

TP Puppet

0. Préparation de l'environnement

Dans cette introduction, nous allons démarrer 2 machines virtuelles. Utilisez :

- **vm0** comme SERVEUR Puppet
- **vm1** comme AGENT Puppet (aussi appelé CLIENT).

1. Démarrer **2** machines virtuelles avec PCOCC avec le *template* du TP Puppet

```
pcocc alloc -c4 mycentos74-tp-puppet:2
```

2. Se connecter à la première machine virtuelle **vm0** et ouvrir un shell *root*

```
pcocc ssh vm0  
sudo -s
```

3. En étant *root*, installer et démarrer *puppetserver* sur **vm0**

```
yum -y install puppetserver  
source /etc/profile.d/puppet-agent.sh  
systemctl start puppetserver
```

4. Connectez-vous sur **vm1** via ssh et installer l'agent

```
yum -y install puppet-agent  
source /etc/profile.d/puppet-agent.sh
```

1. Enregistrer l'agent

Chaque agent puppet doit enregistrer sa clé auprès du *puppetserver* afin de crypter ses communications :

1. Se connecter sur un client puppet (**vm1**), et exécuter l'agent pour créer une requête de certificats

```
puppet agent -t
```

2. Se connecter sur le serveur puppet (**vm0**) et accepter la demande de certificat

```
puppet cert list
```

```
puppet cert sign <NOMDUCLIENTPUPPET>
```

3. Sur l'agent, réexécuter l'agent Puppet afin de vérifier que le certificat est signé et que l'agent applique un catalogue, vide pour l'instant

```
puppet agent -t
```

2. Configuration locale

- Sur **vm0**, si vous utilisez **Vim** vous pouvez installer la coloration syntaxique :

```
cp -r /opt/puppetlabs/puppet/share/vim/puppet-vimfiles ~/.vim
```

1. Créer un simple fichier `site.pp` dans les **manifests** de l'environnement **production** qui installe la *dernière version* d'un package de votre choix.
2. Vérifier sa syntaxe avec l'aide de Puppet
3. Appliquer cette configuration localement, sans avoir recours au mode client-serveur de Puppet

3. Premier module Puppet

En se souvenant que l'environnement par défaut se nomme **production**,

1. Créer un module contenant une classe installant le package **rsyslog**
2. Inclure cette classe dans un *manifest* de façon à ce que tous les nœuds récupèrent cette configuration.
3. Appliquer cette configuration avec puppet, d'abord en mode vérification, puis de façon effective.
4. Enrichir la classe afin qu'elle configure le service **rsyslog** afin qu'il soit démarré, et cela, à chaque redémarrage de la machine.

4. Paramétrage

Un serveur *rsyslog* enverra ses messages vers un autre serveur *rsyslog* en écoute.

Nous allons enrichir la classe afin qu'elle puisse configurer 2 fichiers optionnels. Ils seront installés, ou non, en fonction des paramètres de cette classe.

1. Modifier le manifest qui inclut la classe créée dans la partie 3 afin d'avoir 2 blocs de configuration distincts, un pour **vm0** et un pour **vm1**, qui inclut chacun la classe pour **rsyslog**.
2. Enrichir la classe afin de créer un fichier de configuration `/etc/rsyslog.d/forward.conf` dont le propriétaire et le groupe sont *root* et le contenu `"*. * @toto:514"`
3. Rajouter un paramètre optionnel à la classe afin que le nom du serveur que l'on

souhaite contacter soit configurable.

4. A l'aide de l'instruction **if**, tester la valeur de ce paramètre afin que le fichier ne soit installé que si ce paramètre est défini.
5. Modifier le bloc de la machine virtuelle *cliente* afin qu'elle utilise ce paramètre pour installer ce fichier.
6. De la même façon, positionner un fichier `/etc/rsyslog.d/tcp.conf` avec le contenu suivant "**\\$ModLoad imtcp\n\n\$InputTCPServerRun PORT\n**" et utiliser une variable pour rendre ce port paramétrable. Le fichier ne sera mis en place que si le numéro de port est défini.
7. Utiliser ce paramètre pour que rsyslog, sur le nœud serveur, écoute sur le port utilisé par le client pour envoyer ses messages syslog.
8. Appliquer ces configurations sur chacune des machines virtuelles

5. Module ntp

1. En s'inspirant du module précédent et de vos connaissances du service NTP, créer un module qui démarre un service NTP automatiquement avec un fichier de configuration, utilisant un template puppet, qui déclare un unique serveur NTP source dont l'adresse IP est un paramètre de la classe.
2. Utiliser ce module afin que le serveur Puppet utilise son horloge locale (IP : 127.127.1.0) et que la VM du client Puppet se synchronise sur la VM serveur Puppet.

6. Rôles

Un *external node classifier*, simpliste, est installé dans la VM, c'est le script `/usr/sbin/enc.sh`. Lorsqu'il est appelé avec **vm0** en paramètre, il indique de charger la classe *server* et *client* pour **vm1**.

1. Modifier les manifests afin de créer 2 profils, contenant chacun une classe, l'un pour le rôle client, l'autre pour le rôle server. Vérifier que la configuration s'applique toujours correctement
2. Configurer puppetserver afin qu'il utilise le script `/usr/sbin/enc.sh` comme *external node classifier*. C'est ce script qui définit quel profil charger. Simplifier les manifests en conséquence, en supprimant ce qui est maintenant inutile.
3. En reprenant la hiérarchie par défaut de Hiera, créer des fichiers YAML afin que les paramètres des classes créées précédemment soient récupérés grâce à Hiera plutôt que dans les fichiers *manifests*.
4. Vérifiez que la configuration s'applique toujours correctement.