Puppet Tutorial

OSCON
2013-07-22
Portland, OR

Garrett Honeycutt
Automation Consultant @ GH Solutions, LLC
gh@garretthoneycutt.com
http://linkedin.com/in/garretthoneycutt
# whoami

an engineer and obviously not a graphic designer
Why?

- reduce entropy
Why?

- reduce entropy
- change management
Why?

• reduce entropy
• change management
• infrastructure as code
How Puppet Works

1. Define: With Puppet’s declarative language you design a graph of relationships between resources within reusable modules. These modules define your infrastructure in its desired state.

2. Simulate: With this resource graph, Puppet is unique in its ability to simulate deployments, enabling you to test changes without disruption to your infrastructure.

3. Enforce: Puppet compares your system to the desired state as you define it, and automatically enforces it to the desired state ensuring your system is in compliance.

4. Report: Puppet Dashboard reports track relationships between components and all changes, allowing you to keep up with security and compliance mandates. And with the open API you can integrate Puppet with third party monitoring tools.
Use Puppet to create composable configurations and manage the enterprise infrastructure

1. Define Your Resources in Modules. With Puppet, you define your modules by node classifications, such as Web Server or Database, allowing you to define relationships between resources and configure thousands of servers at once.

2. Assign resource relationships automatically. You can then assign and deploy configurations via Puppet Dashboard, or with your own customized CMDB tools.

3. Reusable, composable configurations. With Puppet you can re-use modules across multiple nodes, in whatever combination you need, reducing repetitive tasks and eliminating error-prone scripts.
Puppet Assigns and Maintains a Node’s Desired Role
Managing Configuration Drift
How Puppet Manages Data Flow for Individual Nodes

1. **Facts**
The node sends normalized data about itself to the Puppet Master.

2. **Catalog**
Puppet uses the Facts to compile a Catalog that specifies how the node should be configured.

3. **Report**
The node reports back to Puppet indicating the configuration is complete, which is visible in the Puppet Dashboard.

4. **Report Collector**
Puppet’s open API can also send data to third party tools.

SSL secure encryption on all data transport.
Facts

Automatically Maintained Asset Inventory
architecture => i386
domain => local
facterversion => 1.6.6
fqdn => sliver.local
hardwareisa => i386
hardwaremodel => i386
hostname => sliver
id => gh
interfaces => lo0,gif0,stf0,en0,en1,fw0
ipaddress => 192.168.101.185
ipaddress_en1 => 192.168.101.185
ipaddress_lo0 => 127.0.0.1
is_virtual => false
kernel => Darwin
kernelmajversion => 10.8
kernelrelease => 10.8.0
kernelversion => 10.8.0
memoryfree => 102.80 MB
Custom Facts

```ruby
# role.rb
require 'facter'
Facter.add("role") do
  setcode do
    Facter::Util::Resolution.exec("cat /etc/role")
  end
end
```
How Puppet Manages Data Flow for Individual Nodes

1. **Facts**
   - The node sends normalized data about itself to the Puppet Master.

2. **Catalog**
   - Puppet uses the Facts to compile a Catalog that specifies how the node should be configured.

3. **Report**
   - The node reports back to Puppet indicating the configuration is complete, which is visible in the Puppet Dashboard.

4. **Report**
   - Puppet’s open API can also send data to third party tools.

---

**SSL secure encryption on all data transport**

**Report Collector**
(Puppet or 3rd party tool)
Catalog

Comprehensive list of resources
How Puppet Manages Data Flow for Individual Nodes

1. **Facts**: The node sends normalized data about itself to the Puppet Master.

2. **Catalog**: Puppet uses the Facts to compile a Catalog that specifies how the node should be configured.

3. **Report**: The node reports back to Puppet indicating the configuration is complete, which is visible in the Puppet Dashboard.

4. **Report Collector** (Puppet or 3rd party tool):
   - SSL secure encryption on all data transport
   - Puppet’s open API can also send data to third party tools.
Reporting

every change correlated to every resource
Reporting

- http/https
- log
- store
- tagmail
- custom processors
  - irc
  - twitter
  - jabber
  - growl
package { 'ntp':
  ensure => installed,
}
What not how

$ ls puppet/lib/puppet/provider/package
aix.rb     blastwave.rb   hpux.rb   ports.rb       up2date.rb
app dmg.rb   darwinport.rb nim.rb   portupgrade.rb urp mi.rb
apple.rb    dpkg.rb      openbsd.rb rpm.rb       yum.rb
apt.rb      fink.rb      pkg.rb    rug.rb         yumhelper.py
aptitude.rb  freebsd.rb   pkgdmg.rb portage.rb sun.rb
aptrpm.rb   gem.rb       portage.rb sunfreeware.rb zypper.rb
Example Resource Types

- cron
- exec
- file
- group
- host
- zfs
- mount
- package
- service
- sshkey
class motd {
    file { '/etc/motd':
        owner => 'root',
        group => 'root',
        mode  => '0644',
        source => 'puppet:///modules/motd/generic_motd',
    }
}
Templates

# motd.erb

Welcome to `<%= fqdn %>`

kernel version = `<%= kernelversion %>`
puppet version = `<%= puppetversion %>`
facter version = `<%= facterversion %>`
search <%= dnssearchpath %> options ndots:2 timeout:3
<% nameservers.each do |nameserver| %>
nameserver <%= nameserver %>
<% end %>
package-file-service

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

  file { '/etc/ntp.conf':
    owner   => 'root',
    group   => 'root',
    mode    => '0644',
    source  => 'puppet:///modules/etc/ntp.conf',
    require => Package['ntp'],
  }

  service { 'ntpd':
    ensure   => running,
    enable   => true,
    subscribe => File['/etc/ntp.conf'],
  }
}
Bootstrap
Modify /etc/hosts

172.16.132.128 puppet.oscon.priv puppet
useful agent flags

Run one time, verbose, in the foreground

- # puppet agent -t

add -d for debug

- # puppet agent -t -d
Facter

all facts
  • # facter

specific fact
  • # facter fqdn

custom facts
  • # facter -p
puppet resource

Specific resource

- # puppet resource <resource type> <specific resource>
- # puppet resource user root

All resources of that type

- # puppet resource <resource type>
- # puppet resource group
puppet resource

add a user

- # puppet resource user gh ensure=present

stop and start a service

- # puppet resource service postfix ensure=stopped
- # puppet resource service postfix ensure=running
configs

/etc/puppet/puppet.conf

- # puppet config print
- # puppet config print modulepath
modulepath

- where to look for modules
- acts like $PATH - stop at first match
- great for working with multiple teams, just put your path first
modules

- manage specific parts of your system
- directory structure for finding your code
Create our first module

- # cd /etc/puppet/modules

- # puppet module generate yourname-motd && mv yourname-motd motd
modules

What’s all here?

- # tree motd
Modulefile

- meta data
- semver.org
- dependencies
Forge

- search for packages
  - # puppet module search mysql

- http://forge.puppetlabs.com
install a package

# puppet module install ghoneycutt-dnsclient
site manifest

/etc/puppet/manifests/site.pp

include dnsclient class

• # puppet agent -t

check out /etc/resolv.conf

• # cat /etc/resolv.conf
classes

- contains the code
- can be included in site manifest or in other classes
- check out motd
Resources

type { 'title':
    attribute => value,
}

file {
  "/etc/motd":
    ensure => file,
}

file

file { '/etc/motd':
  ensure => file,
  owner => 'root',
  group => 'root',
  mode => '0644',
}
motd

include motd in site manifest

- # puppet agent -t

modify owner/group/mode

- # puppet agent -t; stat /etc/motd
- run puppet
- run it again
- nothing happened!!
- make a change
- run puppet
- ensures state
filesdemo module

Discuss module
resource defaults

File {
  owner => 'root',
  group => 'root',
  mode  => '0644',
}

test module

• refactor code to use resource defaults

• what about the directory? 0644 would not be good!
validation

- parser check
  
  - # puppet parser validate <file.pp>

- or use the bash function `pc` from `/root/.bashrc`

- style guide - [http://docs.puppetlabs.com/guides/style_guide.html](http://docs.puppetlabs.com/guides/style_guide.html)

  - # puppet-lint <file.pp>

- add these to your pre-commit scripts!
file serving

- create directory `/etc/puppet/modules/motd/files`

- create a file, `motd`, with some text
file { '/etc/motd':
  ensure => file,
  source => 'puppet:///modules/motd/motd',
  owner  => 'root',
  group  => 'root',
  mode   => '0644',
}
file serving

puppet:///modules/<module_name>/<file_name>

puppet looks for <module_name> in $modulepath and <file_name> under files directory.

This is how it finds /etc/puppet/modules/motd/files/motd
variables

# assign a value to variable
$variable = 'value'

# use a variable
notify { "variable is ${variable}": }

# facts are variables
notify { "my fqdn is ${::fqdn}": }
variables - arrays

# defining a variable with an array of values
$nameservers = ['4.2.2.1',
               '4.2.2.2',
               '8.8.8.8',
             ]
$message = "Welcome to ${::fqdn}.\nTry not to break anything"

file { '/etc/motd':
    ensure    => file,
    content   => $message
    owner     => 'root',
    group     => 'root',
    mode      => '0644',
}
templates

- uses the template() function
- which uses erb for the templating engine
- files go under `<module_name>/templates/` with .erb as the suffix
templates

- content => template('motd/motd.erb'),

- # puppet agent -t
templates

content => template('module_name/template.erb'),
puppet looks for <module_name> in $modulepath and <template.erb> under templates directory.
This is how it finds /etc/puppet/modules/motd/templates/motd.erb
template validation

- /usr/bin/erb -P -x -T '--' <file.erb> | /usr/bin/ruby -c

- or use the bash function `pt` from /root/.bashrc
# use a variable

<% = @message %>

templates
# iteration

<% @nameservers.each do |nameserver| -%>
nameserver <%= nameserver %>
<% end -%>
# conditionals

<% if @lsbdistid == 'CentOS' %>  
This system is running CentOS  
<% end %>
set a fact through the environment

# export FACTER_is_virtual = false

or

# export FACTER_virtual = kvm

Then run `puppet agent -t; cat /etc/motd`
ordering

- before and require
ordering - ntp module

- package{} then file{}
ordering - ntp module

- subscribe and notify
ordering - ntp module

- file{} then service{}
- restart service{} if any attribute of file{} changes
Without automation you would:

- # yum -y install ntp
- # vim /etc/ntp.conf
- # service ntpd start
- hope it works
# most common design pattern

```bash
package { 'ntp':
    ensure => present,
}

file { '/etc/ntp.conf':
    ...
    require => Package['ntp'],
}

service { 'ntpd':
    ...
    subscribe => File['/etc/ntp.conf'],
}
```
Parameterized Classes

# define a class

class say (    $msg,
  ) {    notify { “message = ${msg}”: }  }
}

# declare a class
# just like ‘include say’ except you can also specify msg

class {    ‘say’:    msg => ‘automation is fun’,  }

Parameterized Classes

- create say module
- declare say class in site manifest for one node
- include say class in site manifest for other node
Parameterized Classes

# oh noes!!
# we need a default

class say (
    $msg = 'boring default',
) {
    notify { "message = ${msg}": }
}
Hiera

- hierarchical data lookup system with pluggable backends
- separate data from code, yay!
- Defaults to YAML, also has
  - json
  - mysql
  - redis
  - gpg encrypted files
- CLI demo
Hiera

- can lookup parameters
- add to a hiera file
  - say::msg: from hiera
- run puppet
Custom facts

- write facts in ruby
- placed in lib/facter/factname.rb
Custom Facts

# role.rb
require 'facter'
Facter.add("role") do
  setcode do
    exec("cat /etc/role")
  end
end
Custom Facts

# fact for each cciss device
# example: cciss_c0d0 => present
#
require 'facter'
if File.exists?("/dev/cciss")
  Dir.foreach("/dev/cciss") {
    |entry|
    if entry =~ /^c[0-9]d[0-9]$/
      Facter.add("cciss_#{entry}") do
        # only run on systems where the kernel fact is linux
        confine :kernel => :linux
        setcode do
          "present"
        end
      end
  end
end
end
SSL

• on master
  • # puppet cert list
  • # puppet cert --list --all
  • # puppet cert --sign --all
  • # puppet cert --list --all

• on agent
  • # puppet agent -t
Exported Resources

Check out ssh module
/var/local/puppet-modules/modules/ssh/manifests/init.pp
Puppet Tutorial

OSCON
2013-07-22
Portland, OR

Garrett Honeycutt
Automation Consultant @ GH Solutions, LLC
gh@garretthoneycutt.com
http://linkedin.com/in/garretthoneycutt