Puppet Workshop, FrOSCamp 2010

Marc Fournier, Pascal Simon & Andreas Zuber

September 18, 2010

Camptocamp SA & Puzzle ITC

Manual configuration

Login and "Just Do It" !

⇒ doesn't scale
⇒ boring, careless mistakes
⇒ no log of changes
⇒ not reproducible
⇒ requires great rigour (especially in a team) !

Centralized configuration

- shell, perl scripts, Makefiles
- file deployment with rsync
- centralize storage / authentication
- `for i in $SERVER_LIST ; do ssh root@${i} ... ; done`

⇒ error handling ?
⇒ exceptions ?
⇒ logs ?
⇒ undo ?
⇒ different versions ? different OS ? different architecture ?
⇒ "home made" solutions, not very flexible, don't work elsewhere.

Installation procedures

- Document installation and configuration recipies (howtos, wikis, etc)
- step by step procedures, checklists

$\Rightarrow$ quickly obsolete
$\Rightarrow$ 90% of executable commands
$\Rightarrow$ copy-n-paste man
$\Rightarrow$ fire fighting changes don't end up in the doc.

Automation, rationalization

- automated installation (FAI, kickstart, jumpstart, etc)
- template cloning, partimage, Norton Ghost, etc.

$\Rightarrow$ no maintenance / manual maintenance
$\Rightarrow$ template maintenance
$\Rightarrow$ and how does this cope with change ?

Growth

- OS & servers are now disposable
- serveurs number growth (virtualization).

$\Rightarrow$ more sysadmins ?
$\Rightarrow$ more extra-hours ?

Inventory, documentation

- inventory is mandatory
- infrastructure documentation.

$\Rightarrow$ burdensome, boring, never a priority
$\Rightarrow$ quickly out of date / out of sync.

Conclusion

Need for:

- system configuration management
- automatization of maintenance
- authoritative recipies
- don't do the same thing twice (DRY)
- facilitate reuse of know-how
- . . . control entropy and chaos in the server cluster !

James White Infrastructure Manifesto (excerpt)

- There is one system, not a collection of systems
- The actual state of the system must self-correct to the desired state
- The only authoritative source for the actual state of the system is the system
- The entire system must be deployable using source media and text files
- Do not use any product with configurations that are not machine parsable and machine writeable
- Do not improve manual processes if you can automate them instead

⇒ `http://loki.websages.com/ws/`


Puppet

Puppet

Puppet is:

- resource management
- a client-server framework
- a "programming" language
- a toolbox for the sysadmin
- OSS written in Ruby
- runs on most modern Unixes

Puppet

Puppet is not:

- just a bunch of scripts
- an inventory tool
- a software distribution service / fileserver
- a replacement for FAI / kickstart / jumpstart
- (necessarily) used to manage everything
- a goal in itself
- a good reason to slack !

Setup your environment

```
GEM_HOME="~/workshop" gem install puppet --version=0.25.5 \
--no-rdoc --no-ri
cd ~/workshop/
GEM_PATH="." bin/puppet --version

GEM_PATH="." bin/puppetdoc --all --mode pdf
evince /tmp/puppetdoc.pdf
GEM_PATH="." bin/pi -l
GEM_PATH="." bin/pi -p file
```

http://docs.puppetlabs.com/guides/language_tutorial.html

Language features

- declarative, idempotent (Makefile)
- **what** not **how**
- allows to **describe** collections of ressources
- reduced and easy syntax.

Resources

Describe properties of different types of objects:

- user
- file
- package
- cron
- etc.

⇒ create if missing, correct if different
⇒ not only for files or packages !
⇒ abstract OS/distrib specificities
⇒ declared inside "manifests".

Resource example

```
file { "/etc/resolv.conf":
  ensure  => present,
  owner   => "root",
  group   => "root",
  mode    => 0644,
  content => "
    search camptocamp.com
    nameserver 10.27.21.1
    nameserver 10.26.21.1
  ",
}
```

Resources: dependencies

```
package { ["apache", "tomcat"]: ensure => installed }

service { "apache":
  ensure  => running,
  require => Package["apache"],
}

service { "tomcat":
  ensure  => running,
  require => Package["tomcat"],
  before  => Service["apache"],
}
```

Resources: notification

```
file { "/etc/apache/httpd.conf":
  ensure  => present,
  content => template("example.com/httpd.conf.erb"),
  notify  => Service["apache"],
}

service { "apache":
  ensure  => running,
  restart => "apachectl configtest && apachectl graceful",
}
```

Attributes, conditions

```
if ( $use_nagios == "true" ) {

  $warn = $processorcount * 3
  $crit = $processorcount * 6

  monitoring::check { "Load Average":
    command => "check_load",
    options => "-w ${warn} -c ${crit}",
  }

}
```

Classes

organizing resources, inheritance:

```
class apache {
  package { "apache": ensure => present }

  service { "apache":
    ensure  => running,
    require => Package["apache"],
  }

  file { "/var/www":
    ensure => directory,
  }
}
```

Definitions

```
define apache::vhost ($ensure=present, $source) {

  file { "/etc/apache2/sites-enabled/${name}.conf":
    ensure  => $ensure,
    content => template("apache/vhost.conf.erb"),
    notify  => Service["apache"],
  }

  file { "/var/www/$name":
    ensure  => directory,
    source  => $source,
    recurse => true,
  }

}
```

Nodes

Declaration of classes/definitions to apply on a machine:

```
node "websrv1.example.com" {
  include apache
  include mysql
  include php

  apache::vhost { "www.example.com":
     ensure => present,
     source => "puppet:///web/example.com/htdocs/",
  }
}
```

Client characteristics (puppet & puppetd)

- puppet code parser
- run options: agent or standalone (default: 2x/h)
- sends system "facts" to the server
- recieves it's config from the server in a catalog.

Facter

- system characteristics
- available as variables in the puppet language
- easy to extend, Ruby.

Facter: exemple

```
root@example:~# facter | grep operating
operatingsystem => RedHat
operatingsystemrelease => 5
```

Using facts

```
case $operatingsystem {
  RedHat: { include apache-redhat }
  Debian: { include apache-debian }
}
```

A small exercise

Create a file containing easy C code in /tmp, compile it and run it. Ensure gcc is installed. All this using puppet.

The file:

```
#include <stdio.h>

void main() {
  printf("Hello !\n");
}
```

The compilation command:

```
gcc -o /tmp/hello /tmp/hello.c
```

same exercise, a bit more complicated

additional constraints:

- the manifest must work on FreeBSD too (gcc is in /usr/local/bin)
- the program must print the puppet version number instead of "Hello"
- the program must get compiled and run only if the source file changes

Server characteristics (puppetmasterd)

- authenticate clients with SSL certificates
- compiles every client resources in catalogs
- node definition (text file, ldap or script)
- logs all operations
- fileserver
- template engine.

And most importantly:

- centralizes each clients facts in a DB
- centralizes and re-exports client resources.

"Companion tools"

- foreman / dashboard – web interface
- puppetdoc – documentation generation
- puppetrun – force run from the serveur
- puppetca – SSL certs management
- puppet – CLI parser
- ralsh – resources $\Rightarrow$ puppet
- pi – language reference
- $\Rightarrow$ recent versions: single binary.

Advanced stuff

- environments
- exported resources
- extending facter
- Ruby DSL
- `--graph`
- mcollective

Common pitfalls

- unpredictable ordering != a bug
- no use of `--parseonly` and `--noop`
- declarative language: no loops
- variables do not vary
- SSL certificates, avoid problems by:
  - `hostname --fqdn` must match DNS hostname
  - call the puppetmaster `puppet.yourdomain.tld`
- "not a bunch o'scripts": a lot of exec's is bad sign
- `perl -pie / sed -i / awk`: use augeas or templates
- monster modules: write custom ruby providers
- webrick doesn't scale: use mongrel/passenger

Centralized, rebuildable, auto-generated

The puppet server consolidates and centralizes node configuration.

- inventory / directory
- configuration of central services (DNS, monitoring, stats, backups, etc)
- know-how is in one place
- documentation & changelog
- all this stuff is autopopulated and rebuildable from scratch

Programmable infrastructure

Development paradigm can be applied to infrastructure:

- separation of generic/specific code, library of functions
- open-source code
- VCS imposed discipline: diff, branch, revert, blame, etc
- workflow: dev $\Rightarrow$ test $\Rightarrow$ prod
- code review
- inline documentation
- methodologies (XP, Agile, Scrum)
- unit & functional testing, continuous integration.

Benefits

- consistent configurations
- no manual operation in production
- recipies reproducible in a reliable and precise way
- totally automated (cloud computing, diskless servers)
- mutualization and sharing of sysadmin know-how.

$\Rightarrow$ cf. James White Manifesto

Future

- more resource types
- facts in other languages
- more complex data structures in facts
- inter-node dependencies
- jruby
- windows & other peripheral support

Resources

- http://docs.puppetlabs.com/
- puppet-users@googlegroups.com
- #puppet on irc.freenode.net
- "Pulling Strings with Puppet", Apress
- http://puppetlabs.com/training/
- http://spug.ch/
- http://forge.puppetlabs.com/
- http://puppet-modules.git.puzzle.ch/
- http://github.com/camptocamp/

Meet-ups

- puppetcamp America: 7-8 october, San Francisco
- puppetcamp Europe: spring 2011
- devops days: 15-16 october, Hamburg
- next SPUG meeting: 17 november, Bern (check RSS feed/mailing-list)

Conclusion

⇒ puppet is resource management
⇒ need for industrialization and automation of infrastructure management
⇒ quality, consistency, integrity: not feasable manually, at scale
⇒ puppet describes, distributes and applies configurations
⇒ declares the state of the systems as a whole, using code
⇒ mutualization and know-how sharing.

Questions ?

These slides on `http://spug.ch/`