# Testing Puppet code syntax with puppet-lint

command line usage of puppet-lint,

how to put ignore's in the files for things you can't control.

class test {

notify {'hi' }

notify {"hi" } #lint:ignore:double\_quoted\_strings

}

also puppet parser validate

but these in a git pre-commit hook

# Writing unit tests using rspec-puppet

– We will learn to install rspec-puppet and write unit tests  
First, you need to install rspec-puppet, this will bring with it a few dependencies, I usually use gem to install this.

Then you'll need a few extras,

puppetlabs\_spec\_helper

metadata-json-lint

rubocop

Next, either create your module with puppet module generate, or create it, then run rspec-puppet-init.

This makes an empty shell for you to add your tests.

the testing language is fairly straightforward, simple syntax

case $::osfamily {

'RedHat': { notify {'linux':} }

'windows': { notify {'windows':}}

default: { notify {'something':}}

}

**make the test**

require 'spec\_helper'

describe 'rspec\_test' do

context 'with default values for all parameters' do

let(:facts) { {:osfamily => 'Debian' } }

it { should contain\_class('rspec\_test') }

it { should contain\_notify('something') }

end

end

# Writing integration tests using serverspec

serverspec is similar to rspec but is run against real machines instead of a catalog.

You test whether or not services are running, or if ports are listening, instead of asking if the resource is in the catalog.

The idea is that you apply puppet to a node, then run serverspec against it to make sure it's correct.

Let's start with a simple example, we want our mpli test node to be running apache and listening on port 80, we make a simple puppet manifest to do that.

package {'httpd':

ensure => true,

}

service {'httpd':

ensure => true,

enable => true,

}

Now use serverspec-init to configure your spec directories

we'll say use our vagrantfile so it creates the tests as we need them

then run rake spec at the end

I cheated and used the apache test, which is in the sample, lets do something else and make sure that works too.

So install something else, make sure it's port is listening

call it a day and move onto the next one.

# Writing acceptance tests using Beaker

We will learn to install Beaker and write acceptance tests  
  
for our Puppet infrastructure for MPLI Productions.  
show the beaker home page, talk about what it is:

it's a framework. it's like a layer above serverspec, you can use it to provision multiple machines and then run tests to make sure that the machines can communicate with each other. It can get complex, I suggest looking at the howto documents as a start.

make a simple provisoin file, provision.rb

then a simple test.rb file,

a host configuration file.

run beaker

when you start, use the preserve option, show how you can go into that directory and look at the vagrantfile.

beaker --pre-suite provision.rb --tests test.rb test2.rb --host centos7-64ma.yaml --preserve-hosts always

find .vagrant

vagrant status

vagrant ssh

check our service is still running

# Bringing it together with Test Kitchen

We will learn to install Test Kitchen and bring all of the  
above mentioned tests together so that they can be run as a suite.  
test-kitchen is the chef version of beaker...but it supports both chef and puppet.

indeed, even when you specify the puppet provisioner, you get chef installed.

The tool is a little more polished than beaker and if you are already familiar with it, then using the puppet provisioner is a simple change.

the nice thing about kitchen is that is uses the busser framework to execute tests. busser recognises your testing framework by directory name. So you can use things like bats, cucumber, minitest, serverspec and rspec.

ok,

install bundle

bundle init

add to gemfile

gem "test-kitchen"

gem "kitchen-puppet"

bundle install again

kitchen init --provisioner puppet\_apply

if anything was appended to your gemfile, you'll have to bundle install again.

edit the kitchen.yml

---

driver:

name: vagrant

provisioner:

name: puppet-apply

manifests\_path: manifests

modules\_path: modules

hiera\_data\_path: hieradata

platforms:

- name: ubuntu-14.04

- name: centos-7.2

suites:

- name: default

manifest: site.pp

kitchen create

Create Puppetfile

r10k puppetfile check

r10k puppetfile install

kitchen converge

# Building the code on Jenkins CI

We will learn to use the tests that we have written and build the  
  
Puppet code on Jenkins to be merged to the master branch of the repository.  
using either beaker or test-kitchen, you want jenkins to do the steps that you would do in your testing, but do them automatically.

I'll show how to do it for test-kitchen

Add all the relevant files to your git repo.

You need the Gemfile, Puppetfile, .kitchen.yaml files, add things to .gitignore that you don't want.

Now configure jenkins with the git plugin, move your git repo to some place where jenkins can get the repo. You can use the ssh plugin as well here, but we'll show a local repo for simplicity.

The idea is that jenkins will clone your repo down and perform the steps you outline.

For test-kitchen, using bundler this is pretty simple.

Show the configuration, it's just bundle install, bundle puts things in the local ~/bin directory, just add that to the path then run kitchen test.

Wait for the results

profit.

next thing you would do is have your git pre-receive hook run the jenkins test before allowing a merge to master.

You can do via a shared key, show where that is done in the interface.

# Deploying code to the Puppet master

We will learn to deploy tested code to the Puppet masters.