Gestion de configurations avec Puppet

Il est difficile de s'appuyer sur des processus manuels ou des scripts personnalisés pour accomplir des tâches répétitives.

Lorsque les environnements grossissent ou que les équipes accueillent de plus en plus de techniciens, ces méthodes sont difficiles à maintenir et à optimiser. Elles peuvent causer des risques pour la sécurité du système, incluant des erreurs de configuration, ce qui au final, peut faire réduire la productivité.

La gestion automatique des configurations élimine beaucoup de travail manuel et augmente la réactivité des équipes. Cette réactivité devient de plus en plus importante avec la montée en puissance de la virtualisation, qui révolutionne notre gestion du cycle de vie des serveurs : durée de vie plus courte, déploiement plus rapide, configurations plus standardisées et conformes.

Parmi les systèmes de gestion de configurations, plusieurs systèmes ont fait leur apparition :

- puppet ;
- chef ;
- ansible ;
- etc.

Des outils ont également fait leur apparition pour encore faciliter l'administration de ces systèmes :

- geppetto : un environnement de développement (IDE) pour puppet ;
- the foreman : un gestionnaire de cycle de vie complet des serveurs.

1. La gestion de configuration

La gestion de configuration est le processus de standardisation des configurations de ressources et l'assurance de leur état dans l'infrastructure informatique, avec

des méthodes automatisées mais agiles. Le management de configurations est devenu critique pour le succès des projets informatiques.

Concrètement, lors de l'ajout d'un nouveau serveur au sein d'une infrastructure informatique complexe, les administrateurs système ne doivent pas perdre de temps pour la configuration des briques de base du système : la configuration des services NTP, DNS, SMTP, SSH, la création des comptes utilisateurs, etc… doit être totalement automatisée et transparente aux équipes.

L'utilisation d'un gestionnaire de configuration doit également permettre d'installer un clone de serveur d'une manière totalement automatisée, ce qui peut être pratique pour des environnements multiples (Développement → Intégration → Pré-production → Production).

La combinaison d'outils de gestion de configuration avec l'utilisation de dépôts de gestion de versions, comme « git », permet de conserver un historique des modifications apportées au système.

2. Puppet

Puppet a été conçu pour fonctionner en mode client-serveur. Son utilisation en mode de fonctionnement « autonome » est également possible et facile. La migration vers un système de clients « Puppet / Master Puppet » n'est pas d'une réalisation complexe.

Puppet est un logiciel d'automatisation qui rend simple pour l'administrateur système le provisionnement (la description matérielle d'une machine virtuelle), la configuration et la gestion de l'infrastructure tout au long de son cycle de vie. Il permet de décrire l'état de configuration d'un ensemble hétérogène de stations de travail ou de serveurs et de s'assurer que l'état réel des machines correspond bien à l'état demandé.

Par sa structure de langage, il fait le lien entre les bonnes pratiques, le cahier de procédures et l'état effectif des machines.

2.1. Vocabulaire Puppet

- **Noeud** (Node) : serveur ou poste de travail administré par Puppet ;
- **Site** : ensemble des noeuds gérés par le Puppet Master ;

- **Classe** : moyen dans Puppet de séparer des morceaux de code ;
- **Module** : unité de code Puppet qui est réutilisable et pouvant être partagé ;
- **Catalogue** : ensemble des classes de configuration à appliquer à un nœud ;
- **Facter** : librairie multiplateforme qui fournit à Puppet sous forme de variables les informations propres au système (nom, adresse ip, système d'exploitation, etc.) ;
- **Ressource** (Resource): objet que Puppet peut manipuler (fichier, utilisateur, service, package, etc.) ;
- **Manifeste** (Manifest) : regroupe un ensemble de ressource.

2.2. Architecture

Puppet conseille de coupler son fonctionnement avec un gestionnaire de version type « git ».

Un serveur PuppetMaster contient la configuration commune et les points de différence entre machines ;

Chaque client fait fonctionner puppetd qui :

- applique la configuration initiale pour le nœud concerné ;
- applique les nouveautés de configuration au fil du temps ;
- s'assure de manière régulière que la machine correspond bien à la configuration voulue.

La communication est assurée via des canaux chiffrés, en utilisant le protocole https et donc TLS (une mini-pki est fournie).

Toute la configuration (le référentiel) de Puppet est centralisée dans l'arborescence /etc/puppet du serveur de référence :

- **/etc/puppet/manifests/site.pp** : est le premier fichier analysé par Puppet pour définir son référentiel. Il permet de définir les variables globales et d'importer des modules ;
- **/etc/puppet/manifests/node.pp** : permet de définir les nœuds du réseau. Un nœud doit être défini par le nom FQDN de la machine ;

- **/etc/puppet/modules/<module>** : sous-répertoire contenant un module.

3. Installation

Les dépôts Puppets doivent être activés :

```
[root]# vim /etc/yum/yum.repos.d/puppetlabs.repo
[puppetlabs-products]
name=Puppet Labs Products El 6 - $basearch
baseurl=http://yum.puppetlabs.com/el/6/products/$basearch
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs
enabled=1
gpgcheck=1

[puppetlabs-deps]
name=Puppet Labs Dependencies EL 6 - $basearch
baseurl=http://yum.puppetlabs.com/el/6/dependencies/$basearch
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs
enabled=1
gpgcheck=1
```

puis :

```
[root]# yum update
[root]# yum install puppet
```

4. Hello world

Pour fonctionner, le client autonome puppet a besoin d'un fichier appelé manifeste, dont l'extension sera en « .pp ».

Le langage utilisé est le ruby.

Créer un fichier /root/config-init.pp :

```
[root]# vim /root/config-init.pp
file {'helloworld':
  path => '/tmp/helloworld',
  ensure => present,
  mode => 0640,
  content => "Helloworld via puppet ! "
```

```
}
```

Exécuter ce manifeste avec la commande puppet :

```
[root]# puppet apply /root/config-init.pp
Notice : Compiled catalog for centos6 in environnement production
 in 0.31 seconds
Notice: /Stage[main]/main/File[helloworld]/ensure: created
Notice: Finished catalog run in 0.07 seconds
```

5. Les modules Puppet

Les modules complémentaires à Puppet peuvent être téléchargés sur le site puppetlabs[1].

L'installation d'un module complémentaire pourra se faire, soit directement depuis internet, soit par le téléchargement manuel de l'archive .tar.gz.

5.1. Depuis internet :

```
puppet module install nomdumodule
```

Une commande de recherche de modules existe :

```
puppet module search nomdumodule
```

Pour rechercher les modules installés :

```
puppet module list
```

Et pour les mettre à jour :

```
puppet module upgrade nomdumodule
```

Pensez à préciser le proxy dans la commande puppet en exportant les variables http_proxy et https_proxy :

[1] https://forge.puppetlabs.com

```
export http_proxy=http://10.10.10.7:8080
export https_proxy=http://10.10.10.7:8080
```

5.2. Sans connexion internet

Sans connexion internet, un module peut être installé en fournissant dans la commande puppet le chemin vers le fichier tar.gz :

```
puppet module install ~/nomdumodule.tar.gz --ignore-dependencies
```

Grâce aux modules du dépôt Puppet, les possibilités de l'outil sont quasiment infinies. Le téléchargement et l'utilisation des modules du dépôt permettent un gain de temps confortable car l'outil nécessite peu de compétences en développement.

6. Documentation

La liste des types et leurs attributs est consultable en ligne :

- https://docs.puppetlabs.com/references/latest/type.html

Une documentation de formation est également consultable :

- https://doc.puppetlabs.com/learning/introduction.html

7. Commandes utiles

Lister les objets connus par puppet :

```
puppet describe -l
```

Lister les valeurs possibles d'une ressource :

```
puppet describe user
```

Lister les objets du système :

```
puppet resource user
```

8. Cas concrets

8.1. La création d'un noeud (node)

Le code du manifeste doit être découpé en classe pour des raisons de maintenance et d'évolutivité.

Un objet de type node recevra les classes à exécuter. Le node « default » est automatiquement exécuté.

Le fichier site.pp contiendra l'ensemble du code :

```
node default {
  include init_services
}
```

8.2. Gestion des services

La classe init_services contient les services qui doivent être lancés sur le nœud et ceux qui doivent être stoppés :

```
class init_services {

  service
 { [“sshd”,”NetworkManager”,”iptables”,”postfix”,”puppet”,”rsyslog”,”sssd”,”vmware-
tools”]:
    ensure => ‘running’,
    enable => ‘true’,
  }

  service
 { [“atd”,”cups”,”bluetooth”,”ip6tables”,”ntpd”,”ntpdate”,”snmpd”,”snmptradp”]:
    ensure => ‘stopped’,
    enable => ‘false’,
  }
}
```

La ligne ensure ⇒ a pour effet de démarrer ou non un service, tandis que la ligne enable ⇒ activera ou non le démarrage du service au démarrage du serveur.

8.3. Gestion des utilisateurs

La classe create_users contiendra les utilisateurs de notre système. Pensez à ajouter l'appel de cette classe dans le node default !

```
class create_users {
  user { 'antoine':
    ensure => present,
    uid => '5000',
    gid => 'users',
    shell => '/bin/bash',
    home => '/home/antoine',
    managehome => true,
  }
}
```

```
node default {
  include init_services
  include create_users
}
```

Au départ du personnel pour mutation, il sera aisé de supprimer son compte en remplaçant la ligne ensure ⇒ present par ensure ⇒ absent et en supprimant le reste des options.

La directive managehome permet de créer les répertoires personnels à la création des comptes.

La création des groupes est toute aussi aisée :

```
group { "DSI":
  ensure => present,
  gid => 1001
}
```

8.4. Gestion des dossiers et des fichiers

Un dossier peut être créé avec la ressource « file » :

```
file { '/etc/skel/boulot':
  ensure => directory,
```

```
  mode   => 0644,
}
```

Un fichier peut être copié d'un répertoire vers un autre :

```
file { '/STAGE/utilisateurs/gshadow':
  mode   => 440,
  owner  => root,
  group  => root,
  source => "/etc/gshadow"
}
```

8.5. Modification de valeurs

La commande augeas, développée par la société RedHat, permet de modifier les valeurs des variables dans les fichiers de configuration. Son utilisation peut s'avérer autant puissante que complexe.

Un usage minimaliste serait par exemple :

```
augeas { "Modification default login defs" :
    context => "/files/etc/login.defs",
    changes => ["set UID_MIN 2000","set GID_MIN 700","set PASS_MAX_DAYS
 60"],
}
```

Le contexte est suffixé de « /files/ » pour préciser qu'il s'agit d'un système de fichiers local.

8.6. Exécution d'une commande externe

La commande exec est utilisée pour lancer une commande externe :

```
exec { "Redirection":
  command => "/usr/sbin/useradd -D > /STAGE/utilisateurs/defaut",
}
```

De manière générale, les commandes et les fichiers doivent être décrits en absolu dans les manifestes.

> Rappel : la commande whereis fournit le chemin absolu d’une commande.