Romain Tartière

FreeBSD user since 2002
(I guess... FreeBSD 5.0-BETA1)
FreeBSD developer since 2010
(romain@)

Was a Systems Administrator for
HeathGrid working on EGI (European
Grid Infrastructure)

Discovered Puppet at that time
(~10 years ago... 0.25 -> 2.6)
Agenda

- Understanding how Puppet works
- Puppet from Zero to Hero
  - Installing
  - Managing Code
  - Organizing Code
  - Hiera
  - Custom Facts
  - PuppetDB
  - Orchestration

As soon as something is unclear, raise your hand!
Why would you use Puppet?

Automation!

Why automate?

- Consistency
- Predictability
- Reliability
- Speed
The Puppet Language

Declaring resources

user { 'romain':
    ensure => present,
    comment => '& Tartiere',
    shell    => '/usr/local/bin/zsh',
}
The Puppet Language

Variables

$motd = @("EOT")

  This is ${facts['networking']['fqdn']},
  running ${facts['os']['family']} ${facts['os']['architecture']}
  | EOT

file { '/etc/motd':
  ensure => file,
  owner => 'root',
  group => 'wheel',
  content => $motd,
}
The Puppet Language

Facts

Facts are collected by `facter(1)`.

```plaintext
# facter
[...]
os => {
    architecture => "amd64",
    family => "FreeBSD",
    hardware => "amd64",
    name => "FreeBSD",
    release => {
        full => "11.1-RELEASE-p10",
        major => "11",
        minor => "1-RELEASE-p10"
    }
}
[...]
```
The Puppet Language
Conditionals & functions

if versioncmp($foo_version, '1.0') >= 0 {
    service { 'foo':
        ensure => running,
        enable => true,
    }
}

$users = ['foo', 'bar', 'baz']
$users.each |$user| {
    file { "/home/${user}/.foorc":
        ensure => file,
        owner => $user,
        group => $user,
    }
}

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

  service { 'foo':
    ensure => running,
    enable => true,
  }

  Package['foo'] ~> Service['foo']
}

include foo
require foo
contain foo
class { 'foo': }
The Puppet Language
Defined classes

define root_file (  
    String $text,
  ) {
    file { "/${title}":
        ensure  => file,
        content => $text,
    }
}

root_file { 'LICENSE':  
    text => "BSD 2 clauses\n",
}
root_file { 'SYSADMINS':  
    text => "romain\n",
}
The Puppet Language

Node dependent resources

node 'foo.example.com' {
    file { '/usr/bin/rsh':
        ensure => absent,
    }
}

node /^foo-/ {
    include foo
}

node default {
    service { 'puppet':
        enable => true,
    }
}

FreeBSD
Modules

Adding some abstraction

Wrap all resources to manage something (e.g. *apache*, *postgresql*)

Abstracts OS-specific information, e.g.

- Service names;
- Package names;
- Configuration file paths;
- ...

---

FreeBSD
The Forge
Where to find modules

https://forge.puppet.com
Central repository for modules
5600+ modules available
430+ modules for managing ssh
Some authors do not publish their modules on the forge...
# pkg install puppet5

# puppet resource service puppet ensure=running enable=true
# pkg install puppetserver5

# puppet resource service puppetserver ensure=running enable=true

Hint: You may want to adjust puppetserver_login_class in /etc/rc.conf
Getting started
The first steps

Put your manifest files (*.pp) under
/usr/local/etc/puppet/environments/production/manifests/

Discover the Puppet language; experiment with modules

Hints:
- start with something you master
- start with something that applies to all your nodes (ssh, logging, monitoring, ...)
- stop as soon as you start to copy-paste code
Control repo

Manifests are code

Manifests are code is managed with a VCS

Template: https://github.com/puppetlabs/control-repo/

git branch ↔ Puppet environment

Default branch: production
Control repo

Deployment with R10K

Extracts each branch of the control repo in a separate directory

r10k deploy environment production -vp
puppet generate types --environment production

Hint: implement a post-receive hook
Roles and Profiles

Overview

<table>
<thead>
<tr>
<th>role::website</th>
<th>role::app</th>
<th>role::appapi</th>
<th>role::loadbalancer</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile::appli</td>
<td>profile::database</td>
<td>profile::webserver</td>
<td>profile::openssh</td>
</tr>
<tr>
<td>profile::logserver</td>
<td>profile::logclient</td>
<td>profile::</td>
<td>profile:::</td>
</tr>
<tr>
<td>apache</td>
<td>bacula</td>
<td>postgresql</td>
<td>ntp</td>
</tr>
<tr>
<td>riemann</td>
<td>haproxy</td>
<td>ssh</td>
<td>...</td>
</tr>
<tr>
<td>package</td>
<td>file</td>
<td>user</td>
<td>group</td>
</tr>
<tr>
<td>exec</td>
<td>sshkey</td>
<td>service</td>
<td>...</td>
</tr>
</tbody>
</table>
Find me in manifests/*.pp

node 'ns48724.example.com' {
    include role::website
}

default {
    include role::base
}
Roles and Profiles

Roles

Find me in site/role/manifests/*.pp

class role::base {
    include profile::openssh
    include profile::syslog
}

class role::website inherits role::base {
    include profile::webserver
    include profile::example_com_website
}

class role::product inherits role::base {
    include profile::database
    include profile::product_runner
}
Profiles

Find me in site/profile/manifests/*.pp

class profile::webserver {
  class { 'apache':
    default_vhost => false,
    default_mods => false,
    mpm_module => 'event',
    server_tokens => 'Prod',
  }

  class { 'apache::mod::ssl':
    ssl_cipher => 'HIGH:!aNULL:!MD5:!RC4',
    ssl_protocol => ['all', '-SSLv2', '-SSLv3', '-TLSv1', '-TLSv1.1'],
  }

  # ...
}

FreeBSD
Interlude
include vs. resource-style declaration

include apache

class { 'apache':
    mpm_module => 'event',
    server_tokens => 'Prod',
}

include apache

class { 'apache':
    mpm_module => 'prefork',
    server_tokens => 'Full',
}
Roles and Profiles
Profiles with parameters

class profile::mailserver (    
   Enum['relayhost', 'smarthost'] $configuration = 'smarthost',
 ) {    
   $listen_address = $configuration ? {    
       'relayhost' => [':1', '127.0.0.1'],
       'smarthost' => [':', '0.0.0.0'],
   }    
   # ...

class { 'postfix':    
   listen => $listen_address,
   # ...
 }
Think Facade and Adapter design patterns

A facade is used when a simple interface to a complex or difficult to understand system is desired. Interfaces may be incompatible, but the inner functionality should suit the need. The adapter design pattern allows otherwise incompatible classes to work together by converting the interface of one class into an interface expected by the client.
Roles and Profiles

Summary

Nodes
- include a single role

Roles
- include any number of profiles
- are named after business names

Profiles
- declare actual resources
- are named after technology stack
Used for *Automatic Parameter Lookup*

Configured in `hiera.yaml` and `data/**/*.yaml`

<table>
<thead>
<tr>
<th>alpha</th>
<th>beta</th>
<th>gamma</th>
<th>delta</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>nodes/%{facts.hostname}.yaml</td>
</tr>
<tr>
<td>dc1</td>
<td></td>
<td></td>
<td>dc/%{facts.datacenter}.yaml</td>
</tr>
<tr>
<td>n/a</td>
<td></td>
<td></td>
<td>common.yaml</td>
</tr>
</tbody>
</table>

---

`profile::mailserver::configuration: 'relayhost'`
Custom Facts

Helps classification

Room number (e.g. B21)

Encodes:
- Building (first letter)
- Floor (first digit)
- Actual number of the room (last digit)

Can be static or dynamically inferred from:
- hostname (e.g. b21-02)
- ipaddress (e.g. each room has its own IPv4 /24)
Can be set in /usr/local/etc/facter/facts.d/room.yaml:

```yaml
---
room: B21
building: B
floor: 2
room_number: 1
```
Custom Facts

Dynamic Facts

Usually set in a module in `<module>/lib/facter/room.rb`:

```ruby
Facter.add(:room) do
  setcode do
    if Facter.value('hostname').match(/\A([a-c]\d\d)-\d+-\d+/z/)
      $1.upcase
    end
  end
end

Facter.add(:building) do
  setcode do
    if room = Facter.value('room')
      room[0]
    else
      room[0]
    fi
  end
end
```
Custom Facts

Usually set in a module in `<module>/facts.d/room`:

```bash
#!/bin/sh
room=$(hostname | sed -o '^...' | tr 'a-z' 'A-Z')
set -- $(echo $room | sed -e 's/\(./\)/\1 /g')

cat <<EOT
room=$room
building=$1
floor=$2
room_number=$3
EOT
```
PuppetDB
Put Your Data to Work

Stores:
▶ Facts
▶ Catalogs
▶ Reports

Puppet Query Language

Allows exporting resources when configuring a node and collecting them on another node
Use cases: ssh keys fingerprints, backups, …
Configuration Management vs. Orchestration

The Marionette Collective
- A lot of options to choose from
- Usability depends on your choices
- Security depends on your choices

Choria
- Secure by default
- Easy to maintain
- Production ready
Get the WIP `sysutils/choria` port:
https://github.com/smortex/puppet5/

For assistance: `#choria` channel on `puppetcommunity` slack
https://puppetcommunity.slack.com/messages/C9KFTKRU3/
Jumping in!

Try it!
https://wiki.freebsd.org/Puppet/GettingStarted

Report success & failures to puppet@

For assistance: #freebsd channel on puppetcommunity slack
https://puppetcommunity.slack.com/messages/C6CK0UGB1/

As usual, Problem Reports are welcome!
Contributing with upstream

Most projects are public on GitHub: https://github.com/puppetlabs/

You’ll have to sign a Contributor License Agreement (CLA)

You’ll also need a Jira Account on https://tickets.puppetlabs.com/

Pull-Requests are merged
Thanks!