Tissu for Fabric Documentation

Release 0.1-alpha

Thierry Stiegler

July 17, 2014

Contents

CHAPTER 1

About

Fabric is an incredible tool to automate deployement of application or administration of remote machines. As Fabric's function are rather low-level, you'll probably quickly see a need for managing different environnements like *production*, *staging*...

Tissu give you the hability to define your environement (the remote machines) and the role of each machine.

Tissu's features are:

- a simple convention
- settings à la Django
- settings with python file

Contents:

1.1 Installation

Tissu is on pypi.

1.1.1 Setuptools

```
easy_install -U tissu
```

1.1.2 PIP

We are not stabe for the moment, so you need the extra argument –pre

```
pip install --pre tissu
```

1.1.3 Source code

- Get the source from Tissu's GitHub
- run `python setup.py install`.

1.2 Quickstart

1.2.1 Use the role mechanism of tissu in your fabfile.py

First, you need to install tissu...

So create your fabfile.py with these lines of code.

```
import os, sys

current_dir = os.path.dirname(os.path.abspath(__file__))
sys.path.append(current_dir)

from fabric.api import *
from tissu.tasks import *
```

We need to add the current directory to sys.path. Without these lines the environnement loader can't load any settings.

Now you have 3 tasks, *e* and *tissu_init*.

```
$ fab -l
Available commands:

    e               Environnement loader
    tissu_init      Init the settings scaffolding
    tissu_print     Print FABRIC env dict and TISSU settings public properties
```

Now you have to generate the default settings scaffolding. That's the purpose of *tissu_init*. By default, Tissu need a python module "settings.default".

Now, launch the taks *tissu_ini*

```
$ fab tissu_init
[localhost] local: mkdir -p settings
[localhost] local: touch settings/__init__.py
settings/default.py written

Done.
```

Now tissu had everithing he need for handle your first environment.

```
.
-- fabfile.py
-- settings
    -- default.py
    -- __init__.py
```

Create a new file in settings foo.py, with this sample of code.

```
# My first environnement

SERVER_A = {
 "user" : "foo",
 "hostname" : "localhost",
 "password": "password",
}

SERVER_B = {
 "user" : "foo",
 "hostname" : "server.example.org",
```

```
 "key": "/home/foo/.ssh/id_rsa.pub"

}

FABRIC_ROLES = {
    "db":        [SERVER_A],
    "web":       [SERVER_B],
}
```

We can load our first environnement

```
fab e:foo
Environment foo sucessfully loaded :)

Done.
```

We will add some tasks in order to do something with this environement.

```
def hello():
    run("echo hello && uptime")

@roles("db")
@task
def hello_db():
    hello()

@roles("web")
@task
def hello_web():
    hello()

@roles("all")
@task
def hello_all():
    hello()
```

Now you have three new tasks :

- hello_db will execute hello to the *SERVER_A*, because it is in the role *db*.
- hello_web will execute hello to the *SERVER_B*, because it is in the role *web*.
- hello_all will execute hello in all remote machines.

Note: The role *all* is automaticly generate by tissu. **Note**: You have to setup the foo settings with real remote machines.

A example of what you get when you try these tasks :

```
$ fab e:foo hello_db hello_web hello_all
Environment foo sucessfully loaded :)
[thierry@localhost:22] Executing task 'hello_db'
[thierry@localhost:22] run: echo hello && uptime
[thierry@localhost:22] out: hello
[thierry@localhost:22] out:  16:17:57 up 13 days,  6:02,  5 users,  load average: 0,16, 0,31, 0,38
[thierry@localhost:22] out:

[thierry@192.168.0.19:22] Executing task 'hello_web'
[thierry@192.168.0.19:22] run: echo hello && uptime
[thierry@192.168.0.19:22] out: hello
[thierry@192.168.0.19:22] out:  16:18:00 up 13 days,  6:02,  5 users,  load average: 0,23, 0,32, 0,38
[thierry@192.168.0.19:22] out:
```

```
[thierry@192.168.0.19:22] Executing task 'hello_all'
[thierry@192.168.0.19:22] run: echo hello && uptime
[thierry@192.168.0.19:22] out: hello
[thierry@192.168.0.19:22] out:  16:18:00 up 13 days,  6:02,  5 users,  load average: 0,23, 0,32, 0,38
[thierry@192.168.0.19:22] out:

[thierry@localhost:22] Executing task 'hello_all'
[thierry@localhost:22] run: echo hello && uptime
[thierry@localhost:22] out: hello
[thierry@localhost:22] out:  16:18:00 up 13 days,  6:02,  5 users,  load average: 0,23, 0,32, 0,38
[thierry@localhost:22] out:
```

1.2.2 Retrieve the settings loaded by tissu

We assume that you have a foo.py settings created.

First we add a new settings : *BAR*, our foo settings become :

:: # My first environnement

SERVER_A = { "user" : "thierry", "hostname" : "localhost", "password": "cordonchat",

}

SERVER_B = { "user" : "thierry", "hostname" : "192.168.0.19", "password": "cordonchat"

}

FABRIC_ROLES = { "db": [SERVER_A], "web": [SERVER_B],

}

BAR = "foo"

In our fabfile, we add a new task that print us the value of BAR.

```
@task
def bar():
    from tissu.conf import settings
    puts("BAR value is: %s" % getattr(settings, "BAR", "not found !"))
```

When we don't specify any environnement we've got :

```
$ fab bar
BAR value is: ERROR !

Done.
```

And when we use our foo environement we've got :

```
$ fab e:foo bar
Environment foo sucessfully loaded :)
BAR value is: foo

Done.
```

Additionnaly, Tissu add the settings into the fabric env.

```
@task
def envbar():
```

```
    from fabric.api import env
    puts("BAR value from env is: %s " % getattr(env.my_settings, "BAR", "not found !"))
```

And then we got :

```
$ fab e:foo envbar
Environment foo sucessfully loaded :)
BAR value from env is: foo

Done.
```

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*
- `API doc`