Untangling the Strings

Scaling Puppet with inotify

Steven McDonald
steven.mcdonald@anchor.net.au
Anchor

linux.conf.au 2015
Sysadmin miniconf

Background

- Puppet is a centralised configuration management system.

Background

- Puppet is a centralised configuration management system.
- Each Puppet-managed host (“**node**”) runs a client program (the “**Puppet agent**”).

Background

- Puppet is a centralised configuration management system.
- Each Puppet-managed host (“**node**”) runs a client program (the “**Puppet agent**”).
- The server (“**Puppet master**”) tells the Puppet agent what the node's configuration should be.

Background: Puppet rollouts

- We use a single “production” environment with many (close to 1000) nodes.

Background: Puppet rollouts

- We use a single “production” environment with many (close to 1000) nodes.
- We use global virtual resources for things like monitoring on unmanaged hosts.

Background: Puppet rollouts

- We use a single “production” environment with many (close to 1000) nodes.
- We use global virtual resources for things like monitoring on unmanaged hosts.
- We make very small changes (usually specific to one node) that we want to take effect immediately.

Background: Puppet rollouts

- We use a single “production” environment with many (close to 1000) nodes.
- We use global virtual resources for things like monitoring on unmanaged hosts.
- We make very small changes (usually specific to one node) that we want to take effect immediately.
- This is a very slow workflow with Puppet.

The goal

- Have Puppet manifest changes apply immediately after rolling out to the Puppet master.

The goal

- Have Puppet manifest changes apply immediately after rolling out to the Puppet master.
- Historically, we have achieved this by restarting the Puppet master on every rollout.

The problem

- The Puppet master takes a long time to parse manifests into types.
- The time taken is negligible with up to a few dozen manifests, but quickly escalates from there.

The problem

- The Puppet master takes a long time to parse manifests into types.
- The time taken is negligible with up to a few dozen manifests, but quickly escalates from there.
- Our tree has over 1300 manifests in our site directory (i.e., loaded on startup and not autoloaded). This takes just over a minute to parse.

What solutions does Puppet offer?

- Puppet has very coarse internal caching; it is capable of expiring an entire environment at a time.

What solutions does Puppet offer?

- Puppet has very coarse internal caching; it is capable of expiring an entire environment at a time.
- With all our nodes in one environment, this is as good (or as bad) as a Puppet master restart.

What solutions does Puppet offer?

- Puppet can have distinct environments for different groups of nodes, each with their own (smaller) set of manifests.

What solutions does Puppet offer?

- Puppet can have distinct environments for different groups of nodes, each with their own (smaller) set of manifests.
- While this is a good idea in theory, having all our nodes in the same environment is the best fit for our workflow.

What solutions could we implement?

- Filesystem polling.

What solutions could we implement?

- Filesystem polling.
- Extract changed file information from git.

What solutions could we implement?

- Filesystem polling.
- Extract changed file information from git.
- Listen for changes to files using Linux's inotify subsystem.

What solutions could we implement?

- Filesystem polling.
- Extract changed file information from git.
- Listen for changes to files using Linux's inotify subsystem.
- All of these options require one piece of infrastructure we needed to implement ourselves: the ability to expire code on a per-file basis.

Internal relationships

Per-file code expiration

- We implemented a **general-purpose** file expiration mechanism, to expire code in types, and to expire entire types in type collections.

Per-file code expiration

- We implemented a **general-purpose** file expiration mechanism, to expire code in types, and to expire entire types in type collections.
- Because of the generic nature of the expiration API, it can easily be adapted to any method of determining which files have changed.

Option #1: Filesystem polling

- Most portable.

Option #1: Filesystem polling

- Most portable.
- Too slow. Just about any other option is more efficient.

Option #1: Filesystem polling

- Most portable.
- Too slow. Just about any other option is more efficient.
- This could be implemented as a fallback mechanism that works anywhere, but we wanted to take advantage of the specific features of our environment.

Option #2: Asking git for changes

- This is a very clean and efficient idea.

Option #2: Asking git for changes

- This is a very clean and efficient idea.
- It ties us to a git-based deployment model.

Option #2: Asking git for changes

- This is a very clean and efficient idea.
- It ties us to a git-based deployment model.
- It requires us to queue changes ourselves, in code that's unlikely to see widespread testing.

Option #2: Asking git for changes

- This is a very clean and efficient idea.
- It ties us to a git-based deployment model.
- It requires us to queue changes ourselves, in code that's unlikely to see widespread testing.
- Very easy to introduce bugs that miss changes.

Option #3: inotify

- It ties us to Linux Puppet masters.

Option #3: inotify

- It ties us to Linux Puppet masters.
- It does not tie us to any specific deployment method.

Option #3: inotify

- It ties us to Linux Puppet masters.
- It does not tie us to any specific deployment method.
- It does not require us to do any dirty work ourselves; the inotify code in the kernel is very well tested.

Option #3: inotify

- It ties us to Linux Puppet masters.
- It does not tie us to any specific deployment method.
- It does not require us to do any dirty work ourselves; the inotify code in the kernel is very well tested.
- Least risk of introducing bugs.

Triggering expiry with inotify

- The autoloader requests that files simply be expired when they change; they will be re-autoloaded as necessary.
- The initial importer (which parses the environment's manifests directory) requests that files be reparsed when they change, since these are always supposed to be loaded.

Triggering expiry with inotify

- The autoloader requests that files simply be expired when they change; they will be re-autoloaded as necessary.
- The initial importer (which parses the environment's manifests directory) requests that files be reparsed when they change, since these are always supposed to be loaded.
- This all happens at the start of a catalog compilation.

Triggering expiry with inotify

- The deprecated “import” function makes it extremely difficult to track which files need to be reparsed when they change.

Triggering expiry with inotify

- The deprecated “import” function makes it extremely difficult to track which files need to be reparsed when they change.
- We elected not to support “import”, and took the time to remove it from our manifests.

Results

- Based on initial testing on a lightly-loaded staging environment, we expected around **70 seconds** shaved off the average agent run immediately after rollout.

Results

- Based on initial testing on a lightly-loaded staging environment, we expected around **70 seconds** shaved off the average agent run immediately after rollout.
- After deploying to production, we had anecdotal speed improvements of **up to 5 minutes** on nodes with complex catalogs.

Limitations

- Our initial implementation does not correctly support the future parser.

Limitations

- Our initial implementation does not correctly support the future parser.
- Reopening a class in a different file is not supported.

Limitations

- Our initial implementation does not correctly support the future parser.
- Reopening a class in a different file is not supported.
- The use of “import” is not supported.

Limitations

- Our initial implementation does not correctly support the future parser.
- Reopening a class in a different file is not supported.
- The use of “import” is not supported.
- Native Ruby code (used for custom functions) does not get reloaded.

Get the source

- https://github.com/AnchorCat/puppet/tree/anchor/3.7.3/in
  - Depends on a yet-unreleased version of rb-inotify, but will work with the current master branch.
- OpenDocument sources for this presentation are available under the WTFPL:
  - http://steven.beta.anchortrove.com/lca2015/untangling_the_s
- Ruby inotify bindings:
  - https://github.com/nex3/rb-inotify