PUPPET

Romain Bélorgey IR3

Ingénieurs 2000

SOMMAIRE

- Qu’est-ce que Puppet ?
- Utilisation de Puppet
- Ses composants
- Son utilisation
- Le langage Puppet
- Démonstration

QU'EST-CE QUE PUPPET ?

- Administration centralisée de machines
- Outil open-source écrit en Ruby
- Fonctionne sur tous les systèmes d'exploitation Unix
- Fonctionnement de type client/serveur
- Un langage pour déclarer la configuration du système
- La couche abstraite entre l'administrateur système et son systeme

UTILISATION DE PUPPET

“People are finally figuring out puppet and how it gets you to the pub by 4pm. Note that I’ve been at this pub since 2pm.”

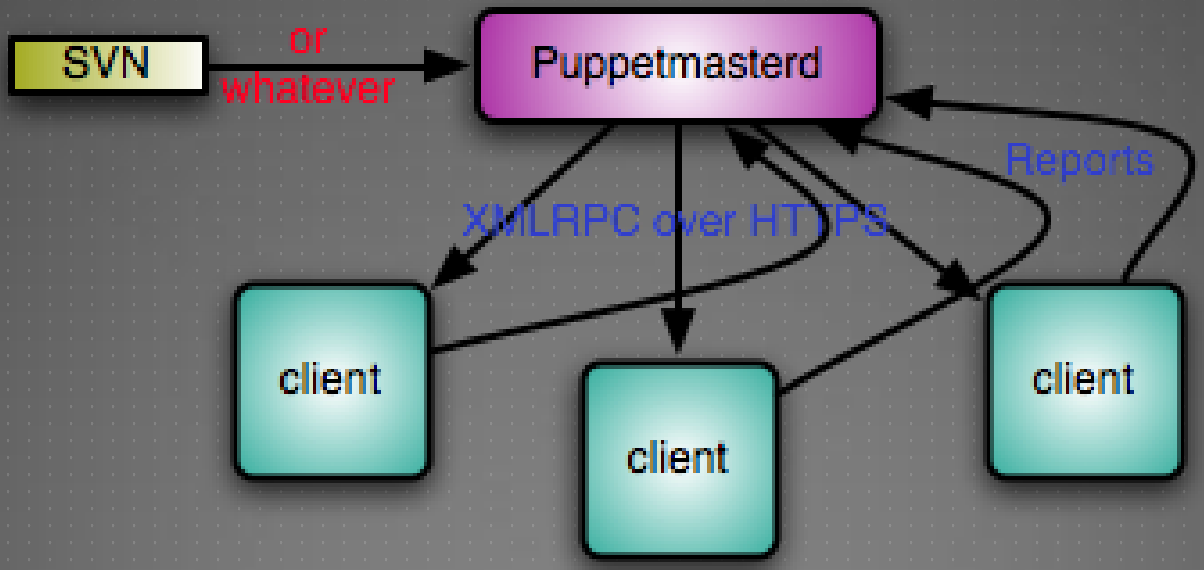
Jorge Castro

UTILISATION DE PUPPET

- Dans de grandes sociétés : Google, SUN, Twitter, Citrix, …
- Dans plusieurs projets de mon entreprise
- Par des administrateurs et ingénieurs systèmes

- Simplifier la gestion d’un parc de machines
- Centraliser les configurations et les modifications
- Le lien entre programmation et administration système

SES COMPOSANTS



SES COMPOSANTS

- Un serveur : puppetmasterd
- Des fichiers de configuration
- Un client : puppetd
- Un validateur d'identités : puppetca
- Une console : ralsh ou puppet ressource

- Une interface : puppet daschboard
- Une librairie de modules : forge
- Une api pour recupérer des informations sur les clients

CONFIGURATION DE PUPPET

- Installation et configuration du serveur
- Installation du client sur la machine à configurer
- Autorisation de cette dernière auprès du serveur

- Spécification de la configuration du client sur le serveur
- Le client vérifie sa configuration toutes les 30mns

LANGAGE DE PUPPET

1. Puppet types
2. La gestion des dépendances
3. La gestion de templates
4. Les nodes
5. Les classes
6. Spécification de nouvelles commandes
7. Exemples de configuration

PUPPET TYPES

- Eléments utiles pour la configuration de puppet
- Files (contenu et droit d'accès)
- Packages (ensure installed ou absent)
- Services (enabled/disabled, running/stopped)
- Exec (exécution de commandes)
- Et bien d'autres : cron, user, group, host, interface, k5login, mailalias, maillist, mount, nagios*, …

EXEMPLE: FILE

```
file { "/etc/collectd/collectd.conf":
        owner => root,
        group => root,
        mode => 644,
        require => Package["collectd"],
content => template("/etc/puppet/files/collectd.conf")
 }
```

GESTION DES DÉPENDANCES

- Installation d’un paquet
- Spécification de son fichier de configuration si nécessaire
- Lancement du daemon
- D’autres actions sont possibles

GESTION DES DÉPENDANCES

```
package { ["haproxy"]:
                ensure => installed;
}

file { "/etc/haproxy/haproxy.cfg":
                owner => root,
                group => root,
                mode => 644,
                source => "puppet:///files/loadbalancer/haproxy.cfg",
                notify => Service["haproxy"],
}
service { "haproxy":
                ensure => running,
                provider => debian,
                hasstatus => true,
                hasrestart => true,
                subscribe => File["/etc/haproxy/haproxy.cfg"],
                require => [ Package["haproxy"], File["/etc/haproxy/haproxy.cfg"] ],
}
```

GESTION DES DÉPENDANCES

- Plus complexe :

```
service { "selenium":
        ensure => running,
        provider => debian,
        hasstatus => true,
        hasrestart => true,
        subscribe => File["/etc/init.d/selenium"],
        require => [ Package["sun-java6-bin"], File["/var/tmp/firefox"],
Service["xvfb"], File["/etc/init.d/selenium"],
File["/home/selenium/selenium-server.jar"], Service["monit"] ],
}
```

GESTION DES TEMPLATES

- Spécification d’un fichier soit par une simple copie soit par des templates
- Fichier identique à plusieurs machines sauf pour certains paramètres
- Réduction du nombre de fichiers
- Spécification des paramètres lors de l’appel de la template

GESTION DES TEMPLATES

- Sans template :

```
file { "/etc/haproxy/haproxy.cfg":
        owner => root,
        group => root,
        mode => 644,
        source =>
"puppet:///files/loadbalancer/haproxy.cfg",
        notify => Service["haproxy"],
}
```

GESTION DES TEMPLATES

- Appel de la template :

```
file { "/etc/ssh/sshd_config":
        owner => root,
        group => root,
        mode => 644,
        require => Package["openssh-server"],
        content => template(“chemin/sshd_config")
}
```

GESTION DES TEMPLATES

- Contenu template :

```
# Package generated configuration file
# See the sshd(8) manpage for details

# What ports, IPs and protocols we listen for
Port 22
# Use these options to restrict which interfaces/protocols sshd
will bind to
#ListenAddress ::
ListenAddress <%= ip_dom0 %>
Protocol 2
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
#Privilege Separation is turned on for security
UsePrivilegeSeparation yes
```

LES NODES

- Correspondent à une machine physique
- Une machine doit avoir un node lui correspondant
- Sinon créer un node “default”
- Peuvent include d’autres nodes ou classes
- Contiennent du langage Puppet

LES NODES : EXEMPLE

```
node ‘machine' inherits autre_node {
        include classe
        $variable=“specifique"
        /*code*/
}
```

LES CLASSES

- Contiennent du langage puppet
- Peuvent hériter d’autres classes
- Peuvent être incluses dans un node
- Contenu plus orienté applications

LES CLASSES : EXEMPLE

```
class collectd {
        package { ["collectd"]:
                ensure => installed;
        }
        file { "/etc/collectd/collectd.conf":
                owner => root,
                group => root,
                mode => 644,
                require => Package["collectd"],
                content => template("/etc/puppet/files/collectd.conf")
         }

        service { "collectd" :
                provider => "debian",
                ensure => running,
                require => Package["collectd"],
                subscribe => File["/etc/collectd/collectd.conf"]
        }
}
```

CRÉATION D'UNE COMMANDE

- Avoir le moins de répétitions
- Utilisation de Define
- Specification des paramètres entre parenthèses
- Puis commande(s) à exécuter entre accolades

CRÉATION D'UNE COMMANDE

```
define download_file( $site="", $cwd="", $creates="", $req="", $user="") {

        exec { $name:
                command => "wget ${site}/${name}",
                cwd => $cwd,
                creates => "${cwd}/${name}",
                require => $req,
                user => $user,
                path => ["/usr/bin", "/usr/sbin"],
        }
}
```

CRÉATION D'UNE COMMANDE

- Appel de la nouvelle commande :

```
download_file { “nom_fichier”:
        source => http://$puppet_server,
        cwd => “directory”,
        creates => “/directory/$name”,
        req => File[“something”]
        user => “root”,
}
```

DIFFÉRENTS EXEMPLES

- Création d’un utilisateur
- Mise en place d’une tâche régulière
- Exécution d’une tâche
- Bien d’autres trouvables sur puppetlabs.com

CRÉATION D'UN UTILISATEUR

```
user { dr:
                home => "/home/dr",
                shell => "/bin/bash",
                password => “password",
                allowdupe => false,
                managehome => true,
                ensure => present,
}
```

- Possibilité de spécifier le groupe
- Le password peut être protégé

CRON : TÂCHE REGULIÈRE

```
cron { rebootselenium:
                command => "/etc/init.d/selenium restart",
                user => root,
                hour => 6,
                minute => 0,
                require => File["/etc/init.d/selenium"],
}
```

EXÉCUTION D'UNE TÂCHE

```
exec { "Set MySQL server root password":
    subscribe => [ Package["mysql-server"], Package["mysql-client"] ],
    refreshonly => true,
    unless => "mysqladmin -uroot -p$password status",
    path => "/bin:/usr/bin",
    command => "mysqladmin -uroot password $password",
  }
```

PLUS D'INFORMATIONS

- Documentation disponible sur http://www.puppetlabs.com

DÉMONSTRATION

- Installation d’un serveur
- Installation d’un client
- Configuration des deux démons
- Autorisation du client
- Prise en compte d’une configuration ou modification

INSTALLATION DU SERVEUR

- Aptitude install puppetmaster
- Installera aussi puppet

```
master:~# aptitude search puppet
i A puppet
i   puppetmaster
master:~#
```

INSTALLATION DU CLIENT

- Aptitude install puppet

slave:~# aptitude search puppet

i puppet

slave:~#

CONFIGURATION DES DÉMONS

- Fichier de conf commun : /etc/puppet/puppet.conf
- Rien à modifier pour le serveur
- Spécifier le serveur pour le client :

...

[puppet]

server = nameoftheserver

AUTORISATION DU CLIENT

- Vérification de l'accès à chaque machine par leur nom
  - DNS ou fichier /etc/hosts
- Autorisation de l'accès aux fichiers de configurations
  - Dans le ficheir fileserver.conf sur le serveur :

[files]

path /etc/puppet/files

allow 192.168.0.*

allow 10.101.0.*

…

AUTORISATION DU CLIENT

- Certification de la connection entre le serveur et le client
- Etape la plus problématique
- Utiliser la commande puppetca sur le serveur
- Possibilité d’une auto-certification (dans le fichier de conf du serveur) :

[ca]

autosign = true

CONFIGURATION ET MODIFICATION

- Apache :

```
Class apache {
        package { [“apache2”,”php5”]
                ensure => installed;
        }
        service {“apache2”:
                ensure => running,
                hasstatus => true,
                hasrestart => true,
                require => Package[“apache2”],
        }
}
```

CONFIGURATION ET MODIFICATION

- Création du node :

```
node slave {
        include apache2;
}
```

- Vérification du bon lancement de apache2 :

```
#puppetd --no-daemon –verbose
#ps-edf | grep apache
```

CONCLUSION

- Outil simple à mettre en place
- Langage intuitif
- Amélioration de l’administration système
- Une communauté prête à améliorer l’outil

SOURCES

- http://www.puppetlabs.com
- Différentes presentations de Ohad Levy et Jorge Castor
- http://www.example42.com

- Avez-vous des questions ?