Puppet for Rubyists (a devop approach)

by Panagiotis Xinos

Who am I

- Unix Software Developer since ... (90 do the math)
- Been doing Linux (more or less) since k0.85
- Puppet experience since December 2011 (not that long I am afraid)
- contact @tehn00b

Typical Sysadmin @work

Devops?

- How computer farms (traditionally) work and evolve
- What was the need for a devop?
- Are we missing something?

The field (Platforms available)

- Puppet (ruby based)
- Chef (ruby extension)
- Capistrano (not a platform really)
- Fabric (Capistrano in python... not interested - heh)

What do I choose

Puppet

Why Puppet?

- Large Installation Base
- Large Developer Base
- Domain Language for the job
- Commercial Track Record
- Not bundled with ruby
- Multiple platforms

Why (not) Chef?

- Ruby driven (extends the language)
- The same model as puppet but tends to be complicated when not running ruby scripts
- Platform restricted

Why (not) Capistrano

- Does not do system awareness
- You have to write scripts that are not modular enough
- Nicer for deployments only (fails if one machine fails)

Why (not) Fabric

- Capistrano like
- ok it is python oriented (no offense but this is a presentation for rubyists)

Now what?

History of Puppet

- Puppet is principally developed by Luke Kanies and his company, Puppet Labs (formerly Reductive Labs).
- Unsatisfied with existing configuration management tools, Kanies began working with tool development in 2001 and in 2005 he founded Puppet Labs, an open source development house.

THE PUPPET MASTER

The tools

- Facter
- MCollective
- Puppet Dashboard
- Vagrant (we have to debug our recipes, no?)

Cut to the chase man...

Puppet Structure

Resources and the RAL

- Resource Abstraction Layer
- Resources ~= Class

Resource

```
$ puppet resource user root

user { 'root':
    home => '/var/root',
    shell => '/bin/sh',
    uid => '0',
    ensure => 'present',
    password => '*',
    gid => '0',
    comment => 'System Administrator'
}
```

Create a user

```
$ puppet resource user panos ensure=present shell="/
bin/zsh" home="/home/panos" managehome=true

notice: /User[panos]/ensure: created
user { 'panos':
    ensure => 'present',
    home => '/home/panos',
    shell => '/bin/zsh'
}

note* --edit opens an editor
```

The Core Resource Types

- notify
- file
- package
- service
- exec
- cron
- user
- group

Puppet ri equivalent

$ puppet describe -s user

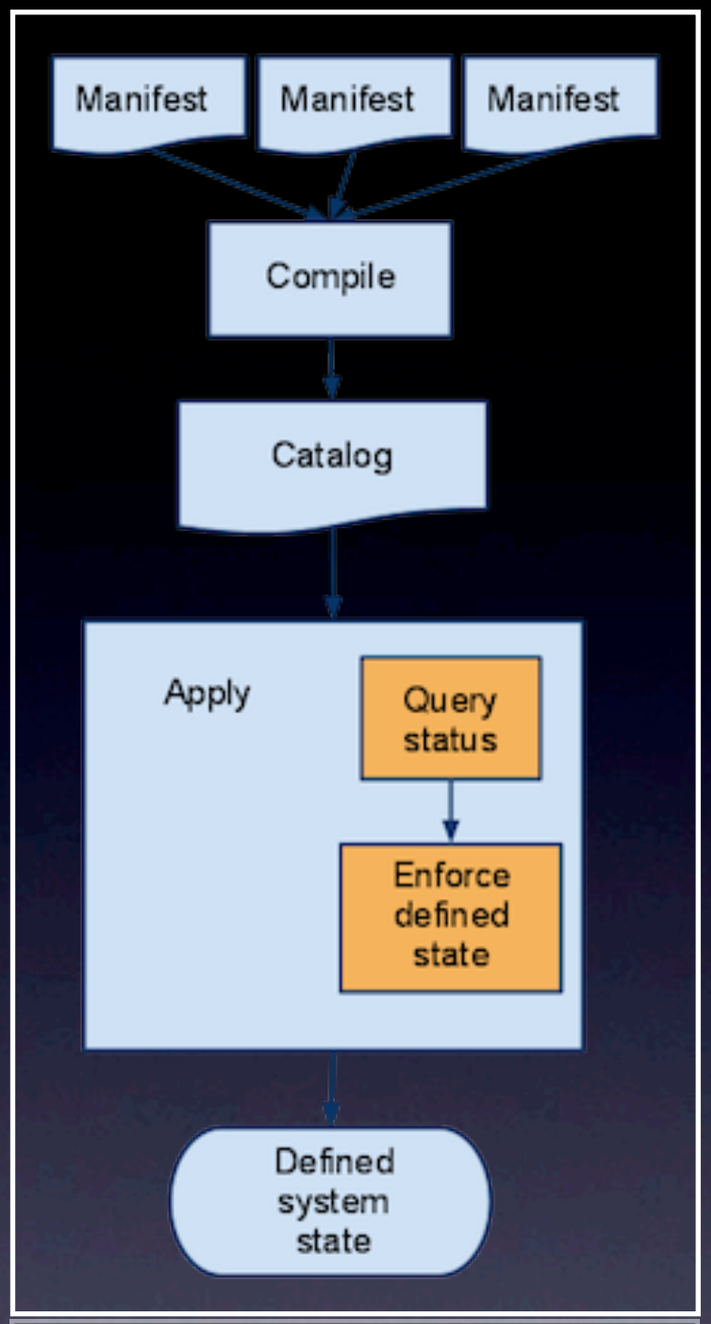
user
====
Manage users. This type is mostly built to manage system
users, so it is lacking some features useful for managing normal
users.
This resource type uses the prescribed native tools for creating
groups and generally uses POSIX APIs for retrieving information
about them. It does not directly modify `/etc/passwd` or anything.

Parameters
----
allowdupe, auth_membership, auths, comment, ensure, expiry, gid, groups,
home, key_membership, keys, managehome, membership, name, password,
password_max_age, password_min_age, profile_membership, profiles,
project, role_membership, roles, shell, uid

Providers
----
directoryservice, hpuxuseradd, ldap, pw, user_role_add, useradd

Manifests

puppet apply my_manifest.pp



What happened?

Declaring a Resource

```
file {'testfile':
    path    => '/tmp/testfile',
    ensure  => present,
    mode    => 0640,
    content => "I'm a test file.",
}
```

Apply Manifest

```
$ puppet apply 1.file.pp
notice: /Stage[main]//File[testfile]/ensure: created
```

```
$ cat /tmp/testfile
I'm a test file.

$ ls -lah /tmp/testfile
-rw-r----- 1 root root 16 Feb 23 13:15 /tmp/testfile
```

Notify Resource

```
notify {'title':
        message => "I'm notifying
you.",
    }
```

644 = 755 For Directories

- Puppet groups the read bit and the traverse bit for directories, which is almost always what you actually want

Metaparameters

- before
- require
- notify
- subscribe

Metaparameter Examples

```
file {'/tmp/test1':
   ensure  => present,
   content => "Hi.",
 }

 notify {'/tmp/test1 has already been synced.':
      require => File['/tmp/test1'],
    }

file {'/tmp/test1':
      ensure  => present,
      content => "Hi.",
      before  => Notify['/tmp/test1 has already been synced.'],
    }

    notify {'/tmp/test1 has already been synced.':}
```

Chaining

```
file {'/tmp/test1':
    ensure  => present,
    content => "Hi.",
    }

notify {'after':
      message => '/tmp/test1 has already been synced.',
    }

    File['/tmp/test1'] -> Notify['after']
```

More complicated

```
  file { '/etc/ssh/sshd_config':
  ensure => file,
  mode   => 600,
  source => '/root/files/sshd_config',
}
```

```
service { 'sshd':
  ensure     => running,
  enable     => true,
  hasrestart => true,
  hasstatus  => true,
  # FYI, those last two attributes default to false, since
  # bad init scripts are more or less endemic.
  subscribe  => File['/etc/ssh/sshd_config'],
}
```

The trio

```
package { 'openssh-server':
    ensure => present,
    before => File['/etc/ssh/sshd_config'],
  }

file { '/etc/ssh/sshd_config':
    ensure => file,
    mode   => 600,
    source => '/root/files/sshd_config',
  }

service { 'sshd':
    ensure     => running,
    enable     => true,
    hasrestart => true,
    hasstatus  => true,
    subscribe  => File['/etc/ssh/sshd_config'],
  }
```

Variables

```
$sshkey = "id-rsa ..............."

file {'authorized_keys':
      path    => '/root/.ssh/authorized_keys',
      content => $sshkey,
    }
```

Facts

```
host {'self':
   ensure       => present,
   name         => $fqdn,
   host_aliases => ['puppet', $hostname],
   ip           => $ipaddress,
    }

file {'motd':
      ensure  => file,
      path    => '/etc/motd',
      mode    => 0644,
      content => "Welcome to ${hostname},\na ${operatingsystem} member of $
{domain}.\n",
    }
```

Conditional Statements

```
if $is_virtual == 'true' {
   service {'ntpd':
     ensure => stopped,
     enable => false,
   }
    }
else {
      service { 'ntpd':
        name       => 'ntpd',
        ensure     => running,
        enable     => true,
        hasrestart => true,
        require => Package['ntp'],
      }
    }
```

What is false?

- undef
- “ (empty string)
- false
- any expression evaluated to false

Case

```
case $operatingsystem {
   centos: { $apache = "httpd" }
   # Note that these matches are case-insensitive.
   redhat: { $apache = "httpd" }
   debian: { $apache = "apache2" }
   ubuntu: { $apache = "apache2" }
   default: { fail("Unrecognized operating system for webserver") }
   # "fail" is a function.
 }
package {'apache':
   name   => $apache,
   ensure => latest,
 }
```

Case with regex

```
case $ipaddress_eth0 {
    /^127[\d.]+$/: {
        notify {'misconfig':
          message => "Possible network misconfiguration:
IP address of $0",
        }
      }
    }
```

Optimized Case

```
case $operatingsystem {
        centos, redhat: { $apache = "httpd" }
        debian, ubuntu: { $apache = "apache2" }
        default: { fail("Unrecognized operating system for
webserver") }
    }
```

Selectors

```
$apache = $operatingsystem ? {
    centos                  => 'httpd',
    redhat                  => 'httpd',
    /(?i)(ubuntu|debian)/ => "apache2-$1",
      # (Don't actually use that package name.)
    default                 => undef,
}
```

Classes! (finally)

```
class ntp {
      case $operatingsystem {
        centos, redhat: {
          $service_name = 'ntpd'
          $conf_file    = 'ntp.conf.el'
        }
        debian, ubuntu: {
          $service_name = 'ntp'
          $conf_file    = 'ntp.conf.debian'
        }
      }

      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  => "/root/learning-manifests/$
{conf_file}",
      }
}
```

Classes

- start with lowercase letters
- can use double colon ::
- must be declared (more on next slide)

Class declaration

- class {'ntp': }
- include ntp - if declared don’t redeclare

Modules

Module structure

`my_module` — This outermost directory's name matches the name of the module.

- `manifests/` — Contains all of the manifests in the module.
  - `init.pp` — Contains a class definition. **This class's name must match the module's name.**
  - `other_class.pp` — Contains a class named `my_module::other_class`.
  - `my_defined_type.pp` — Contains a defined type named `my_module::my_defined_type`.
  - `implementation/` — This directory's name affects the class names beneath it.
    - `foo.pp` — Contains a class named `my_module::implementation::foo`.
    - `bar.pp` — Contains a class named `my_module::implementation::bar`.
- `files/` — Contains static files, which managed nodes can download.
- `lib/` — Contains plugins, like custom facts and custom resource types.
- `templates/` — Contains templates, which can be referenced from the module's manifests.
- `tests/` — Contains examples showing how to declare the module's classes and defined types.

Templates (they will look familiar trust me)

Rendering a template

```
file {'/etc/foo.conf':
  ensure  => file,
  require => Package['foo'],
  content => template('foo/foo.conf.erb'),
}
```

Expressions!

```
<%= sectionheader %>
      environment = <%= gitrevision[0,5] %>

<% servers_real.each do |server| -%>
    server <%= server %>
<% end -%>
```

Parameterized Classes

```
class ntp ($servers = undef) {
    ...
```

```
class {'ntp':}

class {'ntp':
      servers => [ "ntp1.example.com dynamic",
"ntp2.example.com dynamic", ],
}
```

Defined Types

Defining a Type

- A name
- List of parameters
- Collection of resources

```
define planfile ($user = $title, $content) {
    file {"/home/${user}/.plan":
      ensure  => file,
      content => $content,
      mode    => 0644,
      owner   => $user,
      require => User[$user],
    }
  }

user {'panos':
  ensure     => present,
  managehome => true,
  uid        => 1000,
}

planfile {'panos':
  content => "Working on a Puppet Presentation.",
}
```

WOW now I know Puppet!

NO

Agents and Puppet Master

gem install puppet

facter

gem install puppet

puppetmaster facter

ENOUGH! (for now)

Questions?

Thank you