the remote host (which is the whole point of Fabric). With run, you execute the specified command as the given user. run returns the output from the command as a string that can be checked for a failed, succeeded and return_code attribute. ***shell*** controls whether a shell interpreter is created for the command. If turned off, characters will not be escaped automatically in the command. Passing ***pty=False*** causes a psuedo-terminal not to be created while executing this command; this can have some benefit if the command you are running has issues interacting with the psuedo-terminal, but otherwise, it will be created by default. If stderr from the command to be parsable separately from stdout, use ***combine_stderr=False*** to indicate that. ***quiet=True*** will cause the command to run silently, sending no output to the screen while executing. When an error occurs in Fabric, typically the script will abort and indicate as such. You can indicate that Fabric need not abort if a particular command errors using the ***warn_only*** argument. Finally, you can redirect where the remote stderr and stdout redirect to on the local side. If you want the stderr to pipe to stdout on the local end, you could indicate that with ***stderr=sys.stdout***

"Base" operations

- ***sudo(command, shell=True, pty=True, combine_stderr=True, user=None, quiet=False, warn_only=False, stdout=None, stderr=None, group=None)***

  sudo works precisely like run, except that it will elevate privileges prior to executing the command. It basically works the same as if you'd run the command using run, but prepended sudo to the front of command. sudo also takes ***user*** and ***group*** arguments, allowing you to specify which user or group to run the command as. As long as the original user has the permissions to escalate for that particular user/group and command, you are good to go.

Comune di Zola Predosa

Authentication

- Relies on SSH model
- Use SSH keys
- Control access to root user via sudoers
- When you have to revoke access, you just turn off their SSH account

Configuration

- fabric environment (***env***) -- it's just a dictionary
- Hosts and Roles
- Whatever you need
- ~/.fabricrc
- fab command-line options

Comune di Zola Predosa

Development administration tasks

```
def start_vm_demo():
    vm_name = env['host'].split('.')[0]
    vm_path = os.path.expanduser(
        '~/Documents/Virtual Machines.localized/'
        '{}.vmwarevm/'.format(vm_name))
    local('ls "{}"'.format(vm_path))
    with path(
            '"/Applications/VMware Fusion.app/'
            'Contents/Library"'):
        local('vmrun -T fusion start "{}"'.format(vm_path))
```

```
$ fab -u root -H server1.local, server2.local start_vm_demo
```

Comune di
Zola Predosa

System administration tasks

```
def update_system_packages():
    run('apt-get update && apt-get upgrade -y')


def install_system_package(package=None):
    if package is None:
        package = prompt('Which package? ')
    run('apt-get install -y {}'.format(package))
```

```
$ fab -u root -H server1.local, server2.local update_system_packages
```

```
$ fab -u root -H server1.local, server2.local install_system_package:apache2-mod-wsgi
```

Comune di Zola Predosa

Deployment 'bootstrap' task scenario

- create 'europython2013' virtualev
- install mezzanine and psycopg2 packages from pypi into virtualenv
- cloning project's code from Github
- create a production postgres db
- 'clonig' development db to production db (pg_dump, pg_restore)
- call django command 'collectstatic' on virtualenv
- restart apache webserver

Comune di Zola Predosa

Deployment 'bootstrap' task

```
def bootstrap(virtualenv='europython2013',
                    project='europython2013_talk_mezzanine'):
    prepare_virtualenv(virtualenv)
    with cd('/opt/projects/'):
        run('git clone https://github.com/simodalla/{}.git'.format(project))
    database_name = 'europython2013_demo'
    run('createdb -U postgres {}'.format(database_name))
    with settings(warn_only=True):
        local('pg_dump -h 127.0.0.1 -Ft {} |'
                ' pg_restore -U postgres -h {} -d {}'.format(database_name,
                                                              env['host'],
                                                              database_name))
    project_path = '/opt/projects/{}/europython2013_demo'.format(project)
    with cd(project_path), prefix('workon {}'.format(virtualenv)):
        run('python manage.py collectstatic --noinput')
    run('service apache2 restart')
```

Comune di Zola Predosa

Deployment 'bootstrap' task

```
def prepare_virtualenv(virtualenv='europython2013'):
    requirements = 'requirements.txt'
    put(requirements)
    if not exists('/opt/virtualenvs/{}'.format(virtualenv)):
        run('mkvirtualenv --no-site-packages --distribute --clear {}'.format(
            virtualenv))
    with prefix('workon {}'.format(virtualenv)):
        run('lssitepackages')
        run('pip install -r {}'.format(requirements))
```

Comune di Zola Predosa

Deployment task scenario

- are you sure???
- backup 'old' version code
- backup postgres database
- 'pull' new version of code from Github
- call django command 'collectstatic' on virtualenv
- restart apache webserver
- restart other server (celery, supervisot, rabbitmq...)

Comune di
Zola Predosa

Deployment task

```
def deploy(virtualenv='europython2013',
           project='europython2013_talk_mezzanine'):
    now = datetime.datetime.now()
    if not confirm('Sei sicuro di voler fare il deploy del progetto'
                   ' in produzione?', default=False):
        abort('Deploy aborted.')
    project_path = '/opt/projects/{}/europython2013_demo'.format(project)
    database_name = 'europython2013_demo'
    run('tar cfz /opt/projects/backup_{}_{}.tar.gz --exclude={}/static'
        ' {}'.format(project, now.strftime(BACKUP_DATE_FORMAT),
                     project_path, project_path))
    run('pg_dump -U postgres -Ft {} >'
        ' /opt/projects/pg_{}_{}.dump'.format(
          database_name, project, now.strftime(BACKUP_DATE_FORMAT)))
    with cd('{}/../'.format(project_path)):
        run('git pull')
    with cd(project_path), prefix('workon {}'.format(virtualenv)):
        run('python manage.py collectstatic --noinput', quiet=True)
    run('service apache2 restart')
    run(supervisorctl restart all)
```

Tips and Tricks

- be mindful of task thay may fail
- each remote command start fresh
- changing directory
- show or hide remote command output

- ***Context Manager***: cd, prexif, path, settings, hide, show (other...)
- ***Decorators***: hosts, roles, tasks
- ***Contrib Api***: file and directory managers, console output utilities, django integration...

Strategy

- 'single centralized' fabfile for several projects, including different types (use of different external configuration files with ConfigParser)
- specialized fabfile for a single project

REMEMBER: fabfile is Python code, enjoy!!!

More information

Documentation
www.fabfile.org

Fab-user mailing list http://lists.nongnu.org/mailman/listinfo/fab-user

Twitter account @pyfabric
http://twitter.com/pyfabric

Github Issues page
https://github.com/fabric/fabric/issues

Comune di Zola Predosa

Questions?

Fabfile code

https://github.com/simodalla/europython2013_talk_fabric

Demo Mezzanine Project code

https://github.com/simodalla/europython2013_talk_mezzanine