Puppet Tutorial

LOAD
2013-04-07
Antwerpen, BE

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.

---

**DESIRED STATE**

<table>
<thead>
<tr>
<th>IT</th>
<th>ERATE AND INCREASE COVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>REA</td>
<td>TEND</td>
</tr>
</tbody>
</table>
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
# 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 Collector**
   (Puppet or 3rd party tool)
   - **Report**
     Puppet’s open API can also send data to third party tools.

SSL secure encryption on all data transport
Catalog

- Automatically maintained comprehensive resource list
- Easily validated against compliance requirements prior to client configuration
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
Reporting

- Comprehensive report of every change ever made, correlated to every resource being managed
Reporting

- http/https
- log
- store
- tagmail
- custom processors
  - irc
  - twitter
  - jabber
  - growl
Data Flow Technical View

Node

Get and evaluate facts

Apply catalog

Master

External node classifier (Dashboard)

Manifests + facts = catalog

Fact display (Dashboard)

Report display (Dashboard)

INITIAL CONTACT

PLUG-IN SYNC (facts, types, providers)

FACT CODE

SEND FACT VALUES

ASK CATALOG

SEND CATALOG

REQUEST FILES

SEND FILES

SEND REPORT
What not How

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

What not how
Example Resource Types

- cron
- exec
- file
- group
- host
- zfs
- mount
- package
- service
- sshkey
class ntp {
    package { 'ntp':
        ensure => installed,
    }

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

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

class motd {
    file { '/etc/motd':
        owner => 'root',
        group => 'root',
        mode => '0644',
        source => 'puppet:///modules/motd/generic_motd',
    }
}

# motd.erb

Welcome to <%= fqdn %>

kernel version = <%= kernelversion %>
puppet version = <%= puppetversion %>
facter version = <%= facterversion %>
Templates - Advanced

```erb
search <%= dnssearchpath %> 
options ndots:2 timeout:3
<% nameservers.each do |nameserver| -%>
  nameserver <%= nameserver %>
<% end -%>
```
Syntax Checking

$ puppet parser validate
Bootstrap Puppet Master VM

- ensure date is correct
  - # ntpdate us.pool.ntp.org

- set hostname
  - # hostname puppet.labs.priv && bash

- modify /etc/hosts
  - 1.2.3.4 puppet.labs.priv puppet
  - 10.10.10.10 yum.labs.priv yum
• install puppet-server
  • # yum -y install puppet-server

• start puppet master service
  • # service puppetmaster start

• run agent
  • # puppet agent -t
Bootstrap agent VM

- ensure date is correct
  - # ntpdate us.pool.ntp.org

- set hostname
  - # hostname yourname.labs.priv && bash
Bootstrap agent VM

- modify /etc/hosts on master and then scp to agent
  - 1.2.3.4 puppet.labs.priv puppet
  - 10.10.10.10 yum.labs.priv yum
  - 1.2.3.5 yourname.labs.priv yourname
Bootstrap agent VM

- run agent
  - # puppet agent -t
- oh noes!!!
SSL

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

- **on agent**
  - # puppet agent -t
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
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

- modify their password (hint use `grub-md5-crypt` to generate hash)

- modify their shell
• start and stop sysstat service
  • # puppet resource service sysstat ...
• /etc/puppet/puppet.conf

• # puppet config print

• # puppet config print |grep 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
modules

- 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
Forge

• install a package
  • # puppet module install ghoneycutt-dnsclient
site manifest

- `create /etc/puppet/manifests/site.pp`
- `add entries for both of your nodes`
• 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

file { ‘/etc/motd’:
    ensure => file,
}


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

- include motd in site manifest
  - # puppet agent -t

- mess with owner/group/mode
  - # puppet agent -t; ls -la /etc/motd
idempotency

- run puppet
- run it again
- nothing happened!!
- make a change
- run puppet
- ensures state
are your shell scripts idempotent?

• probably not, if you run it 2x... boom!
test module

• generate a new module called test

• associate it with your agent
file - directory

```plaintext
file { '/tmp/testdir':
  ensure => directory,
  owner  => 'root',
  group  => 'root',
  mode   => '0755',
}
```
test module

- add two files
  - /tmp/testdir/one
  - /tmp/testdir/two
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!
file - symlink

file { ‘/tmp/symlink’:
  ensure => symlink,
  target => ‘/tmp/testdir/one’,
}
validation

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

• style guide - http://docs.puppetlabs.com/guides/style_guide.html
  • # puppet-lint <file.pp>

• add these to your pre-commit scripts!
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

- copy /etc/puppet/modules/motd/files/motd to templates/motd.erb

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

- do an agent run
templates

color 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 '-' <filename.erb> | /usr/bin/ruby -c

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

# use a variable

<%= @message %>
templates

# iteration

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

# conditionals

<% if @lsbdistid == 'CentOS' %>
This system is running CentOS
<% end %>
templates

• refactor motd template and use the following.
  • variable interpolation
  • iteration
  • conditional logic
ordering

- before and require
- refactor test module to notify “first” and “after first” in correct order
ordering - ntp module

- generate ntp module
- copy /etc/ntp.conf to ntp/templates/ntp.conf.erb
- manage /etc/ntp.conf as a template
ordering - ntp

# manage service and ensure it happens before file{}

service { 'ntpd':
  ensure => running,
  enable => true,
  hasstatus => true,
  hasrestart => true,
  require => File[‘/etc/ntp.conf’],
}
ordering - ntp module

- subscribe and notify
- change service{} to use subscribe
- modify ntp.conf
  - echo "#junk >> /etc/ntp.conf"
- run puppet agent
Without automation you would:

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

package { ‘ntp’:
    ensure => present,
}

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

service { ‘ntpd’:
    ...
    subscribe => File[‘/etc/ntp.conf’],
}
Refactor ntp module to use this design pattern
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
# oh noes!!
# we need a default

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

- environment variables
- export FACTER_zzz=somevalue; facter zzz
Custom facts

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

```ruby
# role.rb
require 'facter'
Facter.add("role") do
  setcode do
    Facter::Util::Resolution.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
Puppet Tutorial

LOAD
2013-04-07
Antwerpen, BE

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