Tutorial

Last updated: 14 July 2020

Why should you test your Puppet modules?

At first glance, writing tests for your Puppet modules appears to be no more than simply duplicating your manifests in a different language and, for basic “package/file/service” modules, it is.

However, when you start leveling up your modules to include dynamic content from templates, support multiple operating systems or take different actions when passed parameters, these tests become invaluable when adding new functionality to your modules, protecting against regressions when refactoring or upgrading to a new Puppet release.

What should you be testing?

There are a lot of people confused by the purpose of these tests as they can’t test the result of the manifest on a live system. That is not the point of rspec-puppet.

Rspec-puppet tests are there to test the behaviour of Puppet when it compiles your manifests into a catalogue of Puppet resources. For example, you might want to test that your `apache::vhost` defined type creates a `file` resource with a `path` of `/etc/apache2/sites-available/foo` when run on a Debian host.

When writing your test cases, you should only test the first level of resources in your manifest. By this I mean, when testing your ‘webserver’ role class, you would test for the existence of the `apache::vhost` types, but not for the `file` resources created by them, that’s the job of the tests for `apache::vhost` .

Basic structure of a test file

Whether you’re testing classes, defined types, hosts or functions the structure of your test file is always the same.

```
require 'spec_helper'

describe '<name of the thing being tested>' do
  # Your tests go in here
end
```

The important thing is what you name your test file and where you put it. Test files should always end in `_spec.rb` (generally, they're named `<thing being tested>_spec.rb`). Class tests should be placed in `spec/classes`, defined type tests should go in `spec/defines`, host tests should be placed in `spec/hosts` and function tests should go in `spec/functions`.

Writing your first test cases

This is not intended to be an RSpec tutorial, just an explanation of how to use the extended functionality that rspec-puppet provides. If you are not familiar with the basics of RSpec, I highly recommend you take some time before continuing to read through the RSpec documentation.

Lets say you're writing tests for a `logrotate::rule` type that does two things:

1. Includes the `logrotate::setup` class which handles installing logrotate
2. A `file` resource that drops your logrotate rule into `/etc/logrotate.d`

First off, lets create a skeleton spec file for your defined type (`modules/logrotate/spec/defines/rule_spec.rb`)

```
require 'spec_helper'

describe 'logrotate::rule' do

end
```

As this is a defined type, the first thing we need to do is give it a title (the string after the `{` in your manifests).

```
  let(:title) { 'nginx' }
```

Now, lets test that we're including that `logrotate::setup` class

```
  it { is_expected.to contain_class('logrotate::setup') }
```

Remember, we don't want to test what `logrotate::setup` does, we'll leave that to the test cases you're going to be writing for that class.

At this point, your spec file should look like this

```
require 'spec_helper'

describe 'logrotate::rule' do
  let(:title) { 'nginx' }

  it { is_expected.to contain_class('logrotate::setup') }
end
```

OK, on to dealing with that `file` resource, lets use the title of the `logrotate::rule` resource as the name of the file you're dropping into `/etc/logrotate.d/` .

```
  it { is_expected.to contain_file('/etc/logrotate.d/nginx') }
```

As it currently stands, this test is pretty useless as it doesn't actually check anything about the file. We can check values of the parameters passed to the file resource by chaining the `with` method onto our test and passing it a hash of expected parameters and values. Lets say we want to set some sane values: present, owned by root and read only:

```
  it do
    is_expected.to contain_file('/etc/logrotate.d/nginx').with({
      'ensure' => 'present',
      'owner'  => 'root',
      'group'  => 'root',
      'mode'   => '0444',
    })
  end
```

You should now have a spec file that looks like this

```
require 'spec_helper'

describe 'logrotate::rule' do
  let(:title) { 'nginx' }

  it { is_expected.to contain_class('logrotate::setup') }

  it do
    is_expected.to contain_file('/etc/logrotate.d/nginx').with({
      'ensure' => 'present',
      'owner'  => 'root',
      'group'  => 'root',
      'mode'   => '0444',
    })
  end
end
```

What about the most important part of the file, its contents? Before we get to that, we're going to make your type take a boolean parameter called `compress`. If this value is `true`, a line containing `compress` should exist in the file. If this value is `false`, a line containing `nocompress` should exist in the file.

```
  context 'with compress => true' do
    let(:params) { {'compress' => true} }

    it do
      is_expected.to contain_file('/etc/logrotate.d/nginx') \
        .with_content(/^\s*compress$/)
    end
  end

  context 'with compress => false' do
    let(:params) { {'compress' => false} }

    it do
      is_expected.to contain_file('/etc/logrotate.d/nginx') \
        .with_content(/^\s*nocompress$/)
    end
  end
```

You'll note that we're now specifying the parameters that should be sent to our `logrotate::rule` type by setting params to a hash. Similarly, you can also specify the value of facts by using `let(:facts) { }`. The other thing we did was chain a `with_content` method onto our test and passed it a regex that the value should match. You can do this with any other parameter as well, eg `with_ensure`, `with_owner`, `with_foobarbaz`.

As our type can only handle two possible values for `compress`, let's be nice and make sure that compilation will fail if someone passes something else to it.

```
  context 'with compress => foo' do
    let(:params) { {'compress' => 'foo'} }

    it { is_expected.to compile.and_raise_error(/compress must be true or false/) }
  end
```

The final version of your spec file should be:

```
require 'spec_helper'

describe 'logrotate::rule' do
  let(:title) { 'nginx' }

  it { is_expected.to contain_class('logrotate::setup') }

  it do
    is_expected.to contain_file('/etc/logrotate.d/nginx').with({
      'ensure' => 'present',
      'owner'  => 'root',
      'group'  => 'root',
      'mode'   => '0444',
    })
  end

  context 'with compress => true' do
    let(:params) { {'compress' => true} }

    it do
      is_expected.to contain_file('/etc/logrotate.d/nginx') \
        .with_content(/^\s*compress$/)
    end
  end
```

```
  context 'with compress => false' do
    let(:params) { {'compress' => false} }

    it do
      is_expected.to contain_file('/etc/logrotate.d/nginx') \
        .with_content(/^\s*nocompress$/)
    end
  end

  context 'with compress => foo' do
    let(:params) { {'compress' => 'foo'} }

    it do
      expect {
        is_expected.to contain_file('/etc/logrotate.d/nginx')
      }.to raise_error(Puppet::Error, /compress must be true or false/)
    end
  end
end
```

Congratulations, you've just written a set of tests for a defined type without writing a single line of Puppet code. You should now head over to the documentation to learn more.

Now go write the manifests needed to make these tests pass!