Enjoying the Journey from Puppet 3.x to 4.x

(jfdi)

Rob Nelson
Vox Pupuli Contributor
@rnelson0, http://rnelson0.com

What are we talking about today?

- Upgrade our Puppet master(s) and agents to Puppet 4.x
- Refactor our code base for Puppet 4, remove Puppet 2- and 3-isms.
- Not a linear progression, we may hit each step multiple times.

Who does this apply to?

- Puppet Enterprise Users
- Puppet Opensource Users
- Masterful and Masterless setups

This is admittedly focused on a masterful setup, but if you have a masterless setup, a majority will still apply. We'll try and point it out as we go, and feel free to ask questions or point things out as we go.

Why?

- Puppet 4 is new!
- Or: Puppet 3 is old! EOS on December 31, 2016. When the last PE 3.x is EOS, Opensource will be EOS as well.
- Lots of new things in the Puppet 4 language (nee Future Parser) to take advantage of. Iteration, typed variables, etc.
- Application Orchestration!
- Modules are starting to only support Puppet 4.
- Let Ruby 1.8.7 and Puppet w/Passenger go!
- Don't slow down the DevOps. Stairstep upgrades get harder and harder all the time. Stay current, it makes each upgrade minutes long instead of hours, days, or weeks.

Puppet 3 won't turn into a pumpkin on 1/1/2017, but you may find yourself stuck between a rock and a hard place when you need a newer version of a module that requires Puppet 4, probably because it is taking advantage of the language.

Puppetserver is the future and Ruby moves fast, take advantage of the AIO builds when possible.

Blueprint

What's the high level plan to get from here to there?

- Start with Puppet 3.x
- Read the release notes
- Stairstep Roadmap
- Validate / Create rspec tests
- Refactor until your current version passes all the tests
- Snapshot and Upgrade the Master
- Upgrade the Agents
- Repeat the steps for each stairstep until you are get to 4.current

---

If you're on Puppet 2.x, you need to get to 3.x first! But that's not this talk. There were some good talks about this at PuppetConf 2015.

Release Notes

- All of them - not just the latest version
- If you skipped a dozen major/minor/patch versions, you have lots to read and collate. If that sounds horrible, that's because it is!
- Stay up to date - less reading and resolving conflicting notes.

Stairstep Roadmap

Varies depending on your starting point. Find the minimum steps; you can usually skip some MINOR releases.

Enable the Future Parser before you hit 4.0.

For example, the steps between 3.6.x and 4.5.x:

- 3.6.x -> 3.8.x
- 3.8.x w/Future Parser
- 3.8.x -> 4.0.0
- 4.0.0 -> 4.latest

Validate / Create rspec tests

- No tests, no assured behavior!
- Generate modules with puppet-module-skeleton and you get a free rspec setup, too.
- Never written an rspec-puppet test? puppet-retrospec can help! Generates naive tests that need tuned, but a great place to start.
- Tons of blog posts on testing.
- Make sure your existing tests pass before changing the code.
- Turn on Future Parser and Strict Variables as soon as you hit 3.8.x
    - You can do it earlier, but some versions have bugs.

Tests do not guarantee success, but can identify many regressions and determine when failure is guaranteed.

Rspec testing can be tricky, no lie. But it doesn't have to be. Start simple and grow from there. It's worth the effort.

Example rspec tests

```
$ cat spec/classes/apache_spec.rb
require 'spec_helper'
describe 'profile::apache', :type => :class do
  let :facts do
    {
      facts_hash
    }
  end

  context 'with defaults for all parameters' do
    it { is_expected.to create_class('profile::apache') }
    it { is_expected.to contain_package('httpd') }
    it { is_expected.to contain_user("apache") }
  end
end

[rnelson0@build03 profile:production]$ bundle exec rspec spec/classes/apache_spec.rb

profile::apache
  with defaults for all parameters
    should contain Class[profile::apache]
    should contain Package[httpd]
    should contain User[apache]

Finished in 7.82 seconds (files took 2.49 seconds to load)
3 examples, 0 failures
```

Refactor

- Create a new branch against for the next stairstep version, e.g. `3.8.6`.
- Test against this target version in addition to your current version, e.g. `~>3.6` and `~>3.8`.
- Identify failing tests, refactor as needed.
- Upgrade modules as early as possible. Be aware of the required Puppet version for a module version, and look out for defunct or migrated modules, such as those transferred to Vox Pupuli.
- Move forward when tests are green for current and next version.

Testing with particular Puppet versions

```
$ grep PUPPET Gemfile
  gem "puppet", ENV['PUPPET_GEM_VERSION'] || '~> 4.0'

[rnelson0@build controlrepo]$ export PUPPET_GEM_VERSION='~>3.8'; bundle update
Installing puppet 3.8.7 (was 4.6.0)
[rnelson0@build controlrepo]$ bundle exec puppet --version
3.8.7

[rnelson0@build controlrepo]$ export PUPPET_GEM_VERSION='3.8.1'; bundle update
Installing puppet 3.8.1 (was 3.8.7)
[rnelson0@build controlrepo]$ bundle exec puppet --version
3.8.1
```

Remember to export the variable, or you'll have to prepend PUPPET_GEM_VERSION on every usage of bundle.

Snapshot and Upgrade the Master

Not all upgrades will go well. Make sure we have a way back!

- Snapshot (or equivalent) the master, and any canary nodes we have.
- Restrict access to the master. Don't push bad catalogs to nodes, could cause outages.
  - Block tcp/8140 with a firewall.
  - Revoke certificates for non-canary nodes (roll the snapshot back to revert).
  - Revoke the CA, generate a new CA and new agent nodes.
  - Disable puppet agent on nodes with orchestration, ensure it doesn't turn back on in the middle of your upgrade!
- There's no right way, just find a way that works well for you.
- Upgrade the master.
- Test the master against itself, `puppet agent -t`.
- Test the canary nodes with `puppet agent -t` as well.
- If anything fails, collect logs, revert to your snapshots, unblock connection to the master, and go back to the *Refactor* steps to fix it before trying again.
- Optional: Revert to snapshot, remove the block, and upgrade the master again without the block - only when you're ready to move forward.
- There's no right way, just a way that works for you!

Upgrade the Agents

A very non-complete list of methods:

- puppetlabs/puppet_agent (requirements) allows you to update agents on their next checkin.
- MCollective or other orchestration.
- Replace nodes with new instances running the newer agent.
- By hand.
- Some combination of methods.
- You can skip this step on PATCH versions and some MINOR versions

Puppet Agent in action

```
[rnelson0@dns ~]$ puppet --version
3.8.7
[rnelson0@dns ~]$ sudo puppet agent -t --environment=puppet_agent
<run output>
-bash: /usr/bin/puppet: No such file or directory
[rnelson0@dns ~]$ /opt/puppetlabs/puppet/bin/puppet --version
4.5.2
```

There are known issues, such as leaving other puppetlabs repos in place (MODULES-3805 and 3806), but hard to argue with a puppetized solution!

Puppet 4's binary is in a new location, which makes it easy to know when the upgrade works properly!

Repeat!

- After you're done with one upgrade, start working on the next!
- Repeat the Refactor / Snapshot / Upgrade steps only till you hit your target version.

Keep Up

- Once you're done, you're not done! Start refactoring to take advantage of Puppet 4 language improvements including, but not limited to:
  - Replace `create_resources()` with iteration.
  - Replace `validate_*()` with data types.
  - Simplified resource wrappers with * and + operators.
  - Improved per-expression default attributes.
- PE has quarterly upgrades, POSS has more frequent updates.
  - Try not to get more than 2 MINORs behind.
  - Anticipate new versions by changing your Gemfile/rspec-tests to test against puppet version `~>4.0` (latest 4.x) and run `bundle update` before manual tests. Your test setup won't be surprised when `v4.next` is released.
- Aim for master upgrade times of <1h - it's possible!

Tricks, Part One

Some lessons learned from POSS/PE upgrades...

- PE includes support. Get some help planning it out.
- PE Classifier - You can expect some changes as you move forward. Review the Preconfigured Node Groups documentation yourself, engage support if you run into issues.
- Bundled Ruby - If you run an older distro (*cough* EL6 *cough*) and want to get away from Ruby 1.8.7, it is possible to use the PE or AIO Ruby. You can use it in a pinch, but I highly recommend you use rbenv/rvm instead, or update to something with a newer Ruby on the system (EL7 at least has Ruby 2.0!). PUP-6106
- `pe_puppetserver_gem` is out, `puppetserver_gem` is in.
- The hiera eyaml gem will be removed during the upgrade to the 4.0 puppetserver - more accurately, the new puppetserver environment will not have it present.
  - If you only have eyaml configured, enable the yaml backend as well and ensure that the master itself does not rely on any eyaml-only data.
  - The first agent run on the master can redeploy the gem (with puppet/hiera or similar) before an agent connects that requires eyaml data.
  - This does not recur when you upgrade within the 4.x chain.

I know many of us don't like to engage support, but if you don't use it for updates, when will you use it?

Tricks, Part Two

- Beware string conversion, in puppet DSL and hiera data. Understand how it works.
  - `'undef'` in rspec-puppet represents an undefined value.
  - In Puppet DSL, it's the word undef! Use `undef`, without quotes, instead.
  - If you have a file resource with a title or path of `${undefvar}/${populatedvar}`, rspec will start failing because `file { 'undef/etc/app.conf' :}` is not valid.
  - `'true'` vs `true` and `'false'` vs `false` is another likely candidate.
  - Keep this in mind if tests pass but agent runs fail.
- Hiera scope
  - `%{}` in 3.x and in 4.5 resolves to the empty string. This is often used to prevent variable interpolation, as in `%%{}{environment}` to generate the string `%{environment}`. There was a regression in between those versions and it started returning the scope, giving strings like `%<#Hiera:7329A802#>{environment}`. Use `%{::}` instead, as in `%%{::}{environment}`
  - Additionally, some versions expect `::` prepends to variables and others don't. This may affect your `datadir` value in `hiera.yaml`. Change `%{environment}` to `%{::environment}`.

Tricks, Part Three

- Review modules and their supported versions. Some may be incorrect, or have loose assumptions (`>= 3` but not `< 4`).
  - When you upgrade across major versions during refactoring, expect additional troubleshooting time and effort.
  - Generally you want to do this early, but if a particular module gives you problems, consider waiting until the master upgrades are done first.
  - If the latest module reports version `999.999.999`, it is probably defunct. Check out the readme for more information!
  - There are quite a few tools to assist with automating version upgrades in your Puppetfile.
- Don't change too much at once if you can help it. The more variables you change at once, the more difficult troubleshooting is. Try not to change your fleet's distro version during the upgrade unless you really like chasing down issues.
- There is a new Puppet Enterprise Upgrade Service offering now! Especially helpful for PE customers on older version.

Tools

The tools are constantly improving, so if you haven't looked in a while, check it out to see if your concerns were addressed.

- puppetlabs_spec_helper, rspec-puppet, and puppet-lint are all improving their support for Puppet 4 on a regular basis.
- There are a number of catalog diff tools (including the diffs and a viewer) to inspect the differences between catalogs received from different versions of Puppet.
- puppet-ghostbuster helps you find "dead code" that you may want to prune before you start on your refactoring journey.

Links

Additional information on Puppet 4 and Migrations

- Whirlwind Tour of Puppet 4 by the poet, R.I. Pienaar
- Puppet - our journey from Puppet 3.8 to Puppet 4

Q&A

Questions?

I would also like to encourage those in the audience to share their knowledge!