Lab 5 - les fichiers de configuration, le code et la CLI de Puppet

Aperçu

Dans ce lab, nous nous familiariserons avec:

- Fichier de configuration de Puppet: **puppet.conf**
- Puppet depuis la ligne de commande
- Principes de base du code de Puppet
- Cibler le Code à un nœud (**Node Classification**)

Où est-il installé Puppet?

Commençons par regarder le **Puppet Master** (puppet.example.com)

Le fichier de configuration de Puppet, ainsi que les différents autres composants fournis sont stockés sous **/etc/puppetlabs**. Regardons Puppet, sa configuration et son code sous: **/etc/puppetlabs**

Voici les fichiers et répertoires que nous allons commencer à examiner.

```
[root@puppet ~]# tree /etc/puppetlabs/code
/etc/puppetlabs/code
├── environments
│   └── production
│       ├── environment.conf
│       ├── hieradata
│       ├── manifests
│       │   └── site.pp
│       └── modules
└── modules
```

site.pp est le **main manifest** (également appelé **site manifest**) que Puppet lit en premier. C'est le premier morceau de code analysé par Puppet, ou le point d'entrée dans notre base de code Puppet.

Notez qu'il existe un répertoire **production**, qui correspond à l'**environnement de production** Puppet. Il y a un seul environnement appelé **production** bien que d'autres puissent être créés et utilisés.

Dans le répertoire `production/`, nous avons `manifest/`, `modules/` et `hieradata/`. Il peut également y avoir des répertoires pour `files/` et `templates/` mais ils n'ont pas été créé.

Certains de ces répertoires sont également utilisés dans chaque module Puppet. (Un module est un bout de code Puppet empaqueté d'une manière bien définie, de sorte que il peut être distribué et utilisé par d'autres personnes facilement.) Nous en apprendrons plus à propos des modules plus tard, mais pour l'instant, rappelez-vous que l'utilisation de chaque répertoire est bien défini, et vous ne devez pas les utiliser pour ce que vous voulez:

- **files** - utilisé pour les fichiers statiques auxquels votre code Puppet peut faire référence
- **manifests** - où réside votre `site.pp`, et potentiellement d'autres codes développés pour le site
- **modules** - tous les modules de Puppet que vous utilisez sont ici, y compris les modules développés pour le site
- **templates** - similaire au répertoire de fichiers, mais contient des fichiers balisés au format ERB

Le répertoire des environnements

Puppet est livré avec une configuration d'environnement unique, appelée **production**.

Les environnements sont utiles pour contenir différents ensembles de modules, code et données du site. Il est possible, par exemple, de tester une version plus récente d'un module, ou un code Puppet que vous développez activement, dans un environnement différent, totalement séparé de l'environnement **production**.

De cette façon, vous pouvez apporter des modifications à votre code, le tester sur un système de test, sans avoir jamais à se soucier d'affecter les systèmes de production. Nous reviendrons sur ce sujet plus en profondeur dans un lab ultérieur ...

Pour l'instant, sachez ceci: l'environnement obtient son propre répertoire dans le répertoire **environments**, et chaque environnement contient son propre ensemble de manifests, modules, files, templates et données Hiera.

Le répertoire des modules

Le répertoire des modules peut contenir du code Puppet fournis de PuppetLabs ou d'autres tiers, voire des modules développés en interne. Un module commun qui est utilisé par de nombreux autres modules Puppet est **stdlib**. C'est une sorte de "module utilitaire" qui ajoute des lames supplémentaires à votre couteau suisse sous la forme des types de ressources et des fonctions. Il s'agit également d'un module supporté par PuppetLabs .

Commençons par examiner certaines des choses que nous pourrions faire à partir de la ligne de commande par rapport aux modules ...

Liste des modules installés

Dans certains cas, Puppet est livré avec certains modules préinstallés. Pour voir ce qui est installé, exécutez la commande `puppet module list` comme suit sur votre nœud Puppet (le "Puppet Master"):

```
puppet module list
```

Installer un module Puppet

Que faire si vous souhaitez installer un module Puppet à partir de *Puppet Forge*? (Suivez, continuez et exécutez chaque commande pendant que nous en parlons ...)

```
[root@puppet ~]# puppet module install puppetlabs/stdlib
Notice: Preparing to install into
/etc/puppetlabs/code/environments/production/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppetlabs/code/environments/production/modules
└── puppetlabs-stdlib (v4.13.1)
```

Maintenant, regardez à nouveau vos modules installés et vous devriez le voir dans la liste:

```
[root@puppet ~]# puppet module list
/etc/puppet/modules
└── puppetlabs-stdlib (v4.13.1)
 [snip]
```

Notez que lorsque vous avez installé le module Puppet, il était automatiquement installé dans l'environnement **production**. Par défaut,les modules sont installés dans le **premier élément du modulepath**:

```
 [root@puppet ~]#  puppet config print modulepath
/etc/puppetlabs/code/environments/production/modules:/etc/puppetlabs/code/modules:/
opt/puppetlabs/puppet/modules
```

Le modulepath contient des chemins absolus séparés par deux-points vers les emplacements où puppet doit rechercher des modules Puppet. Lors de l'utilisation d'un module, le Puppet Master cherchera dans chaque répertoire de gauche à droite jusqu'à ce qu'il trouve le module. Il utilisera le premier module trouvé si vous avez le même module installé à plusieurs endroits.

Que faire si vous souhaitez utiliser ce même module dans un environnement différent? Et une version différente?

```
[root@puppet ~]# cd /etc/puppetlabs/code/

[root@puppet code]# mkdir -p environments/development/modules

[root@puppet code]# puppet module install --environment development
puppetlabs/stdlib --version 6.6.0

Notice: Preparing to install into
/etc/puppetlabs/code/environments/development/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppetlabs/code/environments/development/modules
└── puppetlabs-stdlib (v6.6.0)
```

Remarquez qu'il existe maintenant deux répertoires d'environnement (**development** et **production**), et nous avons installé deux versions différentes du module **stdlib**.

```
[root@puppet code]# tree -L 3 environments
environments
├── development
│   └── modules
│       └── stdlib
└── production
    ├── environment.conf
    ├── hieradata
    ├── manifests
    │   └── site.pp
    └── modules
        └── stdlib
```

```
[root@puppet code]# grep '"version":' environments/*/modules/stdlib/metadata.json

environments/development/modules/stdlib/metadata.json:  "version": "6.6.0",
environments/production/modules/stdlib/metadata.json:  "version": "4.13.1",
```

- L'environnement de développement a la v6.6.0
- L'environnement de production a la v4.13.0

Que faire si nous voulons installer un module dans le répertoire "site modules" dans /etc/puppetlabs/code/modules? Les modules installés ici seraient/pourraient être utilisés par n'importe quel/chaque agent, quel que soit l'environnement. C'est comme un référentiel **commun** ou **global** de modules. Rappelez-vous simplement que parce que Puppet

cherche dans modulepath, si le répertoire des modules d'un environnement contient le même module, il sera utilisé à la place de ceux trouvés plus bas dans le chemin de recherche du module, comme dans ce cas.

Essayons d'installer une version différente de stdlib ici ...

```
 [root@puppet code]# puppet module install --target-dir
/etc/puppetlabs/code/modules puppetlabs/stdlib --version 4.12.0

Notice: Preparing to install into /etc/puppetlabs/code/modules ...
Error: Could not install module 'puppetlabs-stdlib' (v4.12.0)
Module 'puppetlabs-stdlib' (v4.13.1) is already installed
Use `puppet module upgrade` to install a different version
Use `puppet module install --force` to re-install only this module
```

Oops. Puppet vous avertit que vous essayez d'installer un module déjà installé et plus récent. Forçons l'installation ...

```
[root@puppet code]# puppet module install --target-dir
/etc/puppetlabs/code/modules puppetlabs/stdlib --version 4.12.0 --force

Notice: Preparing to install into /etc/puppetlabs/code/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppetlabs/code/modules
└── puppetlabs-stdlib (v4.12.0)
```

Maintenant, nous allons greper récursivement à travers les fichiers en commençant par notre répertoire de travail actuel et rechercher la chaîne **"version"**. C'est un moyen rapide et facile de voir les versions de tous les modules installés (chaque module doit avoir un fichier metadata.json contenant cette chaîne.)

```
[root@puppet code]# grep -r '"version":' .

environments/production/modules/stdlib/metadata.json:  "version": "4.13.1",
./environments/development/modules/stdlib/metadata.json:  "version": "6.6.0",
./modules/stdlib/metadata.json:  "version": "4.12.0",
```

Vous pouvez également utiliser la commande **puppet module list** pour voir quels modules sont installés pour un environnement particulier et leurs versions:

```
[root@puppet code]# puppet module list --environment=production
[root@puppet code]# puppet module list --environment=development
[root@puppet code]# puppet help module
```

avant de passer à la section suivante, faisons à nouveau une vérification manuelle du Puppet Master et du nœud agent.

```
root@puppet code]# puppet agent -t
```

Notez qu'après avoir installé un module, et au cours de l'exécution de Puppet, tout un tas des fichiers Ruby sont téléchargés et mis en cache sur l'hôte. Certains d'entre eux sont les code ruby pour des facts personnalisés, et d'autres sont du code ruby pour des types et des providers personnalisés qui ont été implémentés dans le module. Avant même d'utiliser le module, ces codes sont téléchargés et mis en cache côté agent.

Sachez également que si le module est déjà utilisé et que vous venez de passer à une version plus récente, le nouveau module pourrait se comporter différemment de l'ancien module, et des modifications **pourraient** être apportées. C'est pourquoi vous devez toujours tester une nouvelle version d'un module dans un environnement différent (autre que celui de production) avant de l'installer en production. Vous pouvez tester sur une hôte de test pour vous assurer que toutes les modifications sont correctes, avant de promouvoir le module dans l'environnement de production. Cette pratique de de test sur une hôte de test et dans un environnement de test sera couvert dans un lab ultérieure.

Puppet depuis la ligne de commande

Explorez la commande «Puppet» à la fois sur le master et sur l'agent. Exécutez les commandes suivantes et prenez note de ce qu'ils font.

```
puppet help
puppet help agent
puppet agent -t
puppet agent -t --noop
puppet agent -t --debug
puppet help config
puppet config print
puppet config print manifest
puppet config print environment
puppet config print modulepath
puppet config print certname
```

Le fichier de configuration de Puppet

Regardons un peu **puppet.conf** ... Voici ce qu'un puppet.conf minimal côté agent qui ressemblerait à ceci:

```
[main]
   server = puppet.example.com
[agent]
   environment = production
   certname = agent.example.com
```

Lorsque vous exécutez puppet à partir de la ligne de commande, comment sait-il à quel master parler? Par défaut, Puppet recherchera simplement un Puppet Master appelé 'puppet'. Si le puppet.conf ne spécifie pas le serveur auquel parler, l'agent utilisera le résolveur de systèmes /etc/resolv.conf pour rechercher uniquement 'puppet', et si il est capable de résoudre ce nom, il utilisera l'adresse IP renvoyée.

Remarque: le puppet.conf est conforme au format "INI" trouvé historiquement sur les systèmes MS-DOS/Windows. Il comporte des sections. Nom de chaque **section** est entre crochets (par exemple, **[main]**) Chaque section peut contenir n'importe quel nombre de paires **nom/valeur**, qui sont les paramètres mappés dans cette section. Les paires nom/valeur sont séparées par un signe égal.

Plutôt que de dépendre du comportement par défaut, nous pouvons définir explicitement le nom du serveur dans puppet.conf comme suit:

```
[main]
server = puppet.example.com
```

Un autre élément de configuration important dans le puppet.conf est le **certname**. Le nom du certificat correspond généralement au nom de domaine complet de l'hôte, mais vous pouvez choisir de le définir comme vous le souhaitez.

```
[agent]
certname = agent.example.com
```

Le nom du certificat est ce qui serait utilisé lorsque l'agent contacte le master pour la première fois, et soumet une demande de signature de certificat SSL. C'est le nom qui apparaîtra lorsque vous exécutez `puppet cert list` sur le master comme suit:

```
puppet cert list --all
```

Par défaut, l'agent s'exécutera dans l'environnement de 'production'.

L'environnement peut également être remplacé dans puppet.conf, on serait alors en mesure de spécifier l'environnement côté agent dans son `/etc/puppetlabs/puppet/puppet.conf` comme suit:

```
[agent]
environment = production
```

En fait, selon la version de Puppet que vous utilisez, cela pourrait casser votre agent Puppet, vous obligeant à modifier manuellement le `puppet.conf` pour ramener l'environnement à **production**.

Sections du puppet.conf

Les sections *main*, *master* et *agent* contrôlent où un paramètre est applicable. La section **[agent]** s'applique à tout agent puppet, y compris l'agent s'exécutant sur master. Cependant, de nombreux paramètres que vous trouvez généralement dans la section [agent] sur un agent, peuvent également être spécifié dans la section [main] ou [master]. Tout ce que vous mettez dans la section **[main]** s'appliquera à la fois au master et à l'agent. Tous les paramètres que vous définissez dans la section **[master]** s'appliqueront uniquement au Puppet Master, et tous les paramètres de la section [agent] s'appliquent uniquement à l'agent de Puppet Master lui-même.

Si vous modifiez le fichier puppet.conf, vous devez également **redémarrer** le service puppet pour qu'il relise sa configuration.

D'accord, alors comment exécuter Puppet manuellement?

L'agent Puppet s'exécute automatiquement en arrière-plan

- Toutes les 30 minutes si son certificat a été signé, ou ...
- Toutes les 5 minutes s'il attend que son certificat soit signé

Cependant, vous pouvez également exécuter l'agent Puppet manuellement. Vous allez découvrir que c'est quelque chose que vous ferez beaucoup lorsque vous développerez un code de Puppet, et tester qu'il fait ce que vous voulez. Exécutez simplement:

```
puppet agent -t
```

L'option **-t** ou **-test** indique à l'agent Puppet Master de s'exécuter en "mode test" qui active plusieurs autres options pour tester votre code: 'onetime', 'verbose', 'ignorecache', 'no-daemonize', 'show_diff' et d'autres.

Parlons de ce qui se passe lorsque vous exécutez **puppet agent -t**. Celles-ci sont les éléments clés qui se produisent, et dans cet ordre:

1. L'agent lit son fichier puppet.conf (s'il est exécuté manuellement)
2. L'agent télécharge et met en cache les types personnalisés et les facts personnalisés pour les modules installés à partir du master
3. L'agent envoie des **facts** sur lui-même au Puppet Master (y compris l'environnement dont il fait partie)
4. Puppet Master utilise des variables (Facts et Puppet variables) avec le code de Puppet pour compiler le catalogue pour l'agent
5. Le master renvoie le catalogue à l'agent pour qu'il soit appliqué
6. L'agent applique le catalogue au système

Si jamais vous devez déterminer pourquoi une exécution de Puppet échoue, il sera très utile pour pouvoir identifier à quel point l'exécution échoue.

Faisons en sorte que puppet gère le fichier /etc/hosts

Une chose ennuyeuse à propos de Vagrant est que si vous configurez la VM avec un nom d'hôte, Vagrant éditera automatiquement /etc/hosts pour s'assurer que l'entrée du nom d'hôte est là, mais il ajoute le nom d'hôte à la ligne '127.0.0.1 localhost'. Ce n'est pas ce que nous voulons. Et chaque fois que nous arrêtons et démarrons la VM avec Vagrant, elle réécrit le fichier hosts de cette façon!

MISE À JOUR: c'était **le cas** des anciennes versions de Vagrant, mais des versions plus récentes semblent résoudre ce problème. Dans tous les cas, nous allons continuez avec notre exemple, car c'est toujours un bon exemple ...

Nous voulons notre /etc/hosts doit avoir les 4 lignes suivantes, et seulement ces 4 lignes:

```
127.0.0.1      localhost
192.168.198.10 puppet.example.com puppet
192.168.198.11 agent.example.com  agent
192.168.198.12 gitlab.example.com gitlab
```

Nous savons que Puppet commence par **site.pp** dans chaque exécution. Le site.pp contient des définitions de nœuds (qui indiquent à Puppet quel code appliquer à quels nœuds), ainsi que des variables de premier ordre et un peu de code de Puppet.

Connectez-vous au Puppet Master, devenez root et cd à: /etc/puppet/manifests

```
cd /etc/puppet/manifests
```

Modifiez **site.pp** et ajoutez ce qui suit à la fin du fichier dans la section **node default**:

```
vi site.pp
```

```
node default {

# remove all unmanaged resources
resources { 'host': purge => true }

# add some host entries
host { 'localhost':          ip => '127.0.0.1', }
host { 'puppet.example.com': ip => '192.168.198.10', host_aliases => [ 'puppet' ] }
host { 'agent.example.com':  ip => '192.168.198.11', host_aliases => [ 'agent' ] }
host { 'gitlab.example.com': ip => '192.168.198.12', host_aliases => [ 'gitlab' ] }
}
```

Ensuite, faites un **puppet agent -t** à la fois sur le master et sur l'agent. Parce que nous mettons ce code dans la définition 'node default', il sera appliqué à tout noeud qui ne correspond pas à une définition de nœud plus spécifique (telle que 'node agent {}' ou 'node puppet {}').

```
[root@puppet manifests]# puppet agent -t
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Loading facts
Info: Caching catalog for puppet.example.com
Info: Applying configuration version '1477078262'
Notice: Finished catalog run in 5.23 seconds
```

Rien n'a changé?

N'oubliez pas que dans un lab précédent, nous avons déjà configuré notre fichier d'hôtes. Puppet l'a regardé et trouvé que c'était déjà correct (selon la config que nous avons mis dans le site.pp) donc il n'a fait aucun changement.

Jetez un œil au fichier /etc/hosts sur le nœud **puppet** et notez qu'il a la même apparence comme avant:

```
127.0.0.1         localhost
192.168.198.10    puppet.example.com    puppet
192.168.198.11    agent.example.com     agent
192.168.198.12    gitlab.example.com    gitlab
```

Les entrées de l'hôte peuvent être dans un ordre différent, mais les 4 mêmes entrées devraient être présentes et, espérons-le, aucune autre. L'option `purge => true` que nous avons utilisée dans notre code indique à Puppet de supprimer toutes les entrées d'hôte non gérées. Si vous ajoutez une nouvelle entrée d'hôte en dehors du code de Puppet, la prochaine fois que Puppet s'exécutera, elle la supprimera.

Essayez ceci: ajoutez manuellement ce qui suit à /etc/hosts à l'aide de votre éditeur de texte préféré:

```
1.2.3.4 unmanaged-host
```

Puis lancez `puppet agent -t` et voyez ce que fait Puppet ...

```
[root@agent ~]# echo '1.2.3.4 unmanaged-host' >> /etc/hosts
```

Notez que nous pouvons résoudre ce nom maintenant:

```
[root@agent ~]# getent hosts unmanaged-host
1.2.3.4         unmanaged-host
```

Maintenant, lancez Puppet et regardez ce qu'il fait ...

```
[root@agent ~]# puppet agent -t
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Loading facts
Info: Caching catalog for agent.example.com
Info: Applying configuration version '1477078872'
Notice: /Stage[main]/Main/Node[default]/Host[unmanaged-host]/ensure: removed
Info: Computing checksum on file /etc/hosts
Notice: Finished catalog run in 0.40 seconds
```

Il a supprimé l'entrée non gérée que nous avons ajoutée! Agréable!

```
[root@agent ~]# cat /etc/hosts
# HEADER: This file was autogenerated at 2016-10-21 19:41:12 +0000
# HEADER: by puppet.  While it can still be managed manually, it
# HEADER: is definitely not recommended.
127.0.0.1    localhost
192.168.198.10    puppet.example.com    puppet
192.168.198.11    agent.example.com    agent
192.168.198.12    gitlab.example.com    gitlab
```

Nous avons donc vu que Puppet a reconnu une modification apportée à /etc/hosts et l'a annulée.

La ressource hôte

La définition de `host { }` est appelée un **puppet resource type**, ou simplement 'host resource' ou un 'type'. En fonction de la documentation que vous lisez, l'auteur peut l'appeler un «type» ou une «ressource» ou un «type de ressource». C'est la même chose. Pour rendre les choses encore plus déroutantes (ou intéressantes!), Vous pouvez même définir votre propres types de ressources personnalisés avec la fonction «define».

Pour une liste complète de toutes les `ressources types` intégrées disponibles, voir cette page:

http://docs.puppetlabs.com/puppet/latest/reference/type.html

Vous pouvez également utiliser la commande **puppet describe** pour obtenir des informations sur une ressource dans la ligne commande. Par exemple, si vous voulez voir les paramètres/attributs disponibles pour le type de ressource `host`, exécutez simplement:

```
puppet describe host
```

Comparez cela avec la ** référence des types ** ici: http://docs.puppetlabs.com/puppet/latest/reference/type.html#host

Si vous souhaitez voir l'état actuel d'un type de ressource particulier, vous pouvez utiliser la commande **puppet resource**. Par exemple, si vous voulez pour voir toutes les ressources hôte sur un système (pas nécessairement gérées par Puppet Master), vous pourriez faire:

```
puppet resource host
```

... et vous verriez quelque chose comme ceci:

```
host { 'gitlab.example.com':
  ensure       => 'present',
  host_aliases => ['gitlab'],
  ip           => '192.168.198.12',
  target       => '/etc/hosts',
}
host { 'agent.example.com':
  ensure       => 'present',
  host_aliases => ['agent'],
  ip           => '192.168.198.11',
  target       => '/etc/hosts',
}
host { 'localhost':
  ensure => 'present',
  ip     => '127.0.0.1',
  target => '/etc/hosts',
}
host { 'puppet.example.com':
  ensure       => 'present',
  host_aliases => ['puppet'],
  ip           => '192.168.198.10',
  target       => '/etc/hosts',
}
```

Si vous souhaitez voir l'état des ressources pour un utilisateur particulier, vous pourrait exécuter:

```
puppet resource user root
```

... et vous verriez quelque chose comme ceci:

```
user { 'root':
       ensure           => 'present',
       comment          => 'root',
       gid              => '0',
       home             => '/root',
       password         => '$1$v4K9E7xJ$gZIhJ5Jtqg5ZgZXeqSShd0',
       password_max_age => '99999',
       password_min_age => '0',
       shell            => '/bin/bash',
       uid              => '0',
}
```

Encore une fois, ces ressources ne sont pas nécessairement gérées par le Puppet Master (mais ils pourraient être). Nous ne savons tout simplement pas. Lorsque vous exécutez Puppet de cette manière, vous interrogez simplement l'état du système à ce moment précis. C'est ainsi que Puppet **voit** l'état du système.

Il est utile d'afficher les ressources de cette manière lorsque vous écrivez un code Puppet pour appliquer un état de configuration, car vous pouvez facilement couper et coller le code pour la ressource dans un manifest et modifiez-le en conséquence.

Pour mieux comprendre cela, examinons une ressource de service et comment la sortie de `puppet resource service puppet` change lorsque nous arrêtons/démarrons ou activer/désactiver un service.

Voyons l'état actuel du service Puppet sur notre puppetmaster:

```
[root@puppet ~]# puppet resource service puppet
   service { 'puppet':
   ensure => 'running',
   enable => 'true',
}
```

Nous voyons que le service de Puppet est à la fois activé et en cours d'exécution.

Nous pouvons également le confirmer avec la commande **systemctl** (sur RHEL7):

```
[root@puppet ~]#  systemctl status puppet
● puppet.service - Puppet agent
Loaded: loaded (/usr/lib/systemd/system/puppet.service; enabled; vendor preset:
disabled)
Active: active (running) since Mon 2016-11-14 18:17:39 UTC; 5h 11min ago
…
```

Maintenant, désactivons le service, mais laissons-le en cours d'exécution, puis affichons la sortie de la ressource Puppet à nouveau. Notez que l'attribut **enabled** est maintenant devenu **false**.

```
[root@puppet ~]# systemctl disable puppet
Removed symlink /etc/systemd/system/multi-user.target.wants/puppet.service.
```

Vérifiez maintenant la ressource du service de Puppet:

```
[root@puppet ~]# puppet resource service puppet
service { 'puppet':
   ensure => 'running',
   enable => 'false',
}
```

Arrêtons le service, puis regardons à nouveau. Notez que l'attribut **ensure** est maintenant devenu **stopped**.

```
[root@puppet ~]# systemctl stop puppet
```

Maintenant, vérifiez à nouveau la ressource du service de Puppet:

```
[root@puppet ~]# puppet resource service puppet
service { 'puppet':
   ensure => 'stopped',
   enable => 'false',
}
```

Maintenant, réactivez et redémarrez le service, puis regardez à nouveau ...

```
[root@puppet ~]# systemctl enable puppet
Created symlink from /etc/systemd/system/multi-user.target.wants/puppet.service to
/usr/lib/systemd/system/puppet.service.
[root@puppet ~]# systemctl start puppet
```

Maintenant, revérifions ...

```
[root@puppet ~]# puppet resource service puppet
service { 'puppet':
   ensure => 'running',
   enable => 'true',
}
```

... et nous sommes revenus à notre point de départ. Notez que la commande **puppet resource service puppet** montre simplement l'état actuel de cette ressource à ce moment précis.

Bon, revenons à la ressource host ...

Rappelez-vous que `puppet describe host` vous montrera tous les paramètres pour une ressource host.

Une ressource host a un titre et plusieurs paramètres/attributs. Le premier élément dans la définition de la ressource host est le titre (par exemple `puppet.example.com`) et le les paires nom/valeur suivantes sont les paramètres (par exemple, `ip => 192.168.198.10`). Dans ce contexte, le terme **attributs** peut être meilleur que **paramètres**, car nous utilisons également des paramètres lors de la définition des classes, des paramètres de classe et les attributs de type sont deux choses différentes.

Notez que l'attribut **host_aliases** accepte un array (entre crochets `[]`), tandis que les attributs **ensure** et **ip** acceptent une chaîne.

Si vous ne l'avez pas déjà fait, jetez un œil au contenu du fichier /etc/hosts:

```
cat /etc/hosts
```

Et nous devrions voir quelque chose comme ça ...

```
# HEADER: This file was autogenerated at 2016-01-19 18:25:19 +0000
# HEADER: by puppet.  While it can still be managed manually, it
# HEADER: is definitely not recommended.
127.0.0.1       localhost
192.168.198.10  puppet.example.com  puppet
192.168.198.11  agent.example.com   agent
192.168.198.12  gitlab.example.com  gitlab
```

À ce stade, chaque ligne existante dans /etc/hosts est gérée par puppet. Le type de ressource host gère chaque ligne individuellement, et nous avons ajouté chacune des lignes existantes avec un code de Puppet.

Souvenez-vous également que nous avons ajouté le morceau de code suivant pour supprimer toute entrée d'hôte qui n'est pas gérée par puppet:

```
resources { 'host': purge => true }
```

... en d'autres termes, nous avons maintenant un fichier /etc/hosts entièrement géré. Si quelqu'un ajoute une ligne au fichier /etc/hosts sans réalisant que Puppet le gère, la prochaine fois que Puppet s'exécute, il aura une surprise!

La bonne chose à propos de la gestion de chaque entrée de /etc/hosts avec puppet est que si notre fichier d'hôtes est supprimé ou corrompu d'une manière ou d'une autre, Puppet le réécrit entièrement, et nous n'avons pas à nous soucier des entrées disparues. Essayons cela sur le nœud **agent** ...

```
[root@agent ~]# rm -f /etc/hosts
```

Remarque: nous avons besoin d'au moins une entrée pour le Puppet Master lui-même, sinon l'agent ne saura pas à quelle adresse IP parler. C'est une excellente raison d'utiliser le DNS (pour que l'adresse IP de notre Puppet Master soit **toujours** résolue).

```
[root@puppet ~]# echo '192.168.198.10 puppet.example.com puppet' > /etc/hosts
[root@puppet ~]# puppet agent -t
Info: Using configured environment 'production'
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Loading facts
Info: Caching catalog for puppet.example.com
Info: Applying configuration version '1479166434'
Notice: /Stage[main]/Main/Node[default]/Host[localhost]/ensure: created
Info: Computing checksum on file /etc/hosts
Notice: /Stage[main]/Main/Node[default]/Host[agent.example.com]/ensure: created
Notice: /Stage[main]/Main/Node[default]/Host[gitlab.example.com]/ensure: created
Notice: Applied catalog in 13.83 seconds
```

Voyez comment Puppet a rajouté toutes les entrées d'hôte manquantes?

```
[root@puppet ~]# cat /etc/hosts
# HEADER: This file was autogenerated at 2016-11-14 23:34:05 +0000
# HEADER: by puppet.  While it can still be managed manually, it
# HEADER: is definitely not recommended.
192.168.198.10  puppet.example.com  puppet
127.0.0.1       localhost
192.168.198.11  agent.example.com   agent
192.168.198.12  gitlab.example.com  gitlab
```

Allez-y et essayez d'ajouter des entrées supplémentaires à /etc/hosts, puis exécutez à nouveau puppet. Vous devriez voir Puppet les supprimer:

J'ai ajouté cette ligne:

```
1.2.3.4   foo.example.com foo
```

Ensuite, Puppet est exécuté et l'a enlevée:

```
[root@agent /]# puppet agent -t
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Loading facts
Info: Caching catalog for agent.example.com
Info: Applying configuration version '1472432976'
Notice: /Stage[main]/Main/Node[default]/Host[foo.example.com]/ensure: removed
Info: Computing checksum on file /etc/hosts
Notice: Finished catalog run in 0.55 seconds
```

Revoyons nos objectifs dans cette section:

- Fichier de configuration Puppet: nous avons examiné certaines des configurations importantes
- Puppet en ligne de commande: nous avons utilisé 'puppet help', 'puppet agent -t', etc.
- Bases du code de Puppet: nous avons eu un petit aperçu du code de Puppet
- Cibler un Code à un nœud via "classification": nous avons utilisé la définition de nœud par défaut dans le `site.pp`

Lectures complémentaires

Pour plus d'informations sur la configuration et l'utilisation du fichier **main manifest** site.pp , voir la documentation PuppetLabs à:

https://docs.puppetlabs.com/puppet/latest/reference/dirs_manifest.html

Pour plus d'informations sur l'utilisation du modulepath:

https://docs.puppetlabs.com/puppet/latest/reference/dirs_modulepath.html#loading-content-from-modules