Software Testing Methodology

Lecture 9

Gregory S. DeLozier, Ph.D.

Reasons For Failure

- Wrong requirements
- Defective code
- Defective tools

- Problem in execution environment
  - This probably is the most common reason for outage
  - Really expensive
  - Sophisticate software architectures are making it worse

Reasons for Environmental Failure

- Missing dependencies
  - Libraries
  - Tools
  - Volumes
  - Ports
  - Connections
  - Credentials

Reasons for Environmental Failure

- Lack of capacity
  - Memory
  - Disk space
  - CPU speed
  - Network Bandwidth
  - Parallel capacity

Reasons for Environmental Failure

- Changes in the environment
  - Additional activity
  - Capacity consumption
  - Connectivity issues
  - Patches
  - Interference – side effects from other work
  - Malware
  - Credentialing issues
  - Power failure

How does this relate to testing?

- Testing the environmental requirements
- Repeating that test on the target environments
- Periodically verifying the target environment
- Environmental assessment in case of failure

Environment as Software

- Tools have been made to _create_ environments.
  - Batch files
  - Recipes
  - Containers

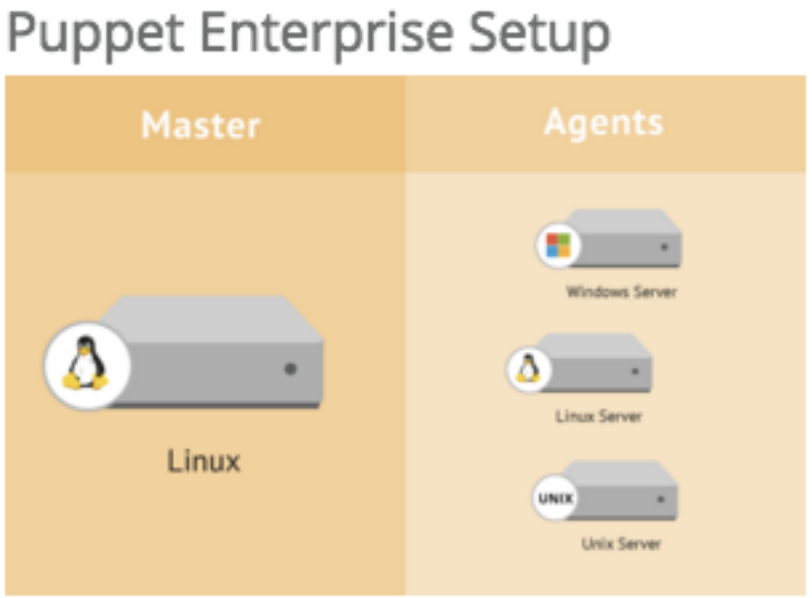
- These can be adapted to testing work
  - Test the correct execution of the setup program
  - Compare recipes to current state
  - Evaluate container construction _and_ state

Recipes

- Create a tool to describe desired state
- Put an agent on a machine, tell it which recipe to follow
- Agent updates machine to match recipe
- Recipe conformance is testable

Puppet

- https://puppetlabs.com/
- https://puppetlabs.com/puppet/what-is-puppet



Catalogs

- Puppet configures systems in two main stages:
  - Compile a catalog
  - Apply the catalog
- What is a Catalog?
  - A catalog is a document that describes the desired system state for one specific computer. It lists all of the resources that need to be managed, as well as any dependencies between those resources.

Example Manifest

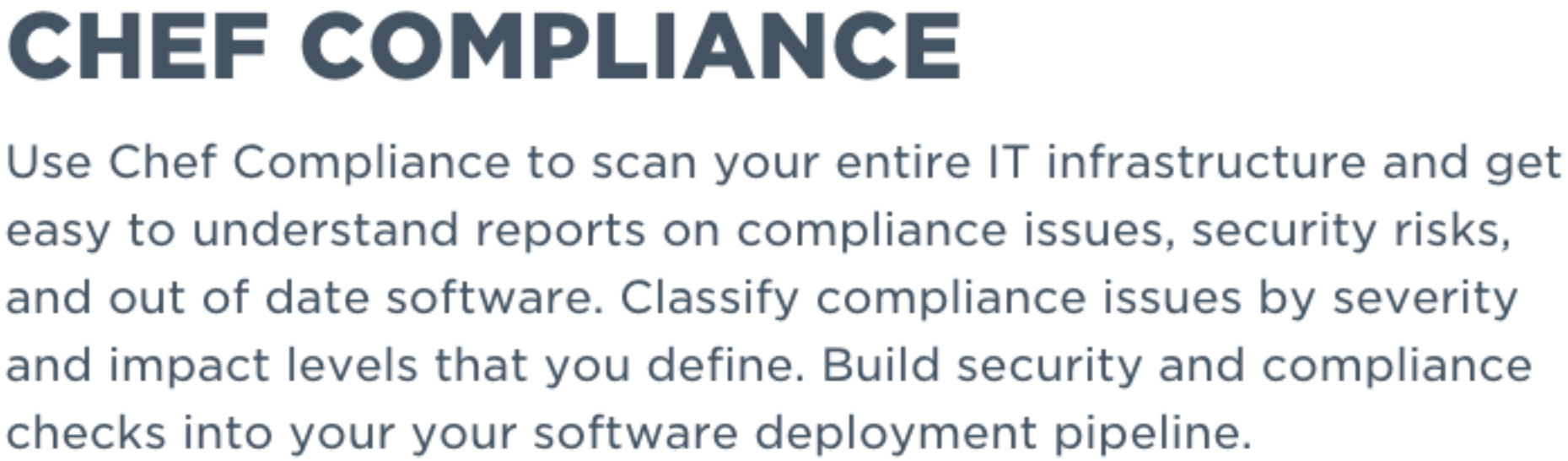
```
case $operatingsystem {
  centos, redhat: { $service_name = 'ntpd' }
  debian, ubuntu: { $service_name = 'ntp' }
}

package { 'ntp':
  ensure => installed,
}

service { 'ntp':
  name      => $service_name,
  ensure    => running,
  enable    => true,
  subscribe => File['ntp.conf'],
}

file { 'ntp.conf':
  path    => '/etc/ntp.conf',
  ensure  => file,
  require => Package['ntp'],
  source  => "puppet:///modules/ntp/ntp.conf",
  # This source file would be located on the Puppet master at
  # /etc/puppetlabs/code/modules/ntp/files/ntp.conf
}
```

Some fun stuff

- https://www.youtube.com/watch?v=3MjioGNw_rY

Chef – Alternative to Puppet

- This is where the “recipe” term comes from
- Scanning for compliance is a large draw here:



Chef is an alternative to Puppet

- https://learn.chef.io/
- It turns out we can play with this one

- < Tutorial Time >

- https://learn.chef.io/learn-the-basics/ubuntu/configure-a-resource/

Ansible – a recent addition

- http://www.ansible.com/
- Modules define actions to be taken
- Playbooks contain information about desired state

Ansible Modules

- Small bits of code that do useful things

```
#!/usr/bin/python

import datetime
import json

date = str(datetime.datetime.now())
print json.dumps({
    "time" : date
})
```

- More examples:
  - http://docs.ansible.com/ansible/developing_modules.html

Ansible Playbooks

```
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Ansible – video tour

- https://ansible.wistia.com/medias/qrqfj371b6

Programmatic Solutions

- Remote control of machines
  - Write function in, say, python
  - Part of the function is executed remotely

- These allow you to program remote machines
  - Functions to set things up – configuration
  - Functions to return values – testing

- Paramiko – remote control
- Fabric – convenient configuration
- Yarn – alternative for Python 3.x

Paramiko

- www.paramiko.org
- Implements SSH for remote login
- Very low-level
- http://jessenoller.com/blog/2009/02/05/ssh-programming-with-paramiko-completely-different

Paramiko Example

- import paramiko
- ssh = paramiko.SSHClient()
- ssh.connect('127.0.0.1', username='jesse', password='lol')

- ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy()) <- add

- stdin, stdout, stderr = ssh.exec_command("uptime")
- type(stdin)
- stdout.readlines()

  ['13:35 up 11 days, 3:13, 4 users, load averages: 0.14 0.18 0.16\n']

More Paramiko

```
ssh.connect('127.0.0.1', username='jesse',
    password='lol')
stdin, stdout, stderr = ssh.exec_command(
    "sudo dmesg")
stdin.write('lol\n')
stdin.flush()
data = stdout.read.splitlines()
for line in data:
    if line.split(':')[0] == 'AirPort':
        print line
```

Fabric – a Pythonic remote solution

```
from fabric.api import run

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

If you save the above as `fabfile.py` (the default module that `fab` loads), you can run the tasks defined in it on one or more servers, like so:

```
$ fab -H localhost,linuxbox host_type
[localhost] run: uname -s
[localhost] out: Darwin
[linuxbox] run: uname -s
[linuxbox] out: Linux

Done.
Disconnecting from localhost... done.
Disconnecting from linuxbox... done.
```

Fabric Issues

- http://www.fabfile.org/
- Author is not porting
- Ports are available
  - Google “fabric3” “python”
  - $pip (or pip3) install fabric3
  - Use same as fabric

Test Driven Infrastructure

- Design a test to verify some aspect of infrastructure
- Watch the test fail
- Fix the problem
  - Probably a dependency
  - Ideally, with automation
- Run the test again

Aside: the “private.py” file

- Use a private credentials file that doesn’t check in

```
user=”myusername"
password=”MyPa$$w0rd"
```

- Add “private.py” to .gitignore
- Then use this to get user and password

```
from private import user, password
```

Basic Example of a Remote Query

```
from remote_api import env,cd,run
from private import user, password
env.host_string = 'ssh.pythonanywhere.com'

env.user = user
env.password = password
with cd("~"):
    print(run("ls -l"))
```

Basic Plan

- Write Unit Tests to Verify Requirements
  - These will eventually become regression tests for the environment
- Write Remote Methods to Set Up Environment
  - These will make the tests pass
  - These will become the basis for automated deployment