March 2014 WMLUG Meeting

OS Configuration Management

Patrick TenHoopen

OS Configuration Management

What is Configuration Management (CM)?

CM is the practice of handling changes systematically so that a system maintains its integrity over time. CM implements the policies, procedures, techniques, and tools that are required to manage, evaluate proposed changes, track the status of changes, and to maintain an inventory of system and support documents as the system changes.

- Wikipedia

Management Methods*

Divergence

Most common among undisciplined administrator teams. The hosts drift away from the desired state due to manual changes on a subset of hosts, inconsistent applying of changes to all hosts, or failure of the change to be applied correctly, and so on. Also the complexity of the changes or the number of hosts add to the problem.

*From “Why Order Matters: Turing Equivalence in Automated Systems Administration” http://www.infrastructures.org/papers/turing/turing.html

Management Methods

Convergence

This method usually follows a divergent one after the need arises to bring order to the chaos. The state of hosts starts to converge to the desired state using at first a manual process then through automation. Usually only a subset of files that are deemed critical are managed.

Management Methods

Congruence

This is the most controlled and desired method. All hosts are always at the target state. If any host diverges from this target state through application failure, administrator error, or security breach, the host can easily be rebuilt as all changes have been recorded for playback during the rebuild process. Changes can easily and predictability be made to hosts as desired.

Advantages of CM

- Hosts are in a known state
- Reduced management cost/unexpected downtime
- Changes are implemented consistently
- Hosts can easily be replaced/rebuilt
- Can easily do impact analysis of changes (predictability of behavior)

Disadvantages of CM

- Complex
- Steep learning curve
- Implementation time
- Mindset change
- Cost for enterprise-level deployments

Popular Systems

- Puppet
- Chef
- CFEngine
- Ansible
- Salt
- ISconf

For a more complete list, see
http://en.wikipedia.org/wiki/Comparison_of_open_source_configuration_management_software

Why use a CM tool instead of writing your own scripts?

By using a CM tool, you get:

- More features than scripts alone can provide
- Ready-made, purpose-based turn-key solution (mostly)
- Not reinventing the wheel (necessarily)
- Reporting/Analysis
- Inventory of system state
- Communities for help

Approaches

Declarative/Model-Based

The declarative/model-based approach specifies what the end result of the configuration should be, as in "this configuration file should have this in it", or "this piece of software should be installed". It uses dependencies and doesn't specify the specific commands to make it happen but instead relies on the local configuration client to figure things out. It defines state, not process.

This approach is able to take a system with a non-conforming state and make it compliant regardless of the starting state.

This also allows a resource to “listen” for changes and act appropriately (e.g., restart the service if the config file changes).

Puppet uses this approach.

Approaches

Programmatic/Procedural

The programmatic or procedural approach specifies exactly how the end result should be using specific commands.

This approach is best suited for systems starting from a known state whereby after you run the tool, you end up at the desired state.

Chef and ISconf use this approach.

What is Puppet?

Puppet is a declarative, model-based approach to IT automation, helping you manage infrastructure throughout its lifecycle, from provisioning and configuration to orchestration and reporting. Using Puppet, you can easily automate repetitive tasks, quickly deploy critical applications, and proactively manage change, scaling from 10s of servers to 1000s, on-premise or in the cloud.

Puppet is available as both open source and commercial software.

-Puppet Labs - http://puppetlabs.com/puppet/what-is-puppet

How Puppet Works

Define the desired state of the infrastructure's configuration using Puppet's declarative configuration language.

Simulate configuration changes before enforcing them.

Enforce the deployed desired state automatically, correcting any configuration drift.

Report on the differences between actual and desired states and any changes made enforcing the desired state.

Puppet vs Chef

- Puppet uses a custom JSON-like language and/or Ruby.
- Chef uses Ruby and Pure Ruby, a subset of Ruby.
- Puppet uses dependency management to perform tasks. Multiple runs of the task may perform differently.
- Chef uses specifically ordered tasks. Multiple runs of the task will perform the same.
- Puppet seems to be oriented toward a system admin and appears to have less of a learning curve.
- Chef seems to be more oriented towards developers, especially Ruby shops.
- Puppet is older than Chef and was influenced by CFEngine while Chef was influenced by Puppet.
- Puppet has a larger following than Chef.

Puppet Basics

Resources are defined which describe a file or a package. Resources can be grouped into classes to define a service or application. Classes can be combined to define a server type such as a web server.

The configuration files that have the resources and classes defined are called manifests and have a .pp extension.

Since Puppet is declarative, the resources don't need to be defined in order.

Example of Puppet Resources Defined in a Class

```
#ntp.pp
case $operatingsystem {
      centos, redhat: { $service_name = 'ntpd' }
      debian, ubuntu: { $service_name = 'ntp' }
    }

    package { 'ntp':
      ensure => installed,
    }

    service { 'ntp':
      name      => $service_name,
      ensure    => running,
      enable    => true,
      subscribe => File['ntp.conf'],
    }

    file { 'ntp.conf':
      path    => '/etc/ntp.conf',
      ensure  => file,
      require => Package['ntp'],
      source  => "puppet:///modules/ntp/ntp.conf",
      # This source file would be located on the puppet master at
      # /etc/puppetlabs/puppet/modules/ntp/files/ntp.conf (in Puppet Enterprise)
      # or
      # /etc/puppet/modules/ntp/files/ntp.conf (in open source Puppet)
    }
```

Learning Puppet

More information for learning Puppet is available at:

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

Puppet Test VM

You can download a free VM of a Puppet environment for testing here:

http://info.puppetlabs.com/download-learning-puppet-VM.html

What is Chef?

Chef is a systems and cloud infrastructure automation framework that makes it easy to deploy servers and applications to any physical, virtual, or cloud location, no matter the size of the infrastructure. The chef-client relies on abstract definitions (known as cookbooks and recipes) that are written in Ruby and are managed like source code. Each definition describes how a specific part of your infrastructure should be built and managed. The chef-client then applies those definitions to servers and applications, as specified, resulting in a fully automated infrastructure. When a new node is brought online, the only thing the chef-client needs to know is which cookbooks and recipes to apply.

- A Quick Overview of Chef - http://docs.opscode.com/chef_quick_overview.html

Chef Philosophy

The key underlying principle of Chef is that you (the user) knows best about what your environment is, what it should do, and how it should be maintained. The chef-client is designed to not make assumptions about any of those things. Only the individuals on the ground—that's you and your team—understand the technical problems and what is required to solve them. Only your team can understand the human problems (skill levels, audit trails, and other internal issues) that are unique to your organization and whether any single technical solution is viable.

Chef Basics

A resource is defined in a recipe using Ruby. A resource defines the actions to be performed on the system/node (service control, user/group creation, etc.) They are processed in the order they are defined within the recipe file which have a .rb extension.

Each resource within a recipe is identified and associated with a provider which does the work to complete the action defined by the resource. Chef maps providers to platforms using data collected by the Ohai system.

A cookbook is a collection of recipes that defines a use-case, such as everything needed to install and configure an app (e.g., web server), and contains all of the components that are required for it.

Knife

Knife is a command-line tool that is run from the Chef admin's workstation that provides an interface between a local chef-repo and the Chef server. It interfaces with a Chef server just like a chef client does. It helps users to manage nodes, cookbooks and recipes, etc.

Example of a Service Resource in Chef

```
#ntp.rb
service "ntp" do
  case node["platform"]
  when "CentOS","RedHat","Fedora"
    service_name "ntpd"
  else
    service_name "ntp"
  end
  supports :restart => true
  action [ :enable, :start ]
end
```

Questions?

CM is a big topic!

What questions do you have?