Cloud Infrastructure
Lecture 11
Deployment with Fabric

Gregory S. DeLozier, Ph.D.
Kent State University
March 1, 2017

Deployment

- Recall the secondary directive
    - Hands off the production servers!
- This leads to various automation tools
    - Puppet (later)
    - Chef (possibly never)
    - Ansible (later, but definitely a major consideration)
    - Fabric (now, because it's really easy and works well)

What is Fabric?

Fabric

- Started with remote SSH tool
    - Paramiko
      http://www.paramiko.org/
    - Allows remote control via SSH
- Fabric is an *organizational tool* on top of Paramiko
    - Organizes deployment commands sent to servers

A Little History

- Stage 1: Paramiko is a DIY kit for remote control.

- Stage 2: (Everyone) Hey! Fabric is awesome!

- Stage 3: (Author) I won't port Fabric, for some reason.

(long pause)

- Stage 4: Hey, it's open source! Let's port it ourselves!

- Stage 5: Much rejoicing, happiness through the land

Fabric 3

- Python3 port of Fabric

- $ pip3 install fabric3

- Completely compatible with Fabric (2.x)

- Documentation is at
  http://www.fabfile.org
  http://docs.fabfile.org

Using Fabric

Fabric Standalone

- Command 'fab' is installed
- $ *fab* <commands>
- *fab* executes against 'fabfile.py'
- Commands select things *inside* the fabfile.

Example fabfile.py

```
def hello():
    print("Hello world!")

def goodbye():
    print("Goodbye!")
```

Running *fab*

```
$ fab hello
Hello!

Done.

$ fab goodbye
Goodbye!

Done.

$
```

Fab Parameters

```
def hello(name="world"):
    print("Hello %s!" % name)
```

Fab Commands with Parameters

```
$ fab hello:name=Jeff
Hello Jeff!

Done.

$ fab hello:Jeff
Hello Jeff!

Done.
```

Talking to the Local Machine

```
from fabric.api import local

def prepare_deploy():
    local("./manage.py test my_app")
    local("git add -p && git commit")
    local("git push")
```

Breaking up Tasks

```
from fabric.api import local

def test():
    local("./manage.py test my_app")

def commit():
    local("git add -p && git commit")

def push():
    local("git push")

def prepare_deploy():
    test()
    commit()
    push()
```

Defining a Remote Server

- The environment variable "hosts"

- Set it in code:

```
env.hosts = ['my_server']
```

- Otherwise *fab* will prompt you

Some Simple Remote Commands

```
from fabric.api import run, env

env.hosts = ['host1', 'host2']

def taskA():
    run('ls')

def taskB():
    run('whoami')
```

Controlling a Remote Server

- Use the "run" command

```
def deploy():
    code_dir = '/srv/django/myproject'
    with cd(code_dir):
        run("git pull")
        run("touch app.wsgi")
```

- Also note the 'cd' context manager
- Necessary because each 'run' is a new session

Fab Imports

- You need a few imports to make this all work

```
from fabric.api import local, settings, abort, run, cd
from fabric.contrib.console import confirm
```

Let's Try It

Setting up SSH keys

- Setting up SSH keys means no password for SSH login

- See discussion here:

  https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys--2

- You can install keys directly into DO & GH

  https://www.digitalocean.com/community/tutorials/how-to-use-ssh-keys-with-digitalocean-droplets

  https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/

About 'disown'

- *disown* causes a job to stay running after logout

- $ *python -m http.server 8080 & disown*

- See discussion here:

  http://unix.stackexchange.com/questions/3886/difference-between-nohup-disown-and

  (And various other places around the web…)

Demo Time

- Get an empty server
- Remotely, deploy libraries and dependencies
- Remotely, get the code from GitHub
- Remotely, start the service

Reading

Links embedded in the slides today…

Specifically, read the documentation at docs.fabfile.org
```
```