All about Fabric

Simple management / deployment tool

What is Fabric

Fabric is a Python (2.5-2.7) library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks.

It provides a basic suite of operations for executing local or remote shell commands (normally or via sudo) and uploading/downloading files, as well as auxiliary functionality such as prompting the running user for input, or aborting execution

High-level key points

- Plain old Python
- Very simple api and easy to use
- Easy to create individual tasks
- Easy to chain together multiple tasks to create more complex scripts
- Notice how many “easies” appear above

Getting started

- pip install fabric
- create a file named fabfile.py (default, but can be named anything)
- import fabric modules
- profit? $$$

Defining a task

- from fabric.api import task
- use `@task` decorator to define a task
- tasks can take arguments
  - fab command:arg1=val1,arg2=val2
- That’s it!

Simple example

```
from fabric.api import run, task, env

env.use_ssh_config = True

@task
def date():
    run("date")
    print "Date on {}".format(env.host)
```

Simple example using args

```
from fabric.api import run, task, env

env.use_ssh_config = True

@task
def ls(path="/tmp"):
    run("ls {}".format(path))
```

Core functions

- local() - Runs a command locally
- run() - Runs a command remotely
- sudo() - Runs a command remotely as another user
- put() - Copy a file from local to remote
  - use_sudo=True to put the file as another user
- get() - Copy a file from remote to local

local()

```
from fabric.api import task, env, local

env.use_ssh_config = True

@task
def date():
    local("date")
    print "Date on {}".format(env.host)
```

run()

```
from fabric.api import run, task, env

env.use_ssh_config = True

@task
def date():
    run("date")
    print "Date on {}".format(env.host)
```

sudo()

```
from fabric.api import task, env, sudo

env.use_ssh_config = True

@task
def chuser():
    sudo("id")
```

put()

```
from fabric.api import task, env, put, local, run

env.use_ssh_config = True

@task
def putit():
    local("touch test.txt")
    put("test.txt", "/tmp")
    run("ls -lrt /tmp | tail -n 1")
```

get()

```
from fabric.api import task, env, get, local, run, sudo

env.use_ssh_config = True

@task
def getit():
    sudo("cp /tmp/SECRETSTUFF.txt /tmp/NOTSOSECRETSTUFF.txt")
    sudo("chown sean /tmp/NOTSOSECRETSTUFF.txt")
    get("/tmp/NOTSOSECRETSTUFF.txt", ".")
    local("cat NOTSOSECRETSTUFF.txt")
```

Context Managers

- with cd(“/some/path”) - cd to a remote directory before executing the command
- with lcd(“/some/path”) - cd to a local directory before executing the command
- with warn_only() - Don’t error out on non-0 exit codes
  - like try, except sorta
- with prefix(“workon myVirtEnv”) - prefix all commands with the prefix defined
- with path(“/some/path”) - add path to your shell’s PATH

with cd()

```
from fabric.api import task, env, put, run
from fabric.context_managers import cd

env.use_ssh_config = True

@task
def cdit():
    with cd("/home/sean"):
        run("touch test.txt")
        run("ls -lrt | tail -n 1")
```

with warn_only()

```
from fabric.api import task, env, put, run, local
from fabric.context_managers import warn_only

env.use_ssh_config = True

@task
def fail():
    run("cat /etc/passwd-")
    local("date")

@task
def win():
    with warn_only():
        run("cat /etc/passwd-")
    local("date")
```

with warn_prefix()

```
from fabric.api import task, env, put, run
from fabric.context_managers import prefix

env.use_ssh_config = True
env.venv_wrapper = "/usr/local/bin/virtualenvwrapper.sh"

@task
def requirements(venv="socpug"):
    with prefix(". {}; workon {}".format(env.venv_wrapper, venv)):
        run("pip freeze")
```

Context Managers

- You can combine multiple context managers by using the syntax `with cm1(), cm2(), cm3()…`
- Allows for less nested code
  - Flat is better than nested
- Easier to read - see point above

More than 1 with

```
from fabric.api import task, env, put, run
from fabric.context_managers import cd, prefix

env.use_ssh_config = True

@task
def multi_with():
    with cd("/home/sean"), prefix("sleep 2"):
        run("ls -lrt | tail -n 1")
        run("ls -lrt | tail -n 1")
```

State management - env

- from fabric.api import env
- env is just a dictionary so you can use it to setup variables you want shared between tasks.
- env changes in one task will persist to the next task
- use `with settings()` to change env settings for a given command
- exists for historical reasons, and today you can use module level variables, but in the future fabric is aiming to be thread-safe and env might be the only safe way of sharing state between threads.

Lots of options!

- abort_exception
- abort_on_prompts
- all_hosts
- always_use_pty
- colorize_errors
- combine_stderr
- command
- command_prefixes
- command_timeout
- connection_attempts
- cwd
- dedupe_hosts
- disable_known_hosts
- eagerly_disconnect
- effective_roles
- exclude_hosts
- fabfile
- gateway
- host_string
- forward_agent
- host
- hosts
- keepalive
- key
- key_filename
- linewise
- local_user
- no_agent
- no_keys
- parallel
- password
- passwords
- path
- pool_size
- prompts
- port
- real_fabfile
- remote_interrupt
- rcfile
- reject_unknown_hosts
- system_known_hosts
- roledefs
- roles
- shell
- skip_bad_hosts
- ssh_config_path
- ok_ret_codes
- sudo_prefix
- sudo_prompt
- sudo_user
- tasks
- timeout
- use_shell
- use_ssh_config
- user
- version
- warn_only

Really just SSH

- Thanks to Paramiko
  - It is a pure Python interface around SSH networking concepts
- Fabric is really just using ssh under the hood, along with subprocesses
- `env.use_ssh_config = True` to use your ~/.ssh/config
- command line options for setting user -u, and identity key -i

Make output nice

- use colors
  - from fabric.colors import green
  - print(green(“Way to go”))
- use with_quiet()
  - only *don’t* use it when you actually care about the output - like `git pull`
  - unfortunately no global way to turn this on / off :(
- set env[“colorize_errors”] = True to have errors colored to make them more noticeable

Parallel execution

- from fabric.api import parallel
- parallel is just a decorator @parallel
- runs each command on N hosts in parallel but the commands on a single host are run synchronously
- use env[‘parallel’] = True to make all tasks run in parallel

How about this suck!

- I want to add a line to a file, but only if it does not already exist - in bash

```
if grep -xq "something" foo.txt
then
  echo "already there"
else
  echo "something" >> foo.txt
fi
```

How about this more suck!

- That sucks! What if we have do do it via sudo… that gets really messy because redirecting the output.

```
if sudo grep -xq "something" foo.txt
then
  echo "already there"
else
  printf "something" | sudo su root -c "\"cat > foo.txt\" | sudo bash"
fi
```

How about this more suck!

- That sucks! What
  redirecting the ou

```
if sudo grep -
then
  echo "alread
else
  printf "some
fi
```

essy because

```
:t\" | sudo bash"
```

How about this awesome!

- I want to add a line to a file, but only if it does not already exist - with Fabric!

```
from fabric.contrib import files
files.append("foo.txt", "something", use_sudo=True)
```

How about this awesome!

Some more awesome!

- files.upload_template
- like put() but uses a template instead of a raw file
- has option to use jinja2 templates
- use things like env[‘host’] to populate the context for the template

So we have come a long way

Time for the killer demo…

Remember at the beginning I started an AWS instance…

and I ran some command on it…

Let’s setup a website from a base linux box.

Back already!?!!!

Man that was slick. Props Sean.

We’re Hiring!

- Great pay, benefits, stock options, and bonus plan
- Top notch developers
- Challenging problems
- Large scale distributed cloud architecture
- We use Python
  - along with Scala, Go and C++
- “Hilarious quotes” wiki page from IRC logs

crowdstrike.com/about-us/careers/