Fedora Infrastructure Puppet Training

Mike McGrath
Fedora Infrastructure
2008-09-15

Puppet Configuration Management

- Controls what config files end up where
- Package installation, removal
- Service restarts, chkconfig, service X start
- Dep checking (file level, service level, etc)

Basic Theory

- Simple provisioning
- Reproducability
- Set once, deploy everywhere
- Auditing
- Automation

Key Terms

- Puppetmaster – The primary central configuration server
- Node – Individual hosts
- Manifest – Anything written in 'puppet language' including classes, nodes, etc.

Master/Node Model

- Puppetmaster uses key based auth and provides a manifest to the nodes.
- Nodes register with puppetmaster on their first run. These requests are signed with puppetca.
- Nodes check in every 30 minutes to pull any updates.
- The nodes do all the translation of the manifest into a localconfig.yaml
- bitbucket

Using Puppet - Language

- Manifest Language Example

```
file { '/etc/ssh/sshd_config':
    source => 'puppet:///configs/system/sshd_config',
    mode => 0600,
    notify => Service['sshd'],
    require => Package['openssh-server'],
}
```

Using Puppet - Packages

- Continuing with ssh example

```
package { openssh-server:
    ensure => present,
    }
package { telnet-server:
    ensure => absent,
    }
```

Using Puppet - Services

- Continuing with ssh example

```
service { sshd:
    ensure => running,
    enable => true,
    require => Package['openssh-server']
    }
```

Using Puppet - Classes

- Continuing with ssh example
- Classes tie everything together

```
class ssh-server {

     file { '/etc/ssh/sshd_config':

          source => 'puppet:///configs/system/sshd_config',

          mode => 0600,

          notify => Service['sshd'],

          require => Package['openssh-server'],

     }

     package { openssh-server:

          ensure => present,

     }

     service { sshd:

          ensure => running,

          enable => true,

          require => Package['openssh-server']

     }

}
```

Quick Review

- File – a resource for configuration and other types of files
- Service – a resource for actual running processes
- Package – a resource for actual package management
- Class – Ties file, service and package types together (as well as other types not yet discussed)

Nodes

- Each host is a node
- Nodes should generally not define resources in the actual definition.

```
node { app1:
   include ssh-server
}
```

Server Group Abstraction

- Technique used to group like classes together to create a server or node type
- Uses classes that include other classes
- The basic workflow is:
  - Node includes servergroup classes
  - Servergroup classes include worker classes
- Example:

```
class proxyServer {
    include proxy
    include ssh-server
}
```

Server Group Abstraction (contd)

- The node would include our proxyServer servergroup class
- Example

```
node { proxy1:
    include proxyServer
}
```

Modules

- Defined on a per package basis
- Self contained
- Sharable
- Contain config files, templates, manifests, etc.

Module Layout

- modules/modulename/
  - files/
  - manifests/
    - manifests/init.pp
  - plugins/
  - templates/
  - README

Module – SSH server

- Lets convert our ssh-server class from earlier into a module.
- Module names should be based on base package name.
  - openssh, not openssh-server
- Starting out we only need the README, manifests/init.pp and files/ directory

Modules - init.pp

- Located in openssh/manifests/init.pp
- Is auto included by puppet
- Slightly different class format:

```
class openssh::sshd {
   file { '/etc/ssh/sshd_config':
      source => 'puppet:///openssh/sshd_config',
      mode => 0600,
      notify => Service['sshd'],
      require => Package['openssh-server'],
   }
}
```

Modules - Exceptions

- A few exceptions exist to the module naming scheme
- Custom scripts should now go in the 'scripts' module
- Legacy code not yet moved is still in the old manifests/ and configs/ format
- Private repo stuff (passwords and things) can be referenced in modules, but should still be stored in the private repo for security

Modules – What's changed

- Specific to Fedora Infrastructure
- We used to have configfile and apachefile
  - All requisites must be explicit
- All 'source =>' parameters must have puppet:///

Advanced Puppet Topics

- Puppet Variables
- Facter
- Basic logic syntax (if, case, arrays)
- Config File templating
- Custom defines

Puppet Variables

- Take on formatting similar to php
- `$groups='sysadmin-main'`
- `$tcpPorts = [ 80, 443 ]`
- `$otherGroups = "$groups foo"`
- `$otherGroups2 = '$groups foo'`

Facter

- Seperate application, yum install facter.
- Can be invoked by just running 'facter'.
- Any variable in facter can be referenced by puppet.
- Custom facter recepes can be written.
- Example: `hostanme => proxy1`

Custom Facter Recipes

- Written in Ruby
- Place .rb files on the local machine in `/usr/lib/ruby/site_ruby/1.8/facter/your-recipe.rb`
- Basic example:

```
Facter.add('variableName') do
    setcode do
        "'Variable Value'"
    end
end
```

if Logic – Manifest

- Can be used in most parts of a manifest.
- Example:

```
if $proxy_backup {
   include proxy_backup_data
} else {
   include proxy_live_data
}
```

case Logic - Manifest

- Similar to if can be used in most places in a manfiest
- Example:

```
case $architecture {
    'i386' : {
        include i386-host
    }
    'x86_64': {
        package { electric-avenue:
            ensure => present
        }
    }
}
```

Templating

- Use an ERB templating format
- Converted tags between <% and %>
- <%= Replaces with a result %>
- <%= Replaces with a result, no new line at end -%>
- <%# Comment %>
- Example:

```
<%= architecture %>
```

Templating – if

- Similar to manifest logic.
- Variable names in ERB do not prepend $
- Example:

```
<% if hostname == 'proxy1' %>
ThisIsAProxyHost=True
<% else %>
ThisIsAnAppServer=True
#This Host Isn't Proxy1
SpecialVar=<%= specialValue %>
<% end %>
```

Using a template

- Place templates in the templates/ directory not files/.
- Append .erb to all filenames, for example httpd.conf.erb
- In the manifest don't use

```
source => 'puppet:///modulename/template.conf'
```

instead use

```
content => template('modulename/templateName.conf.erb')
```

Custom Defines

- Mostly a manifest template used with substitution
- Example is selinux boolians
- allow_httpd_mod_auth_pam
- Normally set with `setsebool -P allow_httpd_mod_auth_pam=on`
- Normally checked with `getsebool allow_httpd_mod_auth_pam`

Custom Defines - selinux_bool

- We're going to define a selinux_bool type
- $name is passed automatically

```
define selinux_bool($bool) {
   exec { "/usr/sbin/setsebool -P $name=$bool":
      unless => "/usr/sbin/getsebool $name | grep -qe
        $bool$",
      cwd => '/',
   }
}
selinux_bool { 'allow_httpd_mod_auth_pam':
   bool => 'on',
}
```

exec

- Should be avoided when possible
- Can be 'notify =>'ed

```
exec { 'fix_sendmail':
    command => '/etc/init.d/sendmail stop; /bin/rpm -e
      sendmail; /etc/init.d/postfix restart',
    onlyif => '/usr/bin/pgrep sendmail',
    cwd => '/',
}
```

Common tasks - Directory

- Create a directory
- Does not auto create parent directories
- Example:

```
file { ['/srv/web', '/srv/web/cache']:
   ensure => directory
}
```

- Could not require => File['/srv']
- Also cannot require => File['/srv/web/']
- Could require => File['/srv/web']

Common Tasks - restart

- Using httpd as an example
- Service httpd restart is the default
- Graceful is more graceful, includes configtest

```
service { httpd:
    ensure => true,
    enable => true,
    restart => '/etc/init.d/httpd graceful'
}
```

Git workflow

- Git manages modules, configs and manfiests
- Emails get sent on every git push
- Hook syncs git HEAD with what puppetmaster actually uses
- Test new configs with puppetd -t --noop

The End

- Questions? - mmcgrath@redhat.com
- See reductivelabs.com for more info
- Thank you