Ansible

Puppet

Chef

**Puppet，Chef，Ansible的共性**

<https://my.oschina.net/zjzhai/blog/600430>

**Fabric 官方文档**

<http://docs.fabfile.org/>

**PYTHON FABRIC实现远程操作和部署**

<http://wklken.me/posts/2013/03/25/python-tool-fabric.html>

**Python远程部署利器Fabric详解**

<http://python.jobbole.com/87241/>

Fabric是一个Python的库，它提供了丰富的同SSH交互的接口，可以用来在本地或远程机器上自动化、流水化地执行Shell命令。因此它非常适合用来做应用的远程部署及系统维护。其上手也极其简单，你需要的只是懂得基本的Shell命令。本文将为大家详细介绍Fabric的使用。

**安装Fabric**

首先Python的版本必须是2.7以上，可以通过下面的命令查看当前Python的版本：

$ python -V

Fabric的官网是www.fabfile.org，源码托管在Github上。你可以clone源码到本地，然后通过下面的命令来安装。

$ python setup.py develop

在执行源码安装前，你必须先将Fabric的依赖包Paramiko装上。所以，个人还是推荐使用pip安装，只需一条命令即可：

$ pip install fabric

**第一个例子**

万事从Hello World开始，我们创建一个”fabfile.py”文件，然后写个hello函数：

def hello():

print "Hello Fabric!"

现在，让我们在”fabfile.py”的目录下执行命令：

$ fab hello

你可以在终端看到”Hello Fabric!”字样。

简单解释下，”fabfile.py”文件中每个函数就是一个任务，任务名即函数名，上例中是”hello”。”fab”命令就是用来执行”fabfile.py”中定义的任务，它必须显式地指定任务名。你可以使用参数”-l”来列出当前”fabfile.py”文件中定义了哪些任务：

$ fab -l

任务可以带参数，比如我们将hello函数改为：

def hello(name, value):

print "Hello Fabric! %s=%s" % (name,value)

此时执行hello任务时，就要传入参数值：

$ fab hello:name=Year,value=2016

Fabric的脚本建议写在”fabfile.py”文件中，如果你想换文件名，那就要在”fab”命令中用”-f”指定。比如我们将脚本放在”script.py”中，就要执行：

$ fab -f script.py hello

**执行本地命令**

“fabric.api”包里的”local()”方法可以用来执行本地Shell命令，比如让我们列出本地”/home/bjhee”目录下的所有文件及目录：

from fabric.api import local

def hello():

local('ls -l /home/bjhee/')

“local()”方法有一个”capture”参数用来捕获标准输出，比如：

def hello():

output = local('echo Hello', capture=True)

这样，Hello字样不会输出到屏幕上，而是保存在变量output里。”capture”参数的默认值是False。

**执行远程命令**

Fabric真正强大之处不是在执行本地命令，而是可以方便的执行远程机器上的Shell命令。它通过SSH实现，你需要的是在脚本中配置远程机器地址及登录信息：

from fabric.api import run, env

env.hosts = ['example1.com', 'example2.com']

env.user = 'bjhee'

env.password = '111111'

def hello():

run('ls -l /home/bjhee/')

“fabric.api”包里的”run()”方法可以用来执行远程Shell命令。上面的任务会分别到两台服务器”example1.com”和”example2.com”上执行”ls -l /home/bjhee/”命令。这里假设两台服务器的用户名都是”bjhee”，密码都是6个1。你也可以把用户直接写在hosts里，比如：

env.hosts = ['bjhee@example1.com', 'bjhee@example2.com']

如果你的”env.hosts”里没有配置某个服务器，但是你又想在这个服务器上执行任务，你可以在命令行中通过”-H”指定远程服务器地址，多个服务器地址用逗号分隔：

$ fab -H bjhee@example3.com,bjhee@example4.com hello

另外，多台机器的任务是串行执行的，关于并行任务的执行我们在之后会介绍。

如果对于不同的服务器，我们想执行不同的任务，上面的方法似乎做不到，那怎么办？我们要对服务器定义角色：



现在让我们执行：

$ fab task

这时Fabric会先在一台build服务器上执行build任务，然后在两台staging服务器上分别执行deploy任务。”@roles”装饰器指定了它所装饰的任务会被哪个角色的服务器执行。

如果某一任务上没有指定某个角色，但是你又想让这个角色的服务器也能运行该任务，你可以通过”-R”来指定角色名，多个角色用逗号分隔：

$ fab -R build deploy

这样”build”和”staging”角色的服务器都会运行”deploy”任务了。注：”staging”是装饰器默认的，因此不用通过”-R”指定。

此外，上面的例子中，服务器的登录密码都是明文写在脚本里的。这样做不安全，推荐的方式是设置SSH自动登录，具体方法大家可以去网上搜搜。

**SSH功能函数**

到目前为止，我们介绍了”local()”和”run()”函数分别用来执行本地和远程Shell命令。Fabric还提供了其他丰富的功能函数来辅助执行命令，这里我们介绍几个常用的：

* sudo: 以超级用户权限执行远程命令

功能类似于”run()”方法，区别是它相当于在Shell命令前加上了”sudo”，所以拥有超级用户的权限。使用此功能前，你需要将你的用户设为sudoer，而且无需输密码。具体操作可参见我的这篇文章。

from fabric.api import env, sudo

env.hosts = ['bjhee@example1.com', 'bjhee@example2.com']

env.password = '111111'

def hello():

sudo('mkdir /var/www/myapp')

* get(remote, local): 从远程机器上下载文件到本地

它的工作原理是基于scp命令，使用的方法如下：

from fabric.api import env, get

env.hosts = ['bjhee@example.com',]

env.password = '111111'

def hello():

get('/var/log/myapp.log', 'myapp-0301.log')

上述任务将远程机上”/var/log/myapp.log”文件下载到本地当前目录，并命名为”myapp-0301.log”。

* put(local, remote): 从本地上传文件到远程机器上

同get一样，put方法也是基于scp命令，使用的方法如下：

from fabric.api import env, put

env.hosts = ['bjhee@example1.com', 'bjhee@example2.com']

env.password = '111111'

def hello():

put('/tmp/myapp-0301.tar.gz', '/var/www/myapp.tar.gz')

上述任务将本地”/tmp/myapp-0301.tar.gz”文件分别上传到两台远程机的”/var/www/”目录下，并命名为”myapp.tar.gz”。如果远程机上的目录需要超级用户权限才能放文件，可以在”put()”方法里加上”use\_sudo”参数：

put('/tmp/myapp-0301.tar.gz', '/var/www/myapp.tar.gz', use\_sudo=True)

* prompt: 提示输入

该方法类似于Shell中的”read”命令，它会在终端显示一段文字来提示用户输入，并将用户的输入保存在变量里：

from fabric.api import env, get, prompt

env.hosts = ['bjhee@example.com',]

env.password = '111111'

def hello():

filename = prompt('Please input file name: ')

get('/var/log/myapp.log', '%s.log' % filename)

现在下载后的文件名将由用户的输入来决定。我们还可以对用户输入给出默认值及类型检查：

port = prompt('Please input port number: ', default=8080, validate=int)

执行任务后，终端会显示：

Please input port number: [8080]

如果你直接按回车，则port变量即为默认值8080；如果你输入字符串，终端会提醒你类型验证失败，让你重新输入，直到正确为止。

* reboot: 重启服务器

看方法名就猜到了，有时候安装好环境后，需要重启服务器，这时就要用到”reboot()”方法，你可以用”wait”参数来控制其等待多少秒后重启，没有此参数则代表立即重启：

from fabric.api import env, reboot

env.hosts = ['bjhee@example.com',]

env.password = '111111'

def restart():

reboot(wait=60)

上面的restart任务将在一分钟后重启服务器。

**上下文管理器**

Fabric的上下文管理器是一系列与Python的”with”语句配合使用的方法，它可以在”with”语句块内设置当前工作环境的上下文。让我们介绍几个常用的：

* cd: 设置远程机器的当前工作目录

“cd()”方法在之前的范例中出现过，”with cd()”语句块可以用来设置远程机的工作目录：

from fabric.api import env, cd, put

env.hosts = ['bjhee@example1.com', ]

env.password = '111111'

def hello():

with cd('/var/www/'):

put('/tmp/myapp-0301.tar.gz', 'myapp.tar.gz')

上例中的文件会上传到远程机的”/var/www/”目录下。出了”with cd()”语句块后，工作目录就回到初始的状态，也就是”bjhee”用户的根目录。

* lcd: 设置本地工作目录

“lcd()”就是”local cd”的意思，用法同”cd()”一样，区别是它设置的是本地的工作目录：

from fabric.api import env, cd, lcd, put

env.hosts = ['bjhee@example1.com', ]

env.password = '111111'

def hello():

with cd('/var/www/'):

with lcd('/tmp/'):

put('myapp-0301.tar.gz', 'myapp.tar.gz')

这个例子的执行效果跟上个例子一样。

* path: 添加远程机的PATH路径



假设我们的PATH环境变量默认是”/sbin:/bin”，在上述”with path()”语句块内PATH变量将变为”/sbin:/bin:/home/bjhee/tmp”。出了with语句块后，PATH又回到原来的值。

* settings: 设置Fabric环境变量参数

Fabric环境变量即是我们例子中一直出现的”fabric.api.env”，它支持的参数可以从官方文档中查到。

from fabric.api import env, run, settings

env.hosts = ['bjhee@example1.com', ]

env.password = '111111'

def hello():

with settings(warn\_only=True):

run('echo $USER')

我们将环境参数”warn\_only”暂时设为True，这样遇到错误时任务不会退出。

* shell\_env: 设置Shell环境变量

可以用来临时设置远程和本地机上Shell的环境变量。

from fabric.api import env, run, local, shell\_env

env.hosts = ['bjhee@example1.com', ]

env.password = '111111'

def hello():

with shell\_env(JAVA\_HOME='/opt/java'):

run('echo $JAVA\_HOME')

local('echo $JAVA\_HOME')

* prefix: 设置命令执行前缀

from fabric.api import env, run, local, prefix

env.hosts = ['bjhee@example1.com', ]

env.password = '111111'

def hello():

with prefix('echo Hi'):

run('pwd')

local('pwd')

在上述”with prefix()”语句块内，所有的”run()”或”local()”方法的执行都会加上”echo Hi && “前缀，也就是效果等同于：

run('echo Hi && pwd')

local('echo Hi && pwd')

配合后一节我们会讲到的错误处理，它可以确保在”prefix()”方法上的命令执行成功后才会执行语句块内的命令。

**错误处理**

默认情况下，Fabric在任务遇到错误时就会退出，如果我们希望捕获这个错误而不是退出任务的话，就要开启”warn\_only”参数。在上面介绍”settings()”上下文管理器时，我们已经看到了临时开启”warn\_only”的方法了，如果要全局开启，有两个办法：

* 在执行”fab”命令时加上”-w”参数

$ fab -w hello

* 设置”env.warn\_only”环境参数为True

from fabric.api import env

env.warn\_only = True

现在遇到错误时，控制台会打出一个警告信息，然后继续执行后续任务。那我们怎么捕获错误并处理呢？像”run()”, “local()”, “sudo()”, “get()”, “put()”等SSH功能函数都有返回值。当返回值的”succeeded”属性为True时，说明执行成功，反之就是失败。你也可以检查返回值的”failed”属性，为True时就表示执行失败，有错误发生。在开启”warn\_only”后，你可以通过”failed”属性检查捕获错误，并执行相应的操作。

from fabric.api import env, cd, put

env.hosts = ['bjhee@example1.com', ]

env.password = '111111'

def hello():

with cd('/var/www/'):

upload = put('/tmp/myapp-0301.tar.gz', 'myapp.tar.gz')

if upload.failed:

sudo('rm myapp.tar.gz')

put('/tmp/myapp-0301.tar.gz', 'myapp.tar.gz', use\_sudo=True)

**并行执行**

我们在介绍执行远程命令时曾提到过多台机器的任务默认情况下是串行执行的。Fabric支持并行任务，当服务器的任务之间没有依赖时，并行可以有效的加快执行速度。怎么开启并行执行呢？办法也是两个：

* 在执行”fab”命令时加上”-P”参数

$ fab -P hello

* 设置”env.parallel”环境参数为True

from fabric.api import env

env.parallel = True

* 如果，我们只想对某一任务做并行的话，我们可以在任务函数上加上”@parallel”装饰器

from fabric.api import parallel

@parallel

def runs\_in\_parallel():

pass

def runs\_serially():

pass

这样即便并行未开启，”runs\_in\_parallel()”任务也会并行执行。反过来，我们可以在任务函数上加上”@serial”装饰器：

from fabric.api import serial

def runs\_in\_parallel():

pass

@serial

def runs\_serially():

pass

这样即便并行已经开启，”runs\_serially()”任务也会串行执行。

**补充**

这个部分用来补充Fabric的一些特别功能：

* 终端输出带颜色

我们习惯上认为绿色表示成功，黄色表示警告，而红色表示错误，Fabric支持带这些颜色的输出来提示相应类型的信息：

from fabric.colors import \*

def hello():

print green("Successful")

print yellow("Warning")

print red("Error")

* 限制任务只能被执行一次

通过”execute()”方法，可以在一个”fab”命令中多次调用同一任务，如果想避免这个发生，就要在任务函数上加上”@runs\_once”装饰器。

from fabric.api import execute, runs\_once

@runs\_once

def hello():

print "Hello Fabric!"

def test():

execute(hello)

execute(hello)

现在不管我们”execute”多少次hello任务，都只会输出一次”Hello Fabric!”字样

更多内容请参阅Fabric的官方文档。