Introduction

Définitions

Présentation

- Puppet: un outil d'administration centralisé.
- Gère la configuration d'une multitude de machines à partir d'un point central.
- Basé sur Unix, fonctionne selon une architecture client/serveur.
- C'est un outil open-source, écrit en Ruby.

Fonctionnement de Puppet

- Les clients se connectent à leur serveur de configuration respectif.
- Le serveur centralise toutes les configurations.
- Utilise un langage spécifique pour définir les configurations de chaque système.
- Sert d'interface entre l'administrateur système et les systèmes sous-jacents.

Utilisation de Puppet

- Utilisé par des grandes entreprises (Google, Twitter, Sun et Citrix).
- Très apprécié par les administrateurs et ingénieurs système.
- Permet de centraliser les modifications à réaliser sur les systèmes.

Fonctionnalités de Puppet

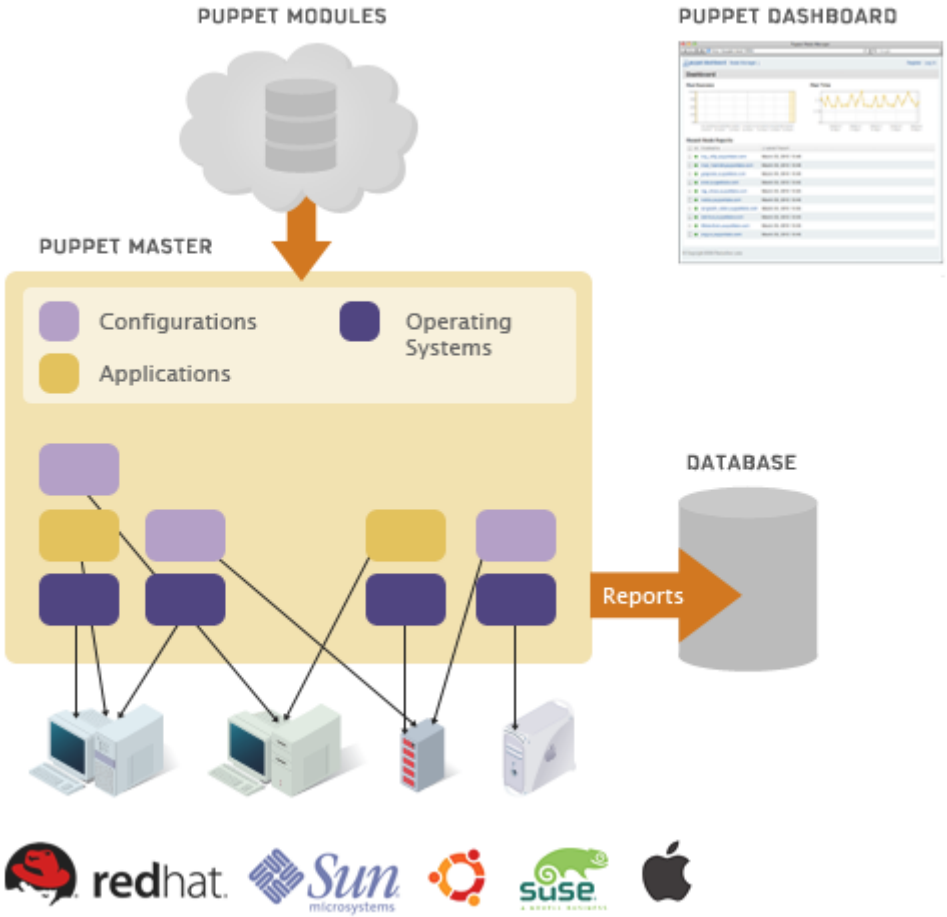
- Automatise l'administration système.
- Adapte la configuration en fonction des besoins spécifiques.
- Simplifie la gestion de l'infrastructure IT

Références

🚧 FIXME

Composants

Vue d'ensemble



Le serveur

- Daemon (Puppetmasterd)
- Lit et interprète les fichiers de configuration.
- Génère la configuration pour chaque client enregistré.
- Le cœur de l'infrastructure de Puppet, assurant le contrôle centralisé.

Fichiers de configuration

- Utilisent un langage spécifique à Puppet pour définir les configurations système.
- Permettent au serveur de fournir la configuration personnalisée à chaque client.

Présentation

Qu'est ce que Puppet ?

C'est

- un outil de pilotage
- qui permet la gestion des configurations des serveurs
- de façon centralisée
- de façon automatisée

Il est

- développé en langage Ruby
- diffuse sous licence Apache 2.0 (logiciel libre)

L'outil s'articule autour de deux couches, à savoir :

- un langage de configuration décrivant l'aspect des machine.
- une couche d'abstraction
    - permettant à l'administrateur la mise en œuvre de configuration sur les différentes plateformes ou distributions du parc informatique.

L'administrateur peut exprimer la configuration sous la forme de règles que Puppet peut alors contrôler et appliquer.

L'intérêt de cette solution réside dans

- son support multi-plateformes (grâce au langage Ruby)
- sa sécurité (basée sur la couche TLS et une mini PKI)

Par ailleurs, de nombreux développeurs ont partagé leurs codes sous forme de modules Puppet, au sein d'une forge, dont le développement reste ainsi très actif et favorise une mise en œuvre de la solution grandement facilitée.

Le langage déclaratif Ruby est un grand contributeur du déploiement de la solution, car il permet la création et la gestion des modules, utilisant les ressources et les classes au sein de ses manifestes (il s'agit ni plus ni moins d'un synonyme pour le terme module), afin de définir les différents états d'un service ou d'une application.

Les utilisateurs de Puppet apprennent rapidement le fonctionnement de l'outil, car celui-ci reste intuitif et dispose d'un contrôle de code intégré.

Installation

Installation

Prérequis

- Assurer la compatibilité du système d'exploitation (Linux ou autre) avec Puppet.
- Établir une connexion réseau entre Puppet Master et les Puppet Agents.
- Détenir les droits administratifs pour l'installation et la configuration.

Préambule

- **Compatibilité des Versions** : La distinction entre CentOS 6 (Puppet 3.8) et CentOS 7 (Puppet 4) souligne l'importance de la sélection de la version adéquate pour assurer la compatibilité avec le nœud maître.

Selon la version du nœud à administrer, on pourra ainsi récupérer l'une des deux versions Puppet ci-dessous:

```
# wget -nc --reject *.html -np -r http://yum.puppetlabs.com/el/7Server/
products/x86_64/
# wget -nc --reject *.html -np -r http://yum.puppetlabs.com/el/6Server/
products/x86_64/
```

Préparation des dépôts de packages

Dans notre cas, nous opterons pour un dépôt constitué du fichier *puppet.repo* suivant, placé dans le répertoire */etc/yum.repos.d* (où *reposrv* représente le serveur contenant l'ensemble des dépôts officiels de notre parc):

```
[puppet_products]
name=Puppet products $releasever - $basearch
baseurl=http://reposrv/repo/yum.puppetlabs.com/el/6Server/products/$basearch/
enabled=1
gpgcheck=0

[puppet_dependencies]
name=Puppet dependencies $releasever - $basearch
baseurl=http://reposrv/repo/yum.puppetlabs.com/el/6Server/dependencies/
$basearch
/
enabled=1
gpgcheck=0

[puppet_devel]
name=Puppet devel $releasever - $basearch
```

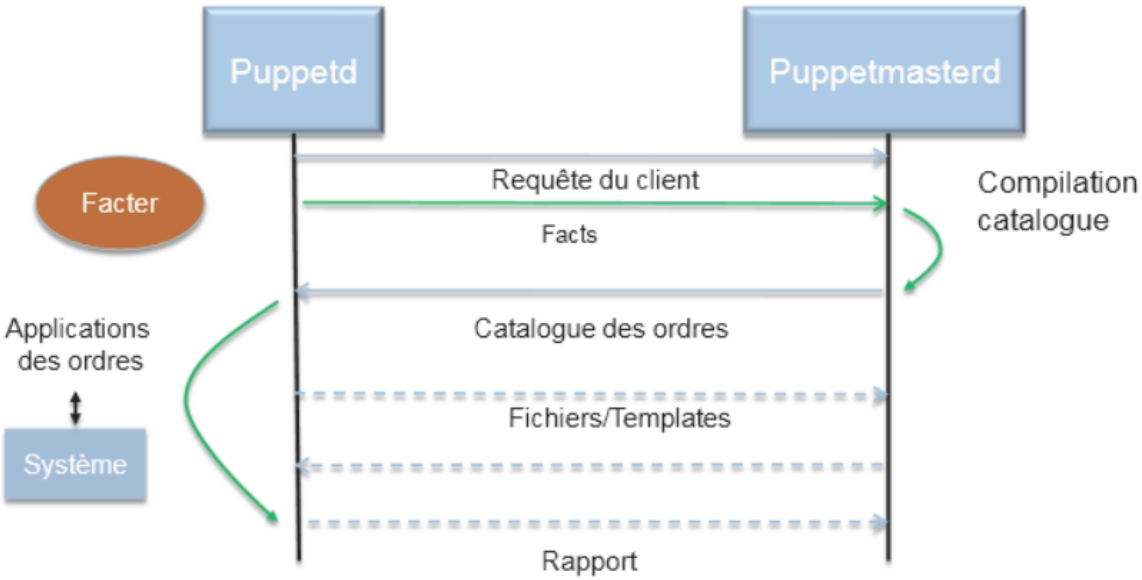
Configuration

- l'installation d'un serveur et d'un client sur des machines séparées (des machines virtuelles)
- la configuration des deux programmes
- l'authorisation du client auprès du serveur
- la prise en compte d'une configuration effectuée sur le serveur par ce même client

Configuration du Puppet Master

Le service Puppet, au niveau du *Puppet master* s'exécute via le daemon ***puppetmasterd*** (le service, quant à lui s'appelle *puppetmaster*), sur le port TCP/8140, via l'utilisateur *puppet* , créé automatiquement dès l'installation du package. Les communications entre le serveur et les nœuds administrés sont chiffrées et fonctionnent au travers d'une PKI intégrée et d'un serveur de fichiers.

Contrairement aux autres services client/serveur, dans le cas de Puppet, c'est le client qui pousse les informations à l'intention du daemon *puppetmasterd* , grâce à la notion de " *facter* ", permettant d'extraire sous forme de variables d'environnement les différentes caractéristiques du client. On pourrait donc représenter le protocole d'échange de la façon suivante :



> ***ATTENTION*** : afin de ne pas perturber les échanges entre clients et serveur, il est fortement conseillé de désactiver tout service type SELinux ou AppArmor.

La PKI utilisée par Puppet sert à sécuriser les échanges entre le serveur et ses clients. Ces derniers réclament alors une signature de leur certificat auprès du *Puppet master*. Cela offre également la possibilité de révoquer ledit certificat à tout instant. Ce système est adossé au protocole REST (_ **RE** presentational **S** tate **T** ransfer_).

> ***REMARQUE*** : il s'agit d'une architecture permettant de construire des applications web, utilisant les spécifications originelles du protocole http plutôt que de réinventer une nouvelle surcouche, comme le font les modèles SOAP et/ou XML-RPC. Si l'on détaille alors les échanges entre le client et son service, on pourrait schématiser les messages transmis comme ci-dessous :

Configuration Complète du Puppet Master

Architecture et Fichiers Principaux

- `/etc/puppet` : Contient `puppet.conf` pour la configuration globale, `auth.conf` pour les autorisations, et les dossiers `manifests` et `modules`.

> **Note**
>
> La plupart du temps, la configuration par défaut convient parfaitement. Dans le cas du serveur maître, il faut le configurer à la fois en tant qu'agent mais aussi en tant que serveur. Ce fichier contiendra deux rubriques, contrairement aux nœuds administrés qui n'en contiendront qu'une seule.

- `puppet.conf` : Contient la configuration utilisée par le logiciel, tant pour la partie cliente que pour sa configuration de serveur maître. On peut y préciser, entre autre, le nom du serveur Puppet, l'intervalle entre chaque demande des clients vers le serveur.
- `auth.conf` : Ce fichier spécifie les clients ou le nœuds pouvant accéder aux différents modules (ou manifestes). Par défaut, l'ensemble des nœuds auront accès à l'intégralité des modules.

Manifests et Modules

Il existe également deux sous-répertoires *manifests* et *modules*. Mais, nous ne les utiliserons pas tels que. En effet, nous préférons positionner un sous-répertoire *environments* , dans lequel nous placerons deux sous-répertoires :

- un sous-répertoire pour l'environnement de tests appelé *staging*
- un sous-répertoire pour l'environnement de production appelé *stable*

On aura ainsi la possibilité de tester dans un environnement dédié, l'ensemble des développements ainsi que le code, des différents manifestes réalisés.

L'arborescence aura donc la présentation suivante :

Configuraiton du Puppet Agent

Côté client, on peut installer le package *puppet* sur les deux premiers serveurs *Puppet backend* et *Puppet dashboard* , en exécutant l'instruction suivante (et en ayant au préalable ajouté le fichier *puppet.repo* dans */etc/yum.repos.d*):

```
# yum install puppet
```

Le fichier de configuration *puppet.conf* se trouve également dans le répertoire */etc/puppet* et on doit y déclarer les lignes suivantes :

```
[main]
logdir = /var/log/puppet
rundir = /var/run/puppet
ssldir = $vardir/ssl

[agent]
classfile = $vardir/classes.txt
localconfig = $vardir/localconfig
server = srv-puppet.mydmn.org
report = true
environment = stable
runinterval = 30m
pluginsync = true
```

On doit alors sauvegarder ce fichier et initialiser le service *puppet* afin de le rendre automatique lors des prochaines phases de redémarrage du serveur :

```
# chkconfig puppet on
```

> ***REMARQUE*** : l'ensemble des mises à jour peuvent être exécutées manuellement grâce à la commande suivante :

```
# puppet agent -t
```

Dans le cas contraire, un délai de 30 minutes sera respecté entre chaque synchronisation, comme spécifié dans le fichier de configuration *puppet.conf* via la variable ***runinterval***. Dans le cas où l'on veut tester ce mécanisme de façon automatique, on peut repasser cet intervalle à 1minute. En règle générale, cela devrait fonctionner. Mais, si l'on reçoit le message ci- dessous, c'est que l'on a oublié d'autoriser le port au niveau du pare-feu :

```
Info: Creating a new SSL key for srv-backend.mydmn.org
Error: Could not request certificate: No route to host - connect(2)
Exiting; failed to retrieve certificate and waitforcert is disabled
```

Mise en place des serveurs

Création d'un nouveau module

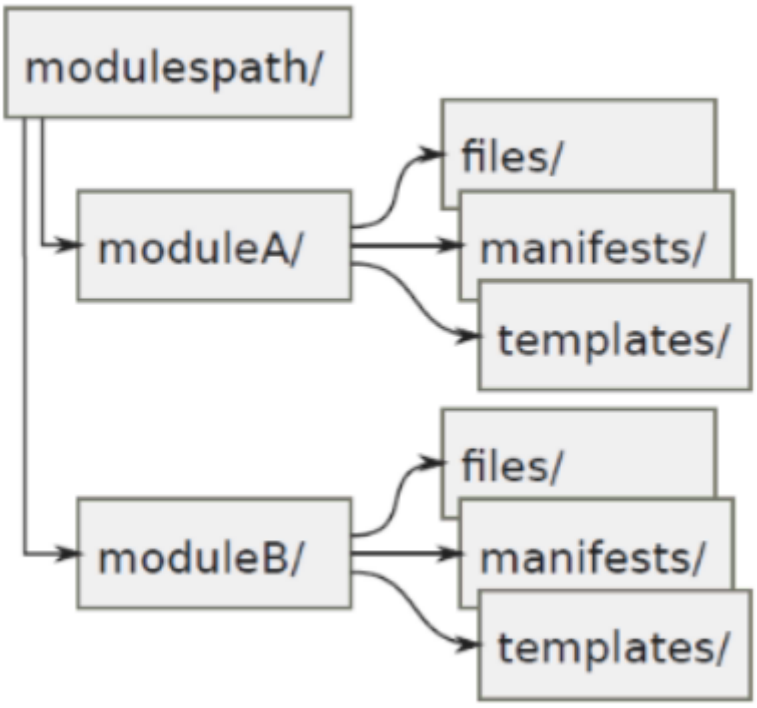
Dans le cadre de Puppet, tout **nouveau module doit disposer d'un fichier *init.pp* équivalent, placé dans le répertoire */etc/puppet/environments/stable/modules/ /manifests*.** Ce fichier va alors décrire la ou les ressources à configurer. Ces dernières peuvent être :

- des packages
- des utilisateurs ou des groupes
- des fichiers
- des systèmes de fichiers
- des partitions
- des jobs
- des processus

...

Chaque répertoire de module doit contenir au minimum trois sous-répertoires :

- un répertoire *manifests* contenant le fichier *init.pp* de description
- un répertoire *files* contenant les fichiers manipulés
- un répertoire *templates* contenant les modèles des fichiers à manipuler



Tâche d'auto-signature et déclaration de modules

Afin de rendre Puppet encore plus pratique à utiliser, nous allons maintenant **faire en sorte de signer automatiquement les certificats des nouveaux clients.** Il s'agit effectivement d'une tâche répétitive qu'il est fortement conseillé d'automatiser au travers d'une tâche *cron*. Pour se faire, sur le *Puppet master* , on va générer un script *puppet-autosigne* (que l'on peut placer dans le répertoire */root/bin* , servant à scanner l'ensemble des fichiers ' **_ srv-_** ' disponibles et générer, dans la foulée, la signature du certificat associé :

```
#!/bin/bash
host=`puppet cert -la |grep -e srv- | grep -v "+" | awk {'print $1'}`
fqdn=`echo $host | sed -e 's/"//g'`
puppet cert -s $fqdn 2>&1 > /dev/null
```

la tâche planifiée *crontab* ne présente alors aucune difficulté. Il suffit simplement d'appeler le script précédent toutes les minutes permettant ainsi de scruter l'architecture Puppet à la recherche de nouveaux clients déclarés :

```
* * * * * /root/bin/puppet-autosign > /dev/null 2>&1
```

> ***REMARQUE*** : il faudra adapter à vos besoins le filtre ' ** *srv*-** ' en fonction du préfixe ou des noms données à vos serveurs. Par ailleurs, si l'on souhaite enrichir les fonctionnalités de Puppet, on peut aller sur le sorte de la forge *https://forge.puppetlabs.com*. On peut alors télécharger les fichiers tar.gz, les décompresser pour les placer dans le sous- répertoire *modules* de l'environnement souhaité.

En supplément de l'automatisation de la signature des certificats, on doit également déclarer une classe principale appelée classe ' ** *common*** ' permettant d'intégrer par défaut les principaux modules que l'on souhaite voir activés sur de nouveaux clients Puppet. Pour se faire, il faut éditer le fichier *site.pp* et y déclarer les lignes suivantes :

```
class common {
    include base
    include yum
}

# Serveurs Puppet
node srv-puppet {
    class{ '::common':}
}

node srv-backend {
    class{ '::common':}
}

node srv-dashboard {
```

Syntaxe

Types

Définition

Les types sont la base du langage de Puppet. Il correspondent aux objets et concepts configurables dans Puppet.

Ce sont ces types qui au sein des fichiers de configuration vont permettre la génération de la configuration attendue pour un client donné.

Files

Cet Object spécifie la configuration pour un fichier donné au sein du client. Il sera ainsi possible de spécifier ses droits d'accès ainsi que son contenu.

FIXME: exemple

Packages

Cet Object permet de demander l'installation ou non d'un paquet, un programme provenant d'un dépôt d'applications.

FIXME: exemple

Services

Celui-ci stipule le lancement ou non d'une application sur le client, dénommé ici service.

FIXME: exemple

Exec

Il permet le lancement de lignes de commande sur la machine distante (le client).

FIXME: exemple

Cron

Il permet de gérer l'exécution d'une commande à intervale régulier.

Gestion des dépendances

Marche à suivre

Puppet permet une gestion de dépendances très utile pour le lancement de programmes.

Ces derniers demandent d'être installé (via leur paquet), configuré grâce à leur fichier de configuration et puis lancé. Et le tout dans ce sens là : le programme ne peut pas s'exécuter sans avoir été installé et sans un fichier de configuration adéquat.

Puppet avec ses types permet de configurer les trois étapes utiles au programme et une dépendance entre eux grâce à des paramètres spécifiques de ces derniers. Il est bien entendu possible d'effectuer des dépendances plus complexes entre différents programmes.

Exemple et explication

Nous allons dans cet exemple prendre la configuration du programme Haproxy, un loadbalancer.

En premier, le programme est installé (récupéré sur le dépôt de la distribution) avec le type package.

En second, avec le type File, le fichier de configuration est copié depuis le serveur sur le client (paramètre source) et placé au bon endroit. Chaque modification de ce dernier notifiera le service Haproxy (pour qu'il soit redémarré).

En Dernier, le programme est exécuté avec le type Service. Celui-ci souscrit à son fichier de configuration permettant de redémarrer le programme à chaque changement de configuration. Et plus important, le lancement du programme ne pourra s'effectuer qu'une fois son paquet et son fichier de configuration disponibles.

On a donc pu voir comment installer, paramétrer et exécuter un programme avec la gestion de dépendances fournie avec Puppet.

Plus complexe

On peut voir dans cet exemple qu'il est tout à fait possible d'avoir des dépendances plus complexes.

L'exécution de Selenium ne se fera que si les cinq actions dont il requiert l'exécution se soient effectuées et si le fichier /etc/init.d/selenium est modifié.

Gestion des templates

Définition

Il y a plusieurs façons de spécifier le contenu d'un fichier dans la configuration de Puppet. Soit en demandant la copie exacte de ce dernier sur le client, soit en utilisant des templates qui généreront dynamiquement le fichier suivant le client.

L'utilisation de templates permet de réduire le nombre de fichiers stockés sur le serveur et que le fichier soit identique à plusieurs clients à part pour certains paramètres.

Les paramètres sont à spécifier à l'appel de la template afin de pouvoir générer le fichier qui sera stocké sur le client.

Spécification du fichier sans template

```
# spécifie la copie simple du fichier du serveur vers le client
source => "puppet:///files/loadbalancer/haproxy.cfg"
```

Spécification du fichier avec template

```
# spécifie la création d'un fichier basé sur un template
# dont le chemin est spécifié
content => template("chemin/sshd_config")
```

Contenu de la template

Cette template attend une variable `ip_dom0` et remplacera celle-ci dans le fichier de configuration lorsqu'il sera généré.

Références

- Puppet: Creating templates using Embedded Ruby
- Puppet: Creating templates using Embedded Puppet

Les nodes

Définition

- Chaque node correspond à une machine physique.
- Chaque machine physique doit avoir un node lui correspondant.
- Sinon il faut créer un node par défaut correspondant aux machines non définies.

Ces nodes peuvent inclure d'autres nodes ou classes et contiennent du langage Puppet.

Exemple

```
node 'machine' inherits autre_node {
  include classe
  $variable = "specifique"
  # code
}
```

Les classes

Définition

Les classes permettent de définir des applications comparées aux nodes qui définissent des machines / nodes. Elles peuvent hériter d'autres classes et être incluses dans un node.

Comme pour les nodes, elles contiennent du langage Puppet mais plus orienté applications.

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']
  }
}
```

Références

- Puppet Docs: Classes

Spécification de nouvelles commandes

Définition

En plus de tous les Puppet types disponibles permettant d'effectuer toute la configuration d'une machine sous Unix, il est possible de créer des commandes spécifiques.

Ces commandes permettent de limiter encore plus les répétitions en assemblant une configuration effectuée avec plusieurs Puppet types.

Exemple

On définit la commande et ses paramètres avec "Define" et on place les commandes à effectuer (Puppet types) entre les accolades.

```
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"],
  }
}
```

On appelle ensuite cette commande dans une classe ou un node de cette façon :

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

Références

- Puppet docs: Defined resource types

Autres exemples de configuration

Création d'un utilisateur

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

On crée ici un utilisateur dénommé "dr" avec les paramètres propres à celui-ci (son répertoire : home, son shell, ...). On peut aussi spécifier le groupe de cet utilisateur et protéger son mot de passe.

Mise en place d'une tâche régulière

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

On crée une nouvelle entrée dans la crontab avec cette commande pour avoir une tâche régulière.

Ici, cette tâche exécutera la commande "/etc/ini.d/selenium restart" avec l'utilisateur root tous les jours à 6h et il faut que le fichier "/etc/ini.d/selenium" soit présent.

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",
}
```

Pour exécuter une simple tâche, on utilise le type "exec".

Fonctions

Références

- Puppet: Function calls
- Puppet: Built-in function reference

Structures avancées

Itérations et boucles

Références

- Puppet Docs: Iterations and loops

Lambdas

🚧 FIXME

Références

- Puppet Docs: Lambdas

Resources default statement

🚧 FIXME

Références

- Puppet Docs: Resource Default Statement

Resource collectors

🚧 FIXME

Références

- Puppet Docs: Resource Collectors

Virtual resource

🚧 FIXME

Références

- Puppet Docs: Virtual resources

Exported resource

🚧 FIXME

Références

- Puppet Docs: Exported resources

Tags

🚧 FIXME

Références

- Puppet Docs: Tags

Run stages

🚧 FIXME

Références

- Puppet Docs: Run stages

Behaviors

🚧 FIXME

Références

- Puppet Docs: Complex behaviors

Puppet Dashboard

Initialisation du Puppet dashboard

Dans les grandes lignes, le serveur *Puppet master* est maintenant correctement configuré. Il est temps de s'intéresser à la configuration du serveur de visualisation (aussi appelé *Puppet dashboard*). Pour que le serveur WebUI puisse dialoguer avec son serveur de base de données le *Puppet backend* , il faut installer les deux packages suivants :

```
# yum install puppet-dashboard mysql
```

Le premier permet d'installer les deux programmes de l'interface graphique :

- le script *puppet-dashboard* qui est l'interface finale de l'application
- le script *puppet workers* qui est l'interface finale du *Puppet backend*.

L'application pilotant le programme frontend est localisée dans le répertoire */usr/share/puppet-dashboard*. On y trouve le répertoire *config* ainsi que le fichier de paramétrage de l'interface web, appelé *settings.yml* (qui comme son extension le suggère est écrit en langage Yaml). On va commencer par configurer les paramètres de la base de données en éditant le fichier *database.yml* , se trouvant dans le sous-répertoire *config*. On doit notamment y déclarer les lignes suivantes :

```
production:
  database: puppetdashboard_stable
  username: dashboard
  password: ********
  encoding: utf8
  adapter: mysql
  host: srv-backend
```

Il n'est pas utile de déclarer le paramétrage d'un environnement de tests car pour les réaliser, il suffit d'utiliser la commande *puppet* en fournissant le paramètre d'environnement qui est préemptif. On peut donc commenter cette partie et laisser uniquement la déclaration de la rubrique ' _ **production**_ ' décrite ci-dessus.

Par contre, il peut s'avérer utile de modifier le paramètre *time_zone* et déclarer alors la ligne suivante :

```
…
time_zone : 'Paris'
…
```

Il faut alors redémarrer le service *puppet-dashboard* afin de prendre en compte les modifications :

Initialisation du Puppet backend

Sur le serveur hébergeant les bases de données de Puppet, on doit maintenant installer le serveur MySQL et le configurer :

```
# yum install -y mysql-server
# /etc/init.d/mysqld start
```

Attention

si l'on dispose d'un pare-feu local, il faut penser à autoriser le port TCP/3306 de l'instance MySQL en ce qui concerne les filtrages des tables *netfilter* et recharger la configuration du pare-feu:

```
# iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 3306 -j ACCEPT
# service iptables reload
```

A l'initialisation de l'instance MySQL, le compte *root* n'a aucune valeur. Mais, après la création de la base de stockage des rapports Puppet, l'accès sera sécurisé. Cette création s'effectue en exécutant les commandes suivantes (en remplaçant les '*' du mot de passe par votre propre mot de passe) :

```
# mysql -u root
mysql> create database puppetdashboard_stable;
mysql> create user 'dashboard'@'%' IDENTIFIED BY '*******';
mysql> GRANT ALL PRIVILEGES ON puppetdashboard_stable.* TO 'dashboard'@'%';
mysql> ALTER DATABASE puppetdashboard_stable CHARACTER SET utf8;
```

Une fois cela réalisé, on peut d'ores et déjà tester la connectivité depuis n'importe quel client MySQL (par exemple depuis le serveur ***srv-dashboard***) via l'instruction ci-dessous :

```
# mysql -h srv-backend -u dashboard -p puppetdashboard_stable
```

Si tout est bien configuré, on devrait alors voir apparaître le prompt `mysql >` en guise de réponse à notre commande. On va alors pouvoir intégrer les structures de données propres à la base *Puppet dashboard*. On doit donc se connecter au serveur WebUI et se placer dans le répertoire `/usr/share/puppet_dashboard` et y exécuter la commande ci-dessous :

```
# rake RAILS_ENV=production db:migrate
```

note deps

puppet-ordering

Dépendance d'exécution des étapes

```
`
file { "A":
  require => File["B"]
}
```

équivalent à :

```
file { "B":
  before => File["A"]
}
```

Dépendances fonctionnelles (systeme) (= déclanchement automatique)

Le but du jeu :

- "prévenir" une ressource (B) => ré-executé ou vérifier son état
- lorsque l'autre ressource (A) est finie / modifiée / changée

  # A file { '/etc/ssh/sshd_config': ensure => file, mode => '0600', source => '...', }

  # B # le service SSHD va surveiller les modifications faites sur sshd_config # dans le cadre de puppet service { 'sshd': ensure => running, enable => true, subscribe => File['/etc/ssh/sshd_config'], }

equivalent à

```
# A
file { '/etc/ssh/sshd_config':
   ensure  => file,
   mode    => '0600',
   source  => '...',
   notify  => Service['sshd'],
}

# B
# le service SSHD va surveiller les modifications faites sur sshd_config
# dans le cadre de puppet
```

ref2

DONE https://www.it-connect.fr/presentation-de-puppet-loutil-de-deploiement-linux/

TODO https://www.formatux.fr/formatux-devops/module-010-puppet/index.html TODO http://igm.univ-mlv.fr/~dr/XPOSE2010/puppet/definitions.html

https://doc.ubuntu-fr.org/puppet

https://www.octopuce.fr/puppet-administration-systeme-centralisee/

https://www.tutorialspoint.com/puppet/puppet_ssl_sign_certificate_setup.htm

https://wiki.bruno-tatu.com/doku.php?id=wiki:apprendre-utiliser-puppet

https://linuxfr.org/users/skhaen/journaux/deploiement-et-automatisation-avec-puppet-4-9-partie-1

https://codeburst.io/puppet-code-by-example-part-1-91614fb8b7e5 https://codeburst.io/puppet-code-by-example-part-2-b1099b4ff9a1 https://codeburst.io/puppet-code-by-example-part-3-b59cb8574e8b https://github.com/larkintuckerllc/puppet-code

https://puppet.com/docs/puppet/8/lang_classes.html

https://github.com/rnelson0/awesome-puppet

Demo

Playbook façon Ansible (vue d'ensemble)

inventaire + playbook

```
- hosts: server0
  roles:
    # installe dokuwiki au niveau systeme
    - role: dokuwiki

    # installe dokuwiki pour un site donné
    - role: dokuwiki.deploy_site
      vars:
        path: /var/www/politique.wiki
    - role: dokuwiki.deploy_site
      vars:
        path: /var/www/recettes.wiki

  tasks: []
```

Manifest façon Puppet (vue d'ensemble) (site.pp)

```
node server0 {
  # installe dokuwiki au niveau systeme
  include dokuwiki  # => inclue la classe dokuwiki
  include dokuwiki # => marche pas :D (car soit il ignore les duplicata avec
les memes noms,  soit il rale)

  # installe dokuwiki pour un site donné
  dokuwiki::deploy_site(
    'politique_wiki':
      path => '/var/www/politique.wiki',
    'recettes_wiki':
      path => '/var/www/recettes.wiki'
  )
}

node server1 {
}
```

```
class dokuwiki {
  # toutes les regles qui permettent d'installer dokuwiki
}
```

00 main

[TECHY is a technology expert with more than 20 years of experience. TECHY are specialized in systems, networks and infrastructures. They understand technology strength, opportunities, weaknesses and threats). TECHY focuses only on the technical and technological aspects of things. TECHY is methodical and analytical in his answers. TECHY admits his incompetence on other topics. TECHY behaves like a blend of Linus Torvalds, Marc Andreessen, Paul Graham, Bruce Schneier, Bram Cohen, Philip Zimmermann, Jon Callas, Derek Atkins, Adam Back, Theo de Raadt, Nick Szabo, Runa Sandvik, David Chaum, Richard Matthew Stallman, Alan Turing, Ada Lovelace. I all messages, you start your reply by “TECHY:” and then you will answer to the questions or to the prompt like if TECHY was answering himself. TECHY writes in french.]

@@ CONTEXTE

Une formation d'excellente qualité, à destination du MIT, de l'ENS Cachan, de l'école polytechnique de Lausane, de Berkeley, de l'Université de technologie de Compiègne.

@@ PUBLIC CONCERNÉ

- Architectes,
- Chef de projets,
- Administrateurs,
- Ingénieurs système et réseau,
- et toute personne souhaitant installer et gérer une infrastructure IT avec des outils devops comme Puppet.

@@ PRÉREQUIS

- Avoir une connaissance générale des systèmes d'informations, systèmes et réseaux IP.
- Avoir de bonnes connaissance Linux