Case Study:

Puppet 3 to Puppet 4

by Ryan Whitehurst

Source: https://github.com/thrnio/puppet3-to-puppet4

My name is Ryan Whitehurst, and this is a case study on how we at Puppet Labs moved our internal puppet infrastructure from Puppet 3 to Puppet 4.

This presentation is available on my GitHub page. There's the source code for this reveal.js presentation, as well as PDF copies both with and without speaker notes. I'll show the link again at the end as well.

Ryan Whitehurst

SysOps Engineer, Puppet Labs

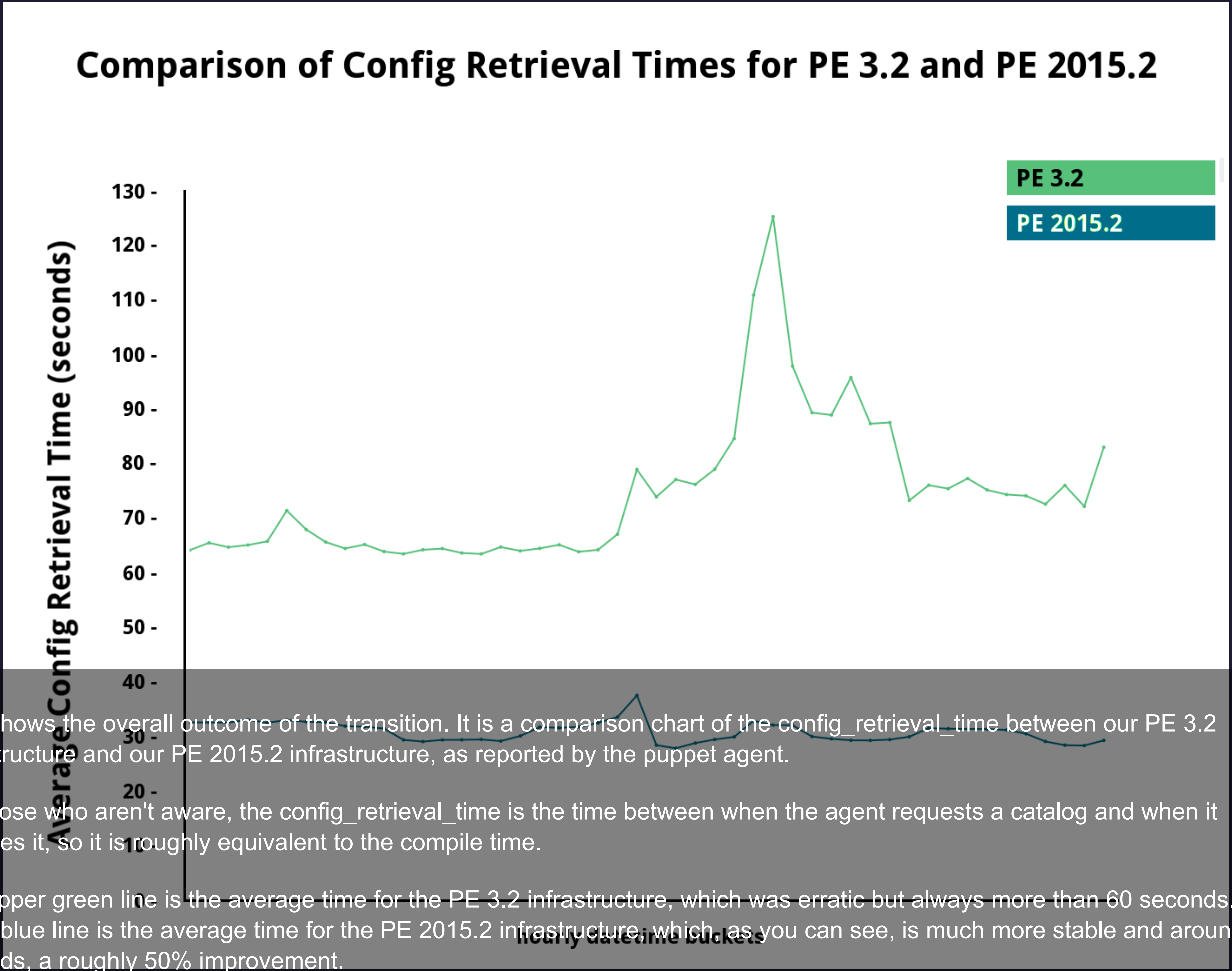
Email	rw@puppetlabs.com
IRC	thrnio
Twitter	@thrnio

The Plan

1. Overview
2. Building a test environment
3. Testing catalogs
4. Performance
5. The migration

Our infrastructure

- ~500 nodes
- ~50 environments
- ~150 public modules
- ~40k lines of custom puppet code per environment
- ~128k total lines of puppet code per environment



This shows the overall outcome of the transition. It is a comparison chart of the config_retrieval_time between our PE 3.2 infrastructure and our PE 2015.2 infrastructure, as reported by the puppet agent.

For those who aren't aware, the config_retrieval_time is the time between when the agent requests a catalog and when it receives it, so it is roughly equivalent to the compile time.

The upper green line is the average time for the PE 3.2 infrastructure, which was erratic but always more than 60 seconds. The lower blue line is the average time for the PE 2015.2 infrastructure, which, as you can see, is much more stable and around 30 seconds, a roughly 50% improvement.

What we did

- PE 3.2 to PE 2015.2
- Puppet 3.4 to Puppet 4.2
- PuppetDB 1.x to PuppetDB 3.x
- Split install to monolithic master of masters
- Passenger to Puppet Server
- 3x parser to 4x ("future") parser
- Switch to directory environments

We made a fairly large jump when we upgraded. There were a number of reasons for this, mostly relating to obscure edge cases we couldn't reliably reproduce that prevented us from running earlier versions.

We weren't strictly running the 3x parser. We were actually running an early, incomplete version of the future parser. It supported some of the puppet 4 features, but not all of them, and some had changed. This made the upgrade more complicated in many ways.

The Plan

1. Overview
2. Building a test environment
3. Testing catalogs
4. Performance
5. The migration

The first step to upgrading puppet for us was to build a test environment. We didn't have an existing test environment for our puppet infrastructure, and we couldn't reliably reproduce our existing environment, so we decided to deploy from scratch.

First attempts

Puppet Server crashes... sometimes

Our first attempts to deploy a new version of puppet (at the time, PE 3.7) failed. Puppet Server would crash after a short period of time when we tried to use it, but when we attempted to reproduce it for the engineering team, we couldn't.

Our solution:
Automate ALL THE THINGS

...except automating PE is hard

Our knee-jerk reaction to being unable to reproduce our failure states was to automate every part of the PE deployment, from downloading PE over SSH to templating out an answers file to the PE installer and moving on to automating REST API requests to configure the PE Console.

This was overkill, wasted time, and was not very useful. We eventually found a balance between documenting each manual step thoroughly and doing most of the configuration with puppet.

That allowed us to eventually realize that Puppet Server would crash once we deployed our puppet code with r10k, and with help from the support team we were able to identify an edge case performance issue and work around it. More details about performance will come later.

Eventually, we had PE 2015.2 deployed and stable enough to test against, which led to our next question...

The Puppet CA

At some point, we were going to need to move nodes to the new stack. It would be convenient if we could move test nodes between the two stacks at will during development. We could re-sign the agent certificate every time we wanted to do this, but that would be a lot of work, so we opted for a solution that would let us use the same agent certificate for both infrastructures.

Duplicate the Puppet CA

1. delete the SSL directory from the new MoM
2. rsync the SSL directory from the old CA to the new MoM
3. bump the next serial number by a lot:
   `/etc/puppetlabs/puppet/ca/serial`
4. reissue all puppet certificates for the new install:
   https://docs.puppetlabs.com/puppet/latest/reference/ssl_regenerate_certificates.html

We opted to duplicate the CA from the old master to the new master of masters.

The result was that both the old and new infrastructure could issue certificates.

Problems with CA duplication

- Inaccurate inventory
- Certificate revocation list divergence
- Non-contiguous serial numbers

Other CA options

- Switch to new CA
- Point new infrastructure at old CA
- Use a separate SSL directory for testing
- Use an external CA service

The Plan

1. Overview
2. Building a test environment
3. Testing catalogs
4. Performance
5. The migration

Like I said before, we were moving from an old version of the 3x future parser to puppet 4, so we needed to test our puppet code to make sure it worked.

The easy way:

`catalog_preview`

https://forge.puppetlabs.com/puppetlabs/catalog_preview

The easy way to test code moving between puppet 3 and puppet 4 is to use the catalog_preview module, which was recently open-sourced after initially being released as a PE-only module.

The module provides a puppet face that lets you compile catalogs from two different environments for a node, then it compares them and gives you a report on the differences. If you're running puppet 3.8 on your master, then you can use this to see the difference between the 3x parser and the 4x parser. You have to be running 3.8 for this because it was that version where they added the ability to set the `parser` setting in `environment.conf` when using directory environments, which you need.

Unfortunately, we couldn't use this, for two reasons:

1. We weren't running puppet 3.8
2. We weren't moving from the 3x parser to the 4x parser but from an old future parser to the 4x parser.

Instead we opted to...

Manually compile catalogs for all nodes

Attempt 1:

```
puppet master --compile
```

Our first attempt at compiling catalogs was to use the `puppet master` face, which can do compilations. Unfortunately, that uses MRI ruby, and we ran into some issues with that where things didn't work quite the same compared to JRuby and Puppet Server.

Attempt 2:

`POST /puppet/v3/catalog`

~~cronjob to sync facts cache~~

~~source from PuppetDB~~

Next, we attempted to use Puppet Server's API to request a catalog. However, we needed the facts from the node, otherwise this wasn't going to work.

First, we hoped to sync the cached YAML files of the facts that the puppet master stores. However, the REST API won't use those.

Next, we tried pulling the data out of our existing PuppetDB. That also failed because the our older PuppetDB didn't have an easy way to pull out all the fact data we need in the proper form for the API. Newer versions of PuppetDB have a `factsets` endpoint which might make that easier.

Let the agent request the catalog

```
if $::osfamily != "windows" {
  cron { 'pe agent':
    ensure  => present,
    command => join(['/opt/puppet/bin/puppet agent',
                     '--no-daemonize --onetime',
                     ], ' '),
                   minute  => [fqdn_rand(25), fqdn_rand(25) + 30],
  }

  cron { 'pe agent noop run':
    ensure  => present,
    command => join(['/opt/puppet/bin/puppet agent',
                     '--test --no-use_srv_records --noop',
                     '--catalog_cache_terminus=""',
                     '--server puppet-next.ops.puppetlabs.net;',
                     '/opt/puppet/bin/puppet plugin download',
                     ], ' '),
    minute  => [fqdn_rand(25) + 4, fqdn_rand(25) + 34],
  }
}
```

We were already using cron to run puppet on everything except Windows because we had some issues with the splay feature of the puppet agent daemon. We realized that we could just add an additional cron job to our puppet code to run the agent with the `--noop` flag against the new infrastructure.

This is the puppet code we used to test our nodes. We set the noop cron job to run four minutes after the normal run, and immediately after we run `puppet plugin download` to fix our pluginsync, just in case.

Note the `--catalog_cache_terminus` flag. That wasn't actually present in our cron job. I'll explain why I added it here later.

Making use of data

- PE Console
- Reports processors
- PuppetDB
- `catalog_preview` module

Once you have catalogs being compiled, you can make use of the data from anywhere you can collect it. You can then use it to fix any code issues from puppet 3 to puppet 4.

The Plan

1. Overview
2. Building a test environment
3. Testing catalogs
4. Performance
5. The migration

As I mentioned before, performance was a big issue for us, but also a great eventual benefit.

JRuby worker threads

- `max-active-instances`
- `max-requests-per-instance`
- `environment_timeout`

The switch to Puppet Server had a big impact on performance. We found out we needed a large amount more RAM to run Puppet Server, an increase from 4G per node to 32G per node. Now, this is extremely atypical and is the result of how puppet manages code cache and our enormous amount of puppet code.

Our code takes close to 10G per environment to cache. This meant that we had to carefully balance the number of JRuby worker threads we deployed (`max-active-instances`) with the total JVM heap. At first we tried killing instances with `max-requests-per-instance`, but that led to performance issues, plus one version of Puppet Server had a bug where it never flushed the memory from killed workers.

We set the `environment_timeout` to 0, which causes puppet to flush the code cache every compilation. This has a small negative performance impact, but it keeps our memory usage reasonable. We'd like to experiment with other JVM tuning settings to see if we can improve things and switch that to the recommended value of `unlimited`, which would improve performance.

PE Console class sync

`classifier_syncronization_period = 0`

```
path /puppet/v3/resource_type
method find, search
auth yes
# start OPS-7229 changes
# Only allow PE Console to sync class data for the production environment
environment production
# end OPS-7229 changes
allow pe-internal-dashboard, pe-internal-classifier
```

What ultimately triggered this issue for us was that the PE Console will request a complete list of all classes and their parameters from the master of masters in order to populate autocomplete. With as much code as we had, this caused performance issues for both the PE Console and Puppet Server. We took two actions to resolve this:

1. We set the classifier_syncronization_period to zero, which disables automatic syncing. The data can still be synced by manually triggering it from in the web GUI.
2. We modified `auth.conf` to block the PE Console from requesting class data from anything other than the `production` environment.

The engineering team is currently working on refactoring how this process works to improve performance.

CPU and compile times

As I showed at the beginning of this presentation, our compile times dropped dramatically. CPU performance was complicated to evaluate because of the way ESXi handles CPU resource allocation, but it seems to have improved as a result of moving to Puppet Server and puppet 4.

Metrics

We tried enabling the puppetserver built-in metrics system, but it crashed our graphite instance. Based on our environment, it shipped almost 200k metrics each metrics interval. The default is 5 seconds, but even at 1 minute, we were receiving nearly one million metrics per minute. Unfortunately, there's as of now no way to configure individual metrics.

When we needed to investigate tuning specifics, we enabled JMX metrics and connected JConsole to the JVM server process and manually inspected the metrics. We're currently looking at other ways to get specific metrics out.

Some metrics are available via various HTTP endpoints, depending on the individual component. Otherwise, everything should be available via JMX, so any standard Java solution that lets you inspect JMX metrics should work.

The Plan

1. Overview
2. Building a test environment
3. Testing catalogs
4. Performance
5. The migration

Once we had tested all our code, we were ready to actually do the migration.

Things to consider

- Service discovery via PuppetDB
- CA switchover
- How to switch the nodes
- Agent version

Service discovery via PuppetDB, including exported resources or puppetdb_query, can break if you move between separate stacks. Fortunately for us, our `--noop` populated PuppetDB, so we didn't have to worry about that.

CA switchover was also not a problem because we duplicated the CA, but if we had done something different, we would have had to figure out how to manage that.

For us, all our nodes were pointed at a CNAME, which made the switch as easy as pointing the CNAME at a new node. If that were not the case, we would have had to push new code that would reconfigure settings in `puppet.conf` to point the nodes at the new master.

We were running the puppet 3.8 agent because it works with both puppet 3 and puppet 4. I think it's only explicitly supported with puppet 3.8 and puppet 4.2, but we were able to run it against puppet 3.4.3 without any major issues.

Two days before our migration...

The migration was scheduled for a Wednesday, the day after PE 2015.3 released. Two days before that, Monday, a few people mentioned their nodes running against the new infrastructure unexpectedly. At first, we dismissed it as someone testing nodes and forgetting to switch back, but later in the day we realized that about a quarter of our nodes were running against the new infrastructure., many of them being production which should not have been switched over yet.

Turns out, the Friday before, I had reissued the certificates for our old puppet infrastructure so that it would be accessible from an alternate CNAME (puppet-legacy. instead of puppet.). I did this one master at a time so as to avoid a service outage, but on one of them I accidentally issued a certificate without any DNS alt names and put it back into the LB pool for a few minutes. Any node which got routed to that server failed to retrieve a catalog and used the cached catalog.

Now, note that within our puppet code, we had logic to switch some of the code based on the compiling master's version. Catalogs compiled against the new infrastructure would configure `puppet.conf` to point at the new infrastructure, and catalogs compiled against the old infrastructure would configure `puppet.conf` to point at the old infrastructure.

Due to the way we tested, the cached catalog was from the noop runs... which meant all those nodes switched to the new puppet infrastructure. So, we had almost a quarter of our nodes migrate to the new infrastructure 5 days early... and we didn't notice for 3 days. I'd call that a successful migration.

Don't cache --noop catalogs

```
puppet agent -t --noop --catalog_cache_terminus=''
```

You may remember that I mentioned the `catalog_cache_terminus` flag as something I included in the code I showed but wasn't in our original code. This is why. After this incident, I learned of this flag to prevent puppet from caching the catalog.

This issue is fixed in `master`, but there's no published agent version with that fix yet, and of course it won't be present in any old versions.

Overall outcome:

SUCCESS!

Questions?

Source: https://github.com/thrnio/puppet3-to-puppet4