UNIVERSITY OF BIRMINGHAM

Converting from cfengine to Puppet

HEP System Managers Meeting, 10th May, 2012

Mark Slater, Birmingham University

Disclaimer

I consider myself neither a cfengine expert nor a puppet expert!

My cfengine experience is entirely based on the scripts I've inherited and some of the issues I've encountered may just be due to ignorance

My puppet experience is very Bham-centric and I know the scripts I've written are not taking full advantage of all puppet can do - out of necessity, they are more a 'get-this-working-ASAP' style.

Consequently, this talk is very much more an introduction to puppet, how it's different from cfengine and what I personally think are the pros and cons!

The World According To cfengine (1)

cfengine runs like a functional program, i.e. top to bottom, one step after another, as defined by the input files

A cfengine process on the head node connects to processes running on the other nodes and works through the required steps

A typical run will sequentially step through and process the following:

- Mount points
- Package management
- Specific shell commands
- File edits
- Process handling

The World According To cfengine (2)

A typical cfengine config file will have entries for the above areas that tell the cfagent process what to do, e.g. create a file, run a script, kill this process, etc.

Config files can be combined using the include directive and additional 'user' created modules composed of custom scripts can be used

There are many built in cfengine commands for manipulating text files and processes as well as providing some conditional execution control

The World According To Puppet (1)

Whereas cfengine runs like a functional program, puppet describes things in a *much more object oriented way*

Everything that you want to control about the target machine is considered a resource and *what each resource is and in what order they are applied gives the control*

Consequently, when defining a puppet 'manifest', instead of sections, you define resources to control elements of the node:

- Files
- Services
- Packages

These can then be chained together as you wish in order to structure the management of the node and ensure all files, packages and services are setup as required

The World According To Puppet (2)

As puppet is more object oriented, it is therefore significantly easier to *modularise the resource definitions*

Each module can contain the *setup and configuration needed for a specific service, package or system resource*

At that point, adding this to a node definition will ensure all the setup is carried out on this node

You still need a head node but with puppet, the client initiates the connection, retrieves it's manifest and then runs through the resources

Unless specifically specified, each resource is handled independently and so can be executed in any order

Pros and Cons

Pros of puppet:

- Light weight (in my experience!) - easy to check if services are running, etc.
- Very understandable manifest system that is similar to C++ classes
- Can chain together any set of resource actions in any order
- Easy to modularise
- Ruby powered templating available
- With the class system, can create e.g. iptables rules
- Easy conditional, argument and variable based resource creation

Cons of puppet:

- Not as intuitive for, e.g. running scripts
- Not as easy to edit files
- Running a one-time script is tricky

Note: I realise the majority of the things puppet does can be done in cfengine, but it's a question of how difficult it is and how many scripts are required!

Install and Setup

Puppet is quite easy to install and get running and is available through the epel repo

You'll need a *dedicated 'puppet master' service* that will hold the resource definitions and each controlled node needs to run the puppet service

After installation, the usual key exchange/verification is required but then the controlled node can query the puppet master and run through all the resources that are defined for it

It can be run manually at any time but will usually be run as a service that will rerun the resource check every 30mins

During this run, *all checks that can be run, will be run*. If there are any errors during a resource check, only dependent resources will not be run, all others will

Resources And Manifests

Resources can be one of a large number of different types that control various aspects of the machine, e.g.

- Files - can handle content, permissions, etc. of files and directories
- Exec - execute a particular script
- Group and User - manage specific users and groups
- Mount - manage mount points
- Service - manage what services are running
- Package - decide whether and how to install a package

Note there are many more resources to handle all elements of the system

These resources are defined in 'manifest' files for each node or module using a simple syntax

They can be dependent (or depend) on other resources in a number of ways, e.g. straight requirement, notification, subscription, etc.

Resources Examples

```
class httpd {

  package { "httpd":
    ensure => installed,
  }

  service { "httpd":
    name   => 'httpd',
    ensure      => running,
    enable      => true,
    hasrestart => true,
    hasstatus   => true,
    subscribe   => [ File['/etc/httpd/conf/httpd.conf'],
Package["httpd"] ],
    require     => Package["httpd"],
  }

  file { '/etc/httpd/conf/httpd.conf':
    ensure  => present,
    source => "puppet:///modules/httpd/httpd.conf",
    require => Package['httpd'],
  }
}
```

Ensure package "httpd" is installed - yum is used by default, but other installers (e.g. rpm) can be used

Ensure the service "httpd" is running

The service requires the package to be installed, the config file to be present and will be restarted when the config changes

A basic file resource using a file from the puppet master as the source

Variables, Conditionals, Facts

A significant degree of control can be had by using *variables and conditionals* within the manifests

Variables can be *defined manually or use the built-in 'Facts'* which give a large amount of info about the system (e.g. hostname, fqid, os, etc.) which can be used both in conditionals for defining different resources

Variables are also vital in file resources that are created (using Ruby) from templates:

```
##########   WN Variables  ##############

USERS_CONF=/root/puppet_files/modules/yaim/users.conf
GROUPS_CONF=/root/puppet_files/modules/yaim/groups.conf

BDII_HOST=lcg-bdii.gridpp.ac.uk
MON_HOST=<%= apel_host %>.ph.bham.ac.uk


SE_LIST="<%= dpm_host %>.ph.bham.ac.uk"
DPM_HOST=<%= dpm_host %>.ph.bham.ac.uk
```

Modules and Classes

A very powerful feature of puppet is the ability to *create classes and modules that can be controlled by arguments* - again, similar to C++ constructors

You can then easily *encapsulate everything about*, e.g. httpd setup through the included resources and just add this single 'resource' to a host definition to make sure it has a *fully installed and setup httpd server on it*

As an example, I have created a class that handles rules in iptables and which allows you to specify them as a new resource:

```
iptables::rule {"httpd-in":
    chain => 'INPUT',
    jump  => 'ACCEPT',
    dport => '80',
    proto => 'tcp',
    src => '147.188.46.0/255.255.255.0',
  }
```

Case Study: Birmingham

As I said previously, I have not used all the functionality of puppet but I have found it to be *a lot easier and more stable than cfengine*

I have written modules for all the major services required, including:

- Web Server
- Torque
- Yaim
- PXEBoot
- CREAM

I can now deploy replacements for these services very quickly thanks to the modular nature of puppet

UNIVERSITY OF BIRMINGHAM

Case Study: Overall Site Config

A significant problem was wanting to have *a single cluster config file* that defined and controlled all the nodes but responded to changes

In order to do this, I set up a *two stage puppet process*: When running puppet on the head node, all the kickstart, configs and puppet manifests are generated from given site manifest

This allows the following to be done in the overall site config file:

```
tftp::vm{ "epgr13":
  ipaddr  => '147.188.46.60',
  macaddr => '00:16:3E:34:C4:06',
  os_version => $os_version,
  vm_host => 'epgpe11',
  num_cpus => '8',
  vm_mem => '8',        # in GB
  slot_num => '1',
  role => 'pbs_server',
}
```

Arguments to the class that will be used in to the generated scripts

```
tftp::vm_host{ "epgpe11":
  ipaddr  => '147.188.46.42',
  macaddr => '78:2B:CB:36:E5:4A',
  os_version => $os_version,
  num_vms => '2',
  total_hd => '450', # in GB
}
```

os_version used was defined elsewhere (5.8 in this case!)

Case Study: Yaim Module

```
class yaim($role) {

  file { '/root/puppet_files/modules/yaim/site-info.def':
    ensure  => present,
    content => template("yaim/main-site-info.def.erb", "yaim/${role}/site-
info.def.erb"),
  }

  case $role {
    'cream_ce':  {
      exec { 'yaim::reyaim':
        command => '/opt/glite/yaim/bin/yaim -c -s
/root/puppet_files/modules/yaim/site-info.def -n creamCE -n TORQUE_utils',
        refreshonly => true,
        subscribe => File['/root/puppet_files/modules/yaim/site-info.def'],
      }

    'pbs_server':  {
      exec { 'yaim::reyaim':
        command => '/opt/glite/yaim/bin/yaim -c -s
/root/puppet_files/modules/yaim/site-info.def -n TORQUE_server -n TORQUE_utils',
        refreshonly => true,
        logoutput => true,
        subscribe => File['/root/puppet_files/modules/yaim/site-info.def'],
        notify => Exec['pbs_server::server_cfg'],
      }
    }
}
```

Role given as argument for class

Resources declared depending on role

Create the site-info file from a combination of a main file and a role specific file

Execute yaim only if site-info has been updated

Run a server config script after yaim has run

Summary

After a couple of months of getting used to puppet, I've managed to write all the manifests required and can now install new nodes quickly and easily

I have had no 'online' problems as yet with respect to crashes or hangs

It took a bit of time to get used to thinking the way puppet requires, but feel it was worth the effort!

With more time, I can see that there is a lot of power to be had from the modules and chaining of resources

Take home message:

I've had to write far fewer scripts with puppet than cfengine!

Links

A good start-up guide:

http://bitfieldconsulting.com/puppet-tutorial

A more in depth tutorial:

http://docs.puppetlabs.com/learning/ral.html

A complete list of Resource types:

http://docs.puppetlabs.com/references/stable/type.html