

Module Exposé: Puppet

Victor Martin, Sébastien Michaux et Riwan Blondé

IUT Nancy-Charlemagne

02 décembre 2011

Plan

① Puppet Labs

Introduction

Fonction principale

Commencer avec Puppet

② Utilisation

Le langage Ruby

Manifests

Modules

Complement à Puppet

③ Evaluation et sources

Avantages

Inconvénients

Sources

Puppet Labs

Introduction

- Fonction principale
- Commencer avec Puppet

Utilisation

- Le langage Ruby
- Manifests
- Modules
- Complément à Puppet

Evaluation et sources

- Avantages
- Inconvénients
- Sources

Qu'est ce que Puppet Labs ?

Qu'est ce que Puppet Labs ?

- un logiciel open source,

Qu'est ce que Puppet Labs ?

- un logiciel open source,
- basé sur le langage de programmation « Ruby »,

Qu'est ce que Puppet Labs ?

- un logiciel open source,
- basé sur le langage de programmation « Ruby »,
- sous licence GPL v2,

Qu'est ce que Puppet Labs ?

- un logiciel open source,
- basé sur le langage de programmation « Ruby »,
- sous licence GPL v2,
- développé par Luke Kanies et son entreprise Puppet Labs.

Qu'est ce que Puppet Labs ?

- un logiciel open source,
- basé sur le langage de programmation « Ruby »,
- sous licence GPL v2,
- développé par Luke Kanies et son entreprise Puppet Labs.
- Son modèle est basé sur 3 piliers :
 - ① Le déploiement
 - ② Un langage de configuration et une couche d'abstraction
 - ③ Sa couche transactionnelle

Déploiement

- Déployé sur un simple modèle client/serveur,
- Le Master s'exécute comme un démon sur un hôte et contient la configuration requise pour votre environnement.
- vérifie toutes les 30minutes le Master afin de voir si des modifications doivent être effectuées.

Configuration Lang/Rsrc

- Puppet utilise son propre langage de déclaration pour définir vos points de configuration qui sont écrits dans une « Ressource ».
- déclarer l'état final voulu de ses hôtes : les packages à installer, les services à exécuter etc.
- La plupart des outils de configuration (scripts shell ou perl par exemple) sont procéduraux. Ils décrivent comment les choses doivent être faites plutôt que de se focaliser sur l'état final attendu.

Transaction Layer

- Le moteur de puppet est sa couche transactionnelle.
- Une transaction puppet :
 - ① Interprète et compile votre configuration
 - ② Communique la configuration compilée à l'agent
 - ③ Applique la configuration sur l'agent
 - ④ Rapporte le résultat de cette application au master
- La couche transactionnelle permet aux configurations d'être créées et appliquées indéfiniment sur un hôte.

Installation et configuration

Côté serveur : Installation

`sudo apt-get install puppet puppetmaster facter`

- **Package puppet** : Ce paquet contient le script de démarrage et les scripts de compatibilité pour l'agent puppet, qui est le processus responsable de la configuration du nœud local.
- **Package puppetmaster** : Ce paquet contient le script de démarrage et les scripts de compatibilité pour le processus maître puppet, qui est le serveur hébergeant les manifests et les fichiers pour les nœuds puppet.
- **Package facter** : Bibliothèque dit « cross-platform », qui permet de collecter des informations sur le système. Particulièrement utile dans la collecte d'un nom de système, d'adresse IP et/ou MAC, clé SSH.

Installation et configuration

Côté serveur : Configuration

Le premier fichier à modifier, on y ajoute à la fin les lignes suivantes :

```
# /etc/puppet/puppet.conf  
[master]  
certname=puppetlabs
```

Puis un second fichier :

```
# /ect/hosts  
127.0.0.1 localhost  
192.168.1.67 puppetlabs
```

Installation et configuration

Côté client : Installation

Comme pour le serveur, nous utilisons l'APT pour installer l'agent Puppet via les packages **Puppet** et **Facter**.

Côté serveur : Configuration

```
# /etc/puppet/puppet.conf  
[main]  
server=puppetlabs
```

```
# /ect/hosts  
127.0.0.1 localhost  
192.168.1.188 puppetclient  
192.168.1.67 puppetlabs
```

Synchronisation

Sur le serveur :

```
sudo /etc/init.d/puppetmaster {start | stop | restart}
```

Sur le client :

```
sudo puppet agent --test
```

On obtient alors une demande de validation de certificat :

```
info: Creating a new certificate request for puppetcl
info: Creating a new SSL key at /var/lib/puppet/ssl/p
warning: peer certificate won't be verified in this S
notice: Did not receive certificate
```

Sur le serveur :

```
sudo puppet cert --list
sudo puppet cert --sign puppetclient
```

Synchronisation

Une fois le certificate distribué, on relance la commande :

```
notice: Got signed certificate
notice Stating Puppet client
err: Could not retrieve catalog: Could not find default
with 'puppetclient' on node puppetclient.
```

L'agent du client a bien été connecté avec le serveur mais n'a pas pu trouver un catalogue pour le dit client.

Plan

- 1 Puppet Labs
 - Introduction
 - Fonction principale
 - Commencer avec Puppet
- 2 Utilisation
 - Le langage Ruby
 - Manifests
 - Modules
 - Complement à Puppet
- 3 Evaluation et sources
 - Avantages
 - Inconvénients
 - Sources

Le langage Ruby

Ruby est un langage de programmation libre créé par Yukihiro Matsumoto, dont il publie une première version en 1996. Ruby est fortement orienté objet :

- toute donnée est un objet, y compris les types ;
- toute fonction est une méthode ;
- toute variable est une référence à un objet.

IRB, l'interpréteur de ruby, fonctionne sur de nombreux systèmes d'exploitation : UNIX, Linux, Microsoft Windows, etc.

Manifests

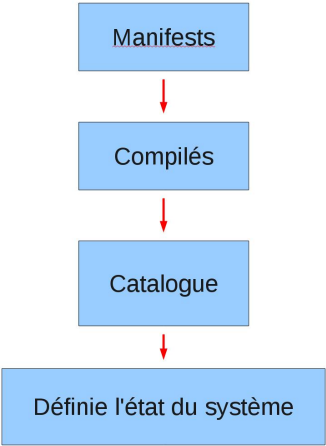
Les programmes de Puppet sont appelés «manifests», et ils utilisent l'extension de fichier .pp.

les manifests sont compilés dans le catalogue, qui est un graphe acyclique dirigé qui ne représente que les ressources et l'ordre dans lequel elles doivent être synchronisées.

Deux fichiers sont importants :

- ① /etc/puppet/manifests/site.pp
- ② /etc/puppet/manifest/node.pp

Manifests



Ressources

Une ressource est un élément que Puppet sait configurer :

- file (contenu, permissions, propriétaire)
- package (présence ou absence)
- service (activation/désactivation, démarrage/arrêt)
- exec(exécution commande)
- cron, group, host, user etc ...

Exemple :

```
user { 'dave':  
  ensure      => present,  
  uid         => '507',  
  gid         => 'admin',  
  shell       => '/bin/zsh',  
  home        => '/home/dave',  
  managehome => true,  
}
```

Modules

Un module est une collection de manifests, ressources, fichiers, templates, classes et définitions.

Un module contiendra tout ce qui est requis pour configurer une application particulière.

Un module a une structure de répertoire spécifique et un fichier appelé `init.pp`.

Modules

Un module est défini de la façon suivante :

- `/etc/puppet/nom-du-module/manifests/` :
 - Le répertoire manifests contiendra notre fichier `init.pp` et tout autre configuration.
 - Le fichier `init.pp` est le coeur de notre module et chaque module doit en avoir un.
 - Dans le fichier `init.pp` nous allons retrouver des classes qui seront instanciées lors de l'appel d'un agent. Dans ces classes on retrouve les configurations de référence.
- `/etc/puppet/nom-du-module/files/` :
 - Le répertoire files contiendra tous les fichiers que nous souhaitons utiliser comme une partie de notre module, par exemple des fichiers à uploader.
- `/etc/puppet/nom-du-module/templates/` :
 - Le répertoire templates contiendra tous les templates que notre module pourrait utiliser.

Exemple de module

Exmple init.pp (1/2) :

```
#/etc/puppet/modules/sudo/manifests/init.pp
class sudo {
  package { sudo:
    ensure => present,
  }

  if $operatingsystem == "Ubuntu" {
    package { "sudo-ldap":
      ensure => present,
      require => Package["sudo"],
    }
  }
}
```


Exemple de module

Exmple init.pp (2/2) :

```
file { "/etc/sudoers":  
  owner => "root",  
  group => "root",  
  mode  => 0440,  
  source => "puppet://$puppetserver/modules/sudo/etc/su  
  require => Package["sudo"],  
}  
}
```

Exemple de module

Il y a trois ressources dans la classe, deux packages et un fichier ressource.

- Le premier package s'assure que le paquet sudo est bien installé
- Le second package utilise la syntaxe if/else de puppet pour mettre une condition sur l'installation du paquet sudo-ldap.
- La dernière ressource dans la classe de sudo est une ressource fichier, [" / etc / sudoers"], qui gère le fichier /etc /sudoers.

Exemple de module

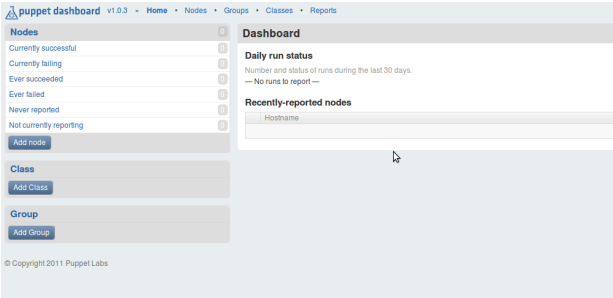
Exemple d'un module ssh avec transfert de fichier texte :

```
root@puppetmaster:/etc/puppet# tree
```

```
.
├── auth.conf
├── fileserver.conf
├── manifests
│   ├── nodes.pp
│   └── site.pp
├── modules
│   ├── ssh
│   │   ├── manifests
│   │   └── init.pp
│   └── test
│       ├── files
│       │   └── test.txt
│       └── manifests
│           └── init.pp
├── puppet.conf
└── templates
```

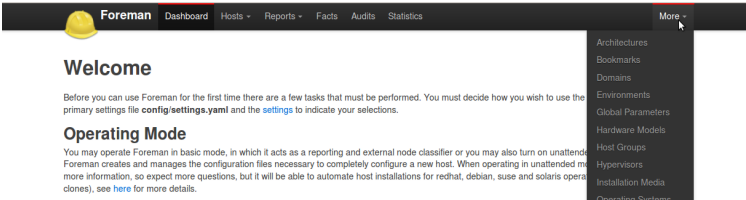
Puppet-Dashboard

Puppet Dashboard est une application avec interface web permettant d'afficher des informations sur le serveur Puppet et son environnement (Master et Agent). Il permet de voir sur des graphes, les données répertoriées d'un serveur Puppet ou de plusieurs (multi-puppet).



The Foreman

Une autre interface web pour Puppet est The Foreman ou Foreman. Elle aussi est réalisée en Ruby. Cette interface permet de montrer l'inventaire du parc informatique basé sur Facter et de donner des informations en temps réel sur le statut des agents basées sur les "Puppet reports".



Puppet Module Tool

Outre les deux interfaces web, « Puppet Forge », propose énormément de modules qui peuvent permettre de faciliter le déploiement rapide d'une infrastructure complexe. Ces modules sont configurés de telle sorte qu'il vous est facile de les modifier.

Les Modules peuvent être télécharger ou installer manuellement, mais il existe un "module manager", Puppet-module tool.

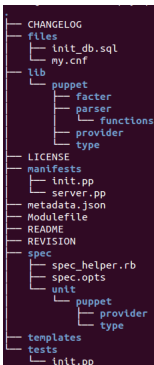
Puppet-module tool permet :

- Créer des squelettes pour vos futurs modules.
- Chercher des modules déjà existants.
- Installer et configurer des modules déjà existants.

Puppet Module Tool

Installation et utilisation :

```
sudo gem install puppet-module  
puppet-module search mysql  
puppet-module install ghoneycutt/mysql
```



```
├── CHANGELOG  
├── files  
│   ├── init_db.sql  
│   └── my.cnf  
├── lib  
│   └── puppet  
│       ├── facter  
│       ├── parser  
│       │   └── functions  
│       ├── provider  
│       └── type  
├── LICENSE  
├── manifests  
│   ├── init.pp  
│   └── server.pp  
├── metadata.json  
├── Modulefile  
├── README  
├── REVISION  
├── spec  
│   ├── spec_helper.rb  
│   ├── spec.opts  
│   └── unit  
│       ├── puppet  
│       │   └── provider  
│       └── type  
├── templates  
├── tests  
└── init.pp
```

Plan

- 1 Puppet Labs
 - Introduction
 - Fonction principale
 - Commencer avec Puppet
- 2 Utilisation
 - Le langage Ruby
 - Manifests
 - Modules
 - Complement à Puppet
- 3 Evaluation et sources
 - Avantages
 - Inconvénients
 - Sources

Avantages

- permet de déployer des applications et de mettre à jour facilement leur configuration.
- s'appuie sur le langage Ruby, et propose donc des fichiers de configuration clairs et lisibles car on y parle objet.
- s'appuie sur des outils génériques et standards.
- sa couche d'abstraction entre l'utilisateur et le logiciel est très agréable.
- permet de gérer l'intégralité d'un réseau.

Inconvénients

Ce que puppet ne fait pas :

- Déploiement automatique d'instances machines
- La supervision distante (type Nagios) mais permet de configurer un Nagios pour les hôtes concernés (typiquement via un module à Puppet)

Introduction à Puppet

Victor Martin,
Sébastien
Michaux et
Riwan Blondé

Puppet Labs

Introduction
Fonction
principale
Commencer avec
Puppet

Utilisation

Le langage
Ruby
Manifests
Modules
Complement à
Puppet

Evaluation et sources

Avantages
Inconvénients

Sources

Sources