Mise en place de Puppet 3.7.2

Dans ce tutoriel il sera question du service centralisé de déploiement et de gestion de service, Puppet. Via ce service j'expliquerai dans un premier temps comment déployer l'infrastructure de base puis la mise en place de divers services.

Table des matières :

1. Pré-requis :

Pour commencer, avant même de parler des notions abordées lors de ce tutoriel un point de vocabulaire est utile afin de s'y retrouver.

- **Puppet Master :** Ce que je vais appeler *Puppet Master* va correspondre au serveur Puppet, la machine sur laquelle toutes les configurations seront définis.
- **Node :** les *Nodes* sont les clients Puppet, donc toutes les machines qui iront chercher leurs configurations sur Puppet Master.
- **Manifest :** Les *Manifests* sont les fichiers de configuration sur le *Puppet Master* dans lesquelles on va définir la configuration des services à déployer sur les *Nodes*. Leurs extensions sont « .pp ».
- **Module :** Les *Modules Puppet*, installés sur *Puppet Master* sont des ensembles de *manifests/templates* créés par la communauté permettant de configurer plus simplement les services à déployer.

Les services que je vais déployer seront les suivants :

- ***Sur tous les nodes :*** L'éditeur de texte Vim ainsi qu'un service NTP.
- ***Un par node :*** Un service web Apache2 avec deux Virtualhosts (un en HTTP et un en HTTPS), Le CMS Wordpress ce qui inclut un autre service web Apache2 et un service de base de données MySQL et enfin un service DHCP.

Pour déployer tout cela plusieurs machines seront utilisées lors de ce tutoriel, voici la liste :

- ***puppetdns.puppet.test :*** Service DNS qui permettra aux Nodes de joindre Puppet Master.
- ***puppetmaster.puppet.test :*** Puppet Master, machine sur laquelle sera centralisés tous les services à déployer.
- ***puppetapache.puppet.test :*** Node où sera installé et configuré un service web Apache2 avec deux virtualhost, ***« web »*** qui sera en HTTP et ***« webssl »*** qui sera en HTTPS.
- ***puppetapachewp.puppet.test :*** Deuxième service web Apache2 sur lequel sera installé Wordpress.
- ***puppetmysql.puppet.test :*** Service de base de données MySQL servant pour Wordpress.
- ***puppetdhcp.puppet.test :*** Service DHCP Isc-dhcp-server.
- ***puppetclient.puppet.test :*** Node quelconque qui servira à faire des tests notamment pour les déploiements communs à tous les nodes tel que Vim ou encore le service NTP.
- ***puppetgraph.puppet.test :*** Client graphique non géré par Puppet Master qui permettra de vérifier le bon fonctionnement des services Apache2.

Hormis le client graphique qui sera sous Windows 7, toutes les autres machines seront sous Debian sans interface graphique.

2. Infrastructure de base :

Cette partie va donc concerner l'infrastructure de base, c'est à dire les éléments de l'infrastructure qui doivent être opérationnelles avant de s'attaquer directement aux divers déploiements de services.

2.1. Service DNS :

Avant toutes choses il faut un service DNS à jour pour pouvoir utiliser le service puppet. Pour la configuration de ce service sur la machine ***« puppetdns »,*** cela ne sera pas abordée car le service DNS ***« Bind9 »*** à déjà fait l'objet d'un tutoriel. Il est disponible à l'adresse suivante https://www.net-folio.ovh .

→ Juste pour un récapitulatif rapide, voici respectivement le fichier de zone et le fichier cache DNS en résolution direct où se trouve les différentes machines déployées.

```
zone "puppet.test" {
        type master;
        file "/var/cache/bind/db.puppet";
};
```

Le nom de domaine sera donc ***« puppet.test »***.

```
$TTL    86400
$ORIGIN puppet.test.
@       IN      SOA     puppetdns help.puppet.test. (
                              2         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                          86400 )       ; Negative Cache TTL
;
@       IN      NS      puppetdns
;Infrastructure de base
puppetdns       IN A    192.168.1.130
puppetmaster    IN A    192.168.1.120
;Nodes ou les services seront déployés
puppetapache    IN A    192.168.1.140
puppetapachewp  IN A    192.168.1.138
puppetmysql     IN A    192.168.1.139
puppetdhcp      IN A    192.168.1.150
;Nodes utilisés pour les tests
puppetgraph     IN A    192.168.1.141
puppetclient    IN A    192.168.1.142
;Virtualhosts HTTP/HTTPS
web     IN CNAME        puppetapache.puppet.test.
websslIN CNAME          puppetapache.puppet.test.
```

A savoir que deux noms canoniques (CNAME) sont déclarés pour les deux Virtualhosts qui seront utilisés plus bas pour un des services web deployés :

- *web.puppet.test:* Site web en HTTP.
- *webssl.puppet.test :* Site web en HTTPS.

2.2. PuppetMaster :

Pour cette partie il va être question de ce qu'il va falloir faire sur Puppet Master afin d'installer le service ***« puppetmaster »*** gérant le déploiement.

2.2.1. Extension Vim :

Avant toutes choses je conseille d'installer une extension Vim afin d'avoir la coloration syntaxique pour les manifests. Ceci est très utile pour aider à prévenir les petites erreurs syntaxiques.

Bien évidement l'éditeur de texte Vim doit être déjà installé sur la machine « *puppetmaster* ».

→ Pour installer une extension Vim il suffit de la télécharger*(en bleu)* puis de l'installer :*(en rouge)*

```
root@PuppetMaster:~# apt-get install vim-puppet
```

```
root@PuppetMaster:~# vim-addons install puppet
Info: installing removed addon 'puppet' to /root/.vim
```

l'extension Vim est désormais activés.

2.2.2. Service NTP :

Il faut maintenant installer un service NTP afin de permettre à Puppet Master de rester à l'heure, ceci est important pour la gestion des certificats.

→ Pour cela on installe le service NTP puis on édite le fichier de configuration ***« /etc/ntp.conf »*** afin de déclarer les serveurs de temps sur lesquels se mettre à jour :

```
root@PuppetMaster:~# apt-get install ntp
```

```
# pool.ntp.org maps to about 1000 low-stratum NTP servers.  Your server will
# pick a different set every time it starts up.  Please consider joining the
# pool: <http://www.pool.ntp.org/join.html>
server 0.fr.pool.ntp.org iburst
server 1.fr.pool.ntp.org iburst
server 2.fr.pool.ntp.org iburst
server 3.fr.pool.ntp.org iburst
```

Il faut modifier les 4 lignes ci-dessus*(en vert)* avec le nom des serveurs NTP appropriés. Le site *http://www.pool.ntp.org* recense et informe sur les serveurs NTP actifs.

L'option ***« iburst »*** signifie que le service réessaiera plusieurs fois de contacter un serveur NTP en cas d'échec.

Pour finir il suffit de redémarrer le service NTP. La commande ***« date »*** permet de donner l'heure de votre système.

```
root@PuppetMaster:~# date
lundi 31 juillet 2017, 03:03:31 (UTC+0200)
```

2.2.3. Service « puppetmaster » :

Maintenant tout est prêt pour l'installation du paquet ***« puppetmaster »***.

→ C'est une installation de paquet classique, une fois fait on exécute la commande ***« service puppetmaster status »*** pour une vérification rapide de l'installation :

```
root@PuppetMaster:~# apt-get install puppetmaster
```

```
root@PuppetMaster:~# service puppetmaster status
● puppetmaster.service - Puppet master
   Loaded: loaded (/lib/systemd/system/puppetmaster.service; enabled)
   Active: active (running) since mer. 2017-07-19 04:05:17 CEST; 9s ago
 Main PID: 3037 (puppet)
   CGroup: /system.slice/puppetmaster.service
           └─3037 /usr/bin/ruby /usr/bin/puppet master
```

Le service ***« puppetmaster »*** est maintenant installé, c'est tout ce qu'il y a à faire pour l'infrastructure de base. Ensuite il va falloir installer le service ***« puppet »*** sur tous les nodes gérés par Puppet Master.

2.3. Initialisation d'un node :

Cette partie sera donc à reproduire sur tous les nodes que l'on souhaite gérer par Puppet. Je ne reviendrai pas sur ces étapes dans le reste du tutoriel.

→ Donc sur le node on commence par installer le paquet ***« puppet »*** :

```
root@puppetclient:~# apt-get install puppet
```

Ce paquet permet la communication avec Puppet Master et entre autre de récupérer les manifests.

→ Une fois installé il faut éditer le fichier de configuration ***« /etc/puppet/puppet.conf »*** afin d'ajouter la ligne ci-dessous*(en rouge)* pour indiquer le Puppet Master :

```
[main]
logdir=/var/log/puppet
vardir=/var/lib/puppet
ssldir=/var/lib/puppet/ssl
rundir=/var/run/puppet
factpath=$vardir/lib/facter
prerun_command=/etc/puppet/etckeeper-commit-pre
postrun_command=/etc/puppet/etckeeper-commit-post
server=puppetmaster.puppet.test
```

Le nom DNS (FQDN) est à indiquer d'où l'importance d'avoir un serveur DNS opérationnel au préalable.

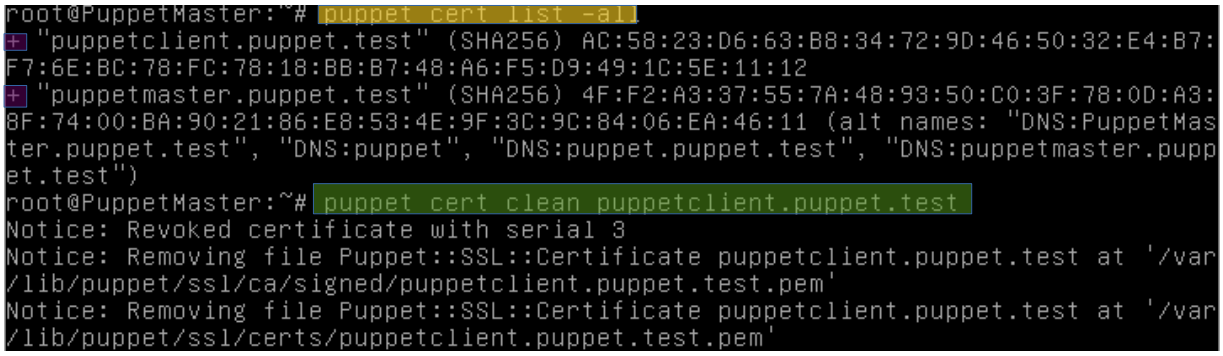
Après avoir redémarré le service ***« puppet »*** sur le node, il va falloir retourner sur le Puppet Master afin de signer le certificat. Les certificats vont permettent aux nodes de s'authentifier, ils sont indispensables sans cela ils ne pourront pas récupérer leurs manifests.

Je vais donc maintenant détailler plusieurs commandes liées à cette manipulation notamment une procédure de révocation en cas d'erreur ou d'expiration d'un certificat.

→ Pour valider le certificat généré par le node ***« puppetclient.puppet.test »*** sur le Puppet Master, il va falloir entrer les commandes ci-dessous :

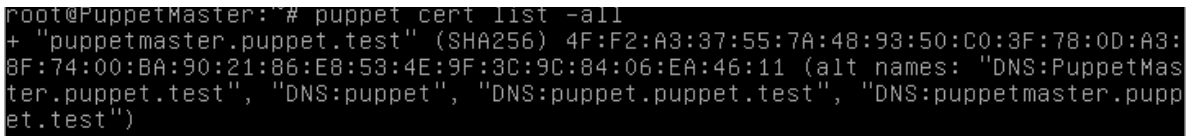
La commande ***« puppet cert list »*** permet d'afficher la liste des certificats en attente*(en rouge)*. On signe donc ce certificat avec la commande suivante*(en bleu)* et pour finir on affiche à nouveau la liste des certificats en attente afin de vérifier qu'il n'y est plus.

→ Pour révoquer un certificat la procédure est la suivante :



La commande pour supprimer un certificat sur Puppet Master est ***« puppet cert clean [Nom] »****(en vert)*. On note l'utilisation de l'argument ***« -all »****(en jaune)* qui nous permet d'obtenir tous les certificats, y compris ceux signés qu'on identifie par la présence d'un ***« + »****(en violet)*.

→ Ensuite on vérifie que le certificat supprimé n'est plus présent :



→ Puis sur le node on supprime le certificat*(en rouge)* et on le ré-génère :

```
root@puppetclient:~# find /var/lib/puppet/ssl -name puppetclient.puppet.test.pem
 -delete
root@puppetclient:~# puppet agent -t
Info: Creating a new SSL key for puppetclient.puppet.test
Info: csr_attributes file loading from /etc/puppet/csr_attributes.yaml
Info: Creating a new SSL certificate request for puppetclient.puppet.test
Info: Certificate Request fingerprint (SHA256): 6F:53:40:68:4A:C5:A2:A5:65:DF:5F
:19:FA:48:A4:35:7D:E7:A5:06:49:9D:FA:9F:5A:9D:26:35:6B:1A:58:D2
Exiting; no certificate found and waitforcert is disabled
```

A noter que les certificats utilisés sont auto-signés c'est à dire que l'autorité de certification dans notre cas est Puppet Master, donc dans le cas où la machine Puppet Master est changée c'est l'autorité de certification qui ne sera plus bonne pour le node. Pour palier à cela je conseille de directement supprimer le répertoire ***« /var/lib/puppet/ssl/ »*** sur le node afin de tout re-générer.

→ Une fois le certificat établi il faut bien penser à activer l'agent sur le node :

```
root@puppetclient:~# puppet agent --enable
```

2.4. Test de l'infrastructure de base :

Pour tester le bon fonctionnement de l'infrastructure de base il va falloir écrire un manifest sur Puppet Master. Ce manifest sera simple et permettra de déployer un fichier sur le node ***« puppetclient.puppet.test »***.

Pour commencer il faut créer le manifest principal, celui qui sera lu par défaut en premier par le service ***« puppetmaster »,*** à savoir le fichier ***« /etc/puppet/manifests/site.pp ».***

→ Dans ce manifest on va utiliser le ***« Ressource type » « file »*** afin de gérer le déploiement d'un fichier :

```
file {'/tmp/example-ip':
        ensure  => present,
        mode    => 0644,
        content => "This IP address is ${ipaddress_eth0}.\n",
}
```

Alors plusieurs choses sont à noter :

- ***/tmp/example-ip :*** Nom de la ressource. Dans ce cas cela indique également l'emplacement du fichier sur le node car l'attribut ***« path »*** n'est pas spécifié.
- ***ensure :*** Permet d'indiquer l'état du fichier. ***« Present »*** signifie que le fichier doit être présent ou créé dans le cas échéant. En remplaçant ***« Present »*** par ***« Absent »*** le fichier sera supprimé s'il est présent.
- ***mode :*** Attribut Permettant d'indiquer les droits à attribuer au fichier.
- ***content :*** Il s'agit simplement du contenu du fichier.

On remarque l'utilisation de la variable ***« ${ipaddress_eth0} »*** qui correspond à l'adresse IP de l'interface eth0 du node. La liste des variables est disponible via la commande ***« facter »***.

Comme on peux le voir dans cette situation de test le contenu du fichier est directement défini dans le manifest, ce qui n'est pas forcement souhaitable en situation réel. Nous verrons dans la suite de ce tutoriel comment utiliser plusieurs manifests.

Maintenant il ne reste plus qu'à faire remonter le manifest sur le node.

→ Pour cela il suffit d'utiliser la commande ***« puppet agent -t »*** :

```
root@puppetclient:~# puppet agent -t
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for puppetclient.puppet.test
Info: Applying configuration version '1500992704'
Notice: /Stage[main]/Main/File[/tmp/exemple-ip]/ensure: created
Notice: Finished catalog run in 0.10 seconds
```

La remonter c'est bien faite sans problème, on peut voir que le fichier déclaré dans le manifest ***« /etc/puppet/manifests/site.pp »*** a bien été créé.*(en rouge)*

→ Pour finir on va afficher le fichier en question pour s'assurer que tout c'est bien passé :

```
root@puppetclient:~# cat /tmp/exemple-ip
This IP address is 192.168.1.142.
```

C'est tout pour l'infrastructure de base, on va pouvoir passer au déploiement de services.

3. Structure des manifests :

Dans cette partie je vais détailler la structure des manifests et leurs mises en place.

En effet, si comme on l'a vu précédemment le manifest ***« /etc/puppet/manifests/site.pp »*** une fois créé est automatiquement lu pas le service puppet, ce n'est pas le cas d'autres manifests que l'on pourrais créer. De plus tout déclarer dans un unique manifest n'est pas non plus souhaitable.

La structure des manifests suivante est un choix personnelle, une des forces de Puppet réside dans sa modularité, en fonction des services à déployer ou encore de l'infrastructure cette structure peut et doit évoluer pour être la plus adaptée possible.

- ***« /etc/puppet/manifests/site.pp » :*** Ce manifest va servir à lier les autres manifests.
- ***« /etc/puppet/manifests/nodes.pp » :*** Déclaration des nodes et lien vers les classes adéquates.
- ***« /etc/puppet/manifests/basics.pp » :*** Contient la classe concernant tous les nodes.
- ***« /etc/puppet/manifests/apacheVH.pp » :*** Classe définissant le service apache2 avec Virtualhosts.
- ***« /etc/puppet/manifests/apacheWP.pp » :*** Classe pour le service apache2 utilisé pour Wordpress.
- ***« /etc/puppet/manifests/mysql.pp » :*** Classe pour le déploiement du service MySQL pour Wordpress.
- ***« /etc/puppet/manifests/dhcp.pp » :*** Classe pour le service DHCP isc-dhcp-server.

```
root@PuppetMaster:/etc/puppet/manifests# ls
apacheVH.pp  apacheWP.pp  basics.pp  dhcp.pp  mysql.pp  nodes.pp  site.pp
```

→ Pour lier tous les manifests créés il va falloir éditer le manifest ***« /etc/puppet/manifests/site.pp »*** :

```
#Declaration des Nodes
import "nodes.pp"
#Classe concernant tous les Nodes
import "basics.pp"
#Classe Apache2 Virtualhosts HTTP et HTTPS
import "apacheVH.pp"
#Classe Apache2 pour Wordpress
import "apacheWP.pp"
#Classe MySQL pour Wordpress
import "mysql.pp"
#Classe DHCP
import "dhcp.pp"
```

Sans l'importation des autres manifests, ils ne seraient pas lus par le service ***« puppetmaster »***.

La structure des manifests est maintenant opérationnelle.

4. Manifest « nodes.pp » :

Ensuite on va définir les différents nodes avec leurs classes respectives dans le manifest dédié à cela.

→ Pour cela on édite donc le manifest ***« /etc/puppet/manifests/nodes.pp »*** :

```
node default {
        include basics
}
node 'puppetapache.puppet.test' {
        include apacheVH
}
node 'puppetapachewp.puppet.test' {
        include apacheWP
}
node 'puppetmysql.puppet.test' {
        include mysql
}
node 'puppetdhcp.puppet.test' {
        include dhcpserv
}
```

Il n'y a rien de spécial à dire si ce n'est qu'avec la nomination ***« default »*** on implique tous les nodes et qu'on spécifie les classes définies dans les manifests avec la fonction ***« include »***.

5. Déploiement basique :

Le premier manifest dont je vais m'occuper va concerner les éléments à déployer sur tous les nodes.

5.1. Vim :

L'éditeur de texte Vim va être installé sur tous les nodes.

→ Pour commencer, il va falloir installer le module ***« saz-vim »*** via Puppet sur Puppet Master :*(en rouge)*

```
root@PuppetMaster:/etc/puppet/manifests# puppet module install saz-vim
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└── saz-vim (v2.6.0)
```

Comme on peut le voir le module a bien été installé, le répertoire de ce module est également indiqué et contient des manifests et templates dont on va se servir par la suite dans le manifest ***« /etc/puppet/manifests/basics.pp ».***

Plusieurs commandes peuvent êtres utiles pour la gestion des modules Puppet comme ***« puppet module search [name] »*** pour rechercher un module en particulier, ***« puppet module list »*** qui donne la liste des modules installés ou encore ***« puppet module uninstall [name] »*** pour désinstaller un module.

→ Maintenant on va déclarer dans le manifest adéquate, le déploiement de Vim et sa configuration :

```
#Manifest Basics
class basics {
#Module Vim
        class { 'vim':
                opt_syntax      => true,
        }
}
```

Dans le manifest ***« /etc/puppet/manifests/basics.pp »*** on crée donc la classe ***« basics »*** dans laquelle on va indiquer tout ce qu'il y a à déployer sur tous les nodes.

Pour Vim, il faut donc appeler la classe de référence ***« vim »*** fournie par son module permettant l'installation de l'éditeur de texte en question. L'attribut ***« opt_syntax »*** permet d'activer la coloration syntaxique.

5.2. NTP

Je souhaite également déployer un service NTP sur tous les nodes, il faut donc ajouter dans le manifest **« /etc/puppet/manifests/basics.pp »** de quoi le faire.

→ Avant cela il va falloir falloir installer le module ***« saz-ntp »*** :

```
root@PuppetMaster:~# puppet module install saz-ntp
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└── saz-ntp (v2.5.0)
```

→ Une fois installé on peut repasser à l'édition du manifest ***« /etc/puppet/manifests/basics.pp »*** :

```
#Module NTP
        class { 'ntp':
                server_list     => [
                        '0.fr.pool.ntp.org iburst',
                        '1.fr.pool.ntp.org iburst',
                        '2.fr.pool.ntp.org iburst',
                        '3.fr.pool.ntp.org iburst',
                ],
        }
```

On déclare donc dans la classe créée précédemment***(« basics »)*** une nouvelle classe de référence pour le service NTP dans laquelle on spécifie la liste des serveurs NTP à joindre via l'attribut ***« server_list »***.*(en rouge)*

C'est tout pour le manifest ***« /etc/puppet/manifests/basics.pp »,*** il faut maintenant faire remonter ce manifest sur les nodes grâce à la commande ***« puppet agent -t ».***

5.3. Test :

Il faut maintenant tester le bon fonctionnement du manifest, pour cela j'utiliserai le node ***« puppetclient.puppet.test »***.

→ Pour Vim on va directement dans le fichier de configuration ***« /etc/vim/vimrc »***, cela va nous indiquer s'il est bien installé et si la coloration syntaxique a été prise en compte :

On peut voir que l'éditeur de texte est bien géré par Puppet et qu'il fonctionne correctement.

```
" vimrc: Managed by puppet - DO NOT EDIT
" Changes manually to this file can, and will, be lost

set nocompatible

runtime! debian.vim

if has("syntax")
  syntax on
  set background=dark
endif
```

→ Pour le service NTP il suffit de vérifier son état avec la commande ***« service ntp status »*** :

```
root@puppetclient:~# service ntp status
● ntp.service - LSB: Start NTP daemon
   Loaded: loaded (/etc/init.d/ntp)
   Active: active (running) since lun. 2017-07-31 12:15:02 CEST; 1s ago
  Process: 2622 ExecStop=/etc/init.d/ntp stop (code=exited, status=0/SUCCESS)
  Process: 2630 ExecStart=/etc/init.d/ntp start (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/ntp.service
           └─2638 /usr/sbin/ntpd -p /var/run/ntpd.pid -g -u 109:114
```

Le service est bien démarré, ce qui implique que le service NTP est bien installé.

6. Déploiement d'apache2 avec Virtualhosts *:*

6.1. Apache2 :

Je vais maintenant aborder le déploiement automatique d'un service ***« apache2 »*** relativement complet comprenant deux Virtualhosts, dont un en HTTPS.

→ Comme pour Vim, il faut déjà installer un module, à savoir ***« puppetlabs-apache »*** :*(en rouge)*

```
root@PuppetMaster:/etc/puppet/manifests# puppet module install puppetlabs-apache
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└─┬ puppetlabs-apache (v1.11.0)
  ├── puppetlabs-concat (v2.2.1)
  └── puppetlabs-stdlib (v4.17.1)
```

Une fois le module installé, il va falloir définir le déploiement et la configuration du service ***« apache2 »*** à déployer.

Le manifest ***« /etc/puppet/manifests/apacheVH.pp »*** est plutôt long je l'ai donc découpé en plusieurs parties pour plus de clarté.

→ Donc pour commencer on déclare une nouvelle classe nommée ***« apacheVH »*** :

```
#Manifest ApacheVH
class apacheVH inherits apache {

}
```

Cette nouvelle classe hérite de la classe de référence ***« apache »*** fournie par le module ***« puppetlabs-apache ».***

→ Ensuite il faut ajouter à cette classe le premier Virtualhost, ***« web.puppet.test »*** qui sera sans SSL :

```
#Virtualhost web - sans HTTPS
        apache::vhost { 'vhostweb':
                servername      => 'web.puppet.test',
                port    => '80',
                docroot => '/var/www/html/web',
        }

        #Fichier index.html web sans ssl
        file { '/var/www/html/web/index.html':
                ensure  => file,
                mode    => 644,
                owner   => 'www-data',
                group   => 'www-data',
                source  => 'puppet:///modules/apache/web/index.html',
        }
```

En rouge on déclare le premier Virtualhost nommé ***« vhostweb »*** avec respectivement le FQDN du site, le port du service web et enfin le répertoire de publication dans lequel les fichiers du site ***« web.puppet.test »*** se trouveront.

En bleu on télécharge dans le répertoire de publication du node ***« puppetapache.puppet.test »*** le fichier ***« index.html »***. Ce fichier sera créé sur Puppet Master dans le répertoire ***« /etc/puppet/module/apache/files/web/ »*** qui est indiqué via l'attribut ***« source »***. A noter qu'on utilise le protocole puppet marqué par ***« puppet:/// »*** ce qui permet de ne pas indiquer le chemin complet, le répertoire ***« files »*** n'est donc pas à indiquer.

→ On passe ensuite au Virtualhost ***« webssl.puppet.test »*** :

```
#Virtualhost webssl - HTTPS
        apache::vhost { 'vhostwebssl':
                servername      => 'webssl.puppet.test',
                port    => '443',
                ssl     => true,
                docroot => '/var/www/html/webssl',
        }

        #Fichier index.html web ssl
        file { '/var/www/html/webssl/index.html':
                ensure  => file,
                mode    => 644,
                owner   => 'www-data',
                group   => 'www-data',
                source  => 'puppet:///modules/apache/webssl/index.html',
        }
```

Le nouveau Virtualhost est déclaré de la même manière que pour le premier, hormis le port qui est cette fois celui du protocole HTTPS, ***« 443 »****(en jaune)* et l'apparition d'une nouvelle directive appelée ***« ssl »*** qui est à définir sur ***« true »*** afin d'activer le chiffrement.*(en vert)*

Puis à l'instar du Virtualhost ***« web.puppet.test »,*** on télécharge le fichier ***« index.html »*** dans le répertoire de publication adéquate du node.

Pour finir avec ce manifest, on va mettre en place une redirection permettant de renvoyer toutes les requêtes pour l'URL ***« http://webssl.puppet.test »*** vers ***« https://webssl.puppet.test »***.

→ Pour cela on déclare simplement un nouveau Virtualhost basique :

```
#Virtualhost redirect webssl
        apache::vhost { 'vhostredirect':
                servername      => 'webssl.puppet.test',
                port    => '80',
                redirect_status => 'permanent',
                redirect_dest   => 'https://webssl.puppet.test',
                docroot => '/var/www/html/webssl',
        }
```

Je l'ai nommé ***« vhostredirect »*** et deux attributs sont à ajouter, ***« redirect_status »*** ainsi que ***« redirect_dest »***.

Maintenant que le manifest est prêt, il ne reste que quelques détails à s'occuper pour son bon fonctionnement.

→ Il faut créer les deux fichiers ***« index.html »****(en rouge)* afin qu'ils puissent être téléchargés sur le node :

```
root@PuppetMaster:/etc/puppet/modules/apache/files# ls -R
.:
httpd  web  webssl

./web:
index.html

./webssl:
index.html
```

Ici j'ai donc créé les deux répertoires correspondants aux deux Virtualhosts et dans lesquels se trouve un fichier ***« index.html »***.

→ C'est fichier ***« index.html »*** sont basiques et sont là pour les tests :

```
<html>
        <body bgcolor='red'>
                <p>Virtualhost Web</p>
        </body>
</html>
```

```
<html>
        <body bgcolor='green'>
                <p>Virtualhost Webssl</p>
        </body>
</html>
```

Pour finir il ne restera plus qu'à faire remonter le manifest sur le node ***« puppetapache.puppet.test »*** avec la commande ***« puppet agent -t »***.

```
root@PuppetApache:~# puppet agent -t
```

6.2. Test :

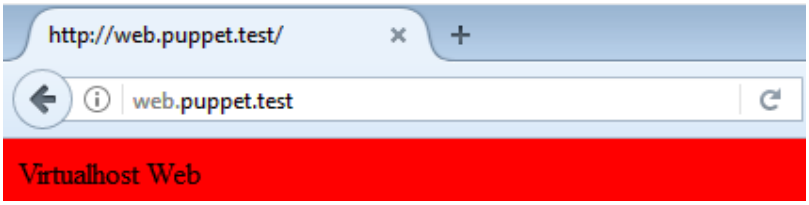
Une fois que le service déployé, il faut passer à la phase de test afin de vérifier que tout à bien été pris en compte.

→ On commence par vérifier directement sur le node que le service ***« apache2 »*** est bien démarré :

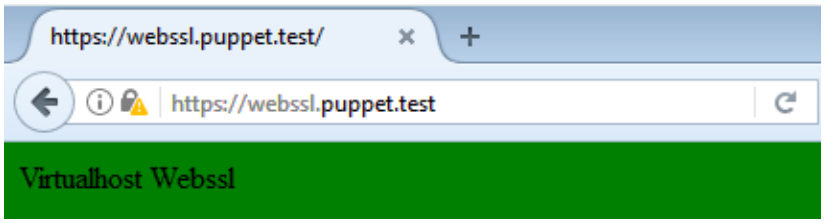
```
root@PuppetApache:~# service apache2 status
● apache2.service - LSB: Apache2 web server
   Loaded: loaded (/etc/init.d/apache2)
  Drop-In: /lib/systemd/system/apache2.service.d
           └─forking.conf
   Active: active (running) since jeu. 2017-07-27 05:18:43 CEST; 1s ago
```

Ensuite on va utiliser un navigateur web graphique via la machine ***« puppetgraph.puppet.test »*** pour tester les deux Virtualhosts :

→ ***« web.puppet.test »*** fonctionne bien correctement :

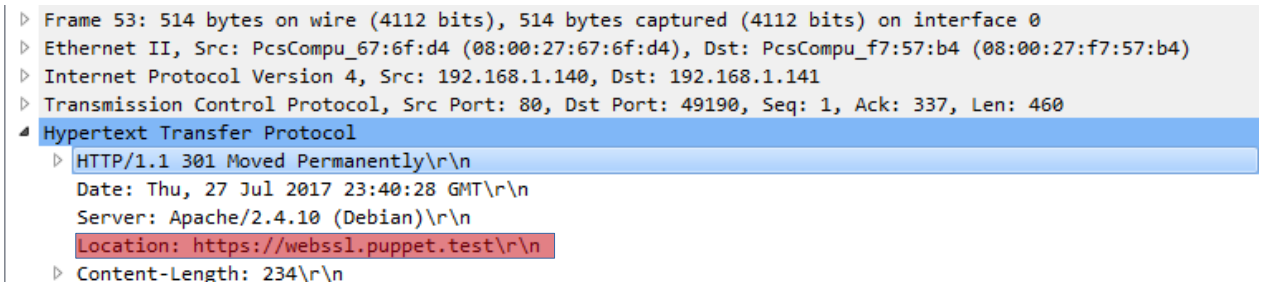


→ ***« webssl.puppet.test »*** fonctionne à priori également correctement :



Enfin il faut tester la redirection HTTPS du site ***« webssl.puppet.test »***, pour cela j'utilise le logiciel d'analyse de paquet WireShark afin de trouver le paquet HTTP de redirection.

→ Je lance donc une analyse de paquets pendant que je tente d'accéder au site ***« http://webssl.puppet.test »*** :



Ci-dessus le paquet en question capturé où l'on peut voir qu'il s'agit bien d'une requête HTTP de redirection vers le site ***« https://webssl.puppet.test ».****(en rouge)*

7. Déploiement de Wordpress :

Lors de cette partie je vais déployer un deuxième service ***« apache2 »*** sur le node ***« puppetapachewp.puppet.test »*** sur lequel sera installé le CMS Wordpress. Cela inclut également un service de base de données ***« mysql-server »*** sur le node ***« puppetmysql.puppet.test »***.

Ce déploiement sera donc déclaré sur deux manifests différents à savoir ***« /etc/puppet/manifests/mysql.pp »*** pour le service MySQL et ***« /etc/puppet/manifests/apacheWP.pp »*** pour le service Web et Wordpress.

7.1. MySQL :

Pour commencer, je vais m'occuper de la partie MySQL.

→ Pour cela il faut donc installer le module correspondant à savoir ***« puppetlabs-mysql »*** :

```
root@PuppetMaster:~# puppet module install puppetlabs-mysql
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└─┬ puppetlabs-mysql (v3.11.0)
  ├── puppet-staging (v2.2.0)
  └── puppetlabs-stdlib (v4.17.0)
```

→ Ensuite il faut éditer le manifest ***« /etc/puppet/manifests/mysql.pp »*** :

En rouge on installe le service ***« mysql-server »*** en définissant le mot de passe root, et en changeant une option du fichier de configuration par défaut permettant ainsi l'accès au serveur de base de données à distance. Cela va permettre au serveur Web sur lequel Wordpress sera installé d'y accéder.

Ensuite on crée la base de données ***« Wordpress »*** avec un utilisateur disposant de tous les droits dessus.
Cette utilisateur sera utilisé par le serveur web.*(En bleu)*

```
#Manifest mysql
class mysql {
#Installation du service MySQL
        class { '::mysql::server' :
                root_password   => 'root',
                override_options        => {
                mysqld  => {bind-address => '0.0.0.0'}
                }
        }
#Création de la BDD "wordpress"
        mysql::db{ 'wordpress':
                user    => 'wordpress',
                password        => 'wordpress',
                host    => '%',
                grant   => ['ALL'],
        }
#Installation d'un client MySQL
        class { '::mysql::client' : }
}
```

Enfin on installe un client MySQL dans le but d'effectuer des tests.*(en vert)*

C'est tout pour le déploiement du service de base de données, on va maintenant passer au service web ***« Apache2 »*** sur lequel Wordpress sera donc installé.

7.2. Apache2 :

Il est maintenant temps de traiter le service web ***« Apache2 »*** afin de pouvoir déployer Wordpress. Le manifest à éditer est donc ***« /etc/puppet/manifests/apacheWP.pp »***. A noter qu'il est impératif d'activer le module Apache2 php5 et de télécharger le paquet ***« php5-mysql »*** pour que Wordpress fonctionne.

Bien entendu le module ***« puppetlabs-apache »*** doit être installé comme vu lors de la partie 6.1.

Dans ce manifest il faut donc déclarer une nouvelle classe dans laquelle sera configurés les paramètres du service web ***« Apache2 »*** ainsi que les modalités de déploiement de Wordpress.

→ La classe gérant tout ceci est donc nommée ***« apacheWP »*** :

```
#Manifest ApacheWP
class apacheWP{

}
```

→ Pour la partie concernant ***« Apache2 »*** plusieurs choses sont à noter :

```
#Installation d'apache2
        class{ 'apache':
        default_vhost   => false,
        default_mods    => false,
        mpm_module      => 'prefork',
        }
#Activation php
        include apache::mod::php
#Virtualhost Wordpress
        apache::vhost { 'vhostweb':
                servername      => 'puppetapachewp.puppet.test',
                port    => '80',
                docroot => '/var/www/wordpress',
        }
#Declaration de la commande apt-get update
        exec { 'apt-update':
                command => '/usr/bin/apt-get update'
        }
#Installation du paquet php5-mysql
        package { 'php5-mysql':
                require => Exec['apt-update'],
                ensure  => installed,
        }
```

On fait donc appelle à la classe de référence ***« apache »*** qui va permettre l'installation du service, dans laquelle on déclare quelques attributs. ***« default_vhost »*** et **« default_mods »** vont permettre de désactiver respectivement le Virtualhost par défaut et les modules apaches. Le fait de désactiver les modules vont permettre d'activer le ***« Multi Processing Module »*** ***« prefork »****(en bleu)* indispensable pour le module ***« php »*** inclut par la suite.*(en rouge)*

Ensuite on déclare un nouveau Virtualhost comme vu lors de la partie 6.1.

Pour finir il va falloir installer le paquet ***« php-mysql »***, pour cela on déclare la commande ***« apt-get update »****(en violet)* puis le paquet à installer.*(en jaune)*

7.3. Wordpress :

Cette partie va servir à définir les modalités de déploiement du CMS Wordpress qui seront ajoutées à la classe ***« apacheWP »*** créée précédemment.

→ On commence par installer le module Wordpress nommé ***« hunner-wordpress »*** comme ci-dessous : *(en rouge)*

```
root@PuppetMaster:/etc/puppet/manifests# puppet module install hunner-wordpress
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└─┬ hunner-wordpress (v1.0.0)
  ├── puppetlabs-concat (v2.2.1)
  ├── puppetlabs-mysql (v3.11.0)
  └── puppetlabs-stdlib (v4.17.1)
```

→ Puis il faut éditer le manifest ***« /etc/puppet/manifests/apacheWP.pp »*** afin d'ajouter la partie ci-dessous dans la classe ***« apacheWP »*** :

```
#Module Worpress
        class { 'wordpress':
                db_user => 'wordpress',
                db_password     => 'wordpress',
                db_host => 'puppetmysql.puppet.test',
                create_db       => false,
                create_db_user  => false,
                install_dir     => '/var/www/wordpress',
        }
```

Alors pour Wordpress on appelle une nouvelle classe de référence nommée ***« wordpress »*** avec les attributs suivants :

- ***db_user :*** Utilisateur de la base de données ayant les droits sur la base de données ***« Wordpress »***.
- ***db_password :*** Mot de passe de cet utilisateur.
- ***db_host :*** Machine sur laquelle le service de base de données est installé.
- ***create_db :*** Permet de désactiver la création d'une nouvelle base de données, en effet on a déjà fait le nécessaire pour en créer une.
- ***create_db_user :*** Permet de désactiver la création d'un nouvel utilisateur sur le système de base de données.
- ***install_dir :*** Répertoire d'installation de Wordpress sur le node, donc le répertoire de publication indiqué dans la partie concernant Apache2.

C'est tout pour Wordpress, il va maintenant falloir faire remonter les manifests sur les nodes dans le bon ordre, à savoir le service de base de données ***« mysql-server »*** puis le service web ***« Apache2 »***.

7.4. Test :

Une fois que les manifests ont été récupérés sur les nodes, il va falloir vérifier que le déploiement fonctionne correctement.

→ On commence donc par voir si les services sont bien démarrés :

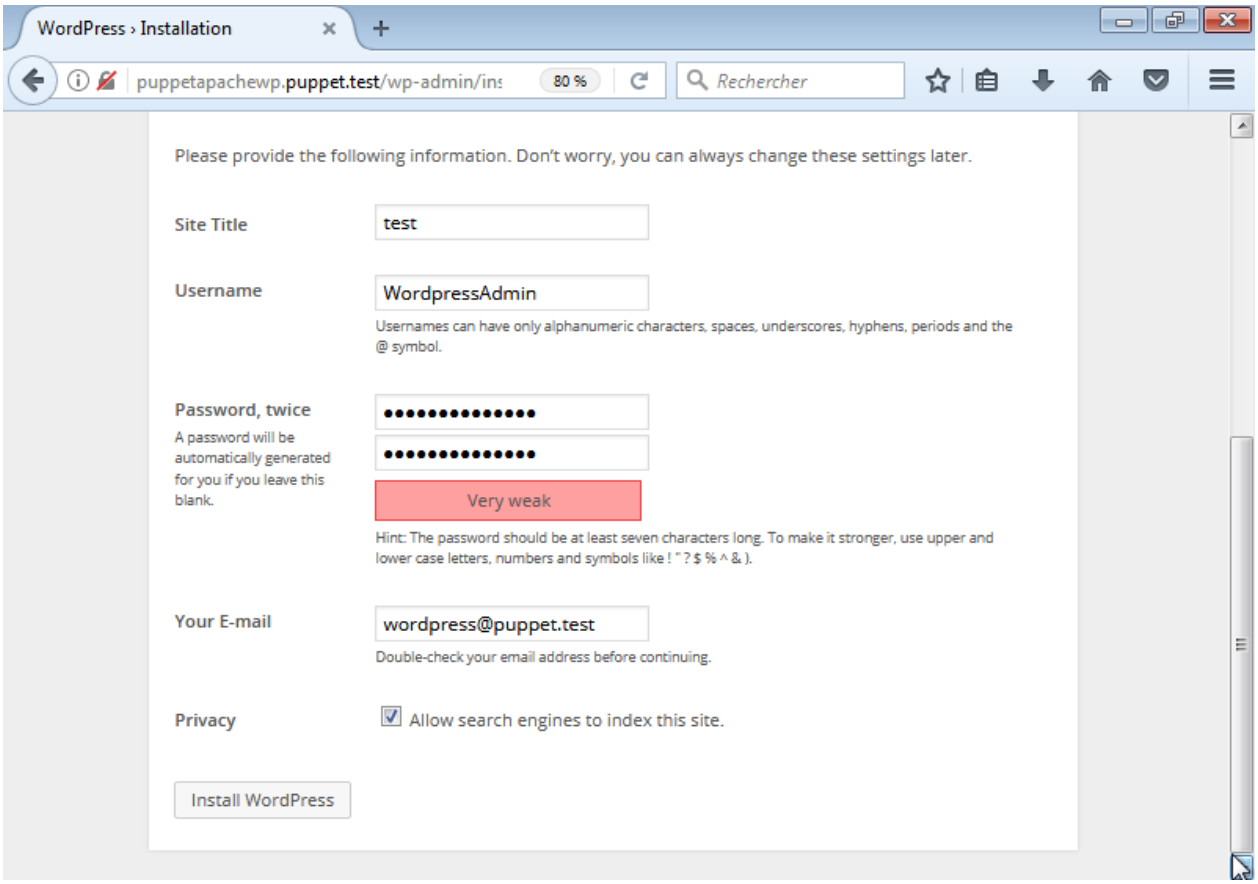
```
root@puppetmysql:~# service mysql status
● mysql.service - LSB: Start and stop the mysql database server daemon
   Loaded: loaded (/etc/init.d/mysql)
   Active: active (running) since sam. 2017-07-29 18:57:28 CEST; 4min 1s ago
   CGroup: /system.slice/mysql.service
           ├─4467 /bin/sh /usr/bin/mysqld_safe
           └─4859 /usr/sbin/mysqld --basedir=/usr --datadir=/var/lib/mysql --
```

```
root@puppetapachewp:~# service apache2 status | less
● apache2.service - LSB: Apache2 web server
   Loaded: loaded (/etc/init.d/apache2)
  Drop-In: /lib/systemd/system/apache2.service.d
           └─forking.conf
   Active: active (running) since sam. 2017-07-29 19:03:16 CEST; 12s ago
  Process: 6269 ExecStop=/etc/init.d/apache2 stop (code=exited, status=0/SUCCESS
)
  Process: 6291 ExecStart=/etc/init.d/apache2 start (code=exited, status=0/SUCCE
SS)
```

→ On en profite pour vérifier la présence de ***« php5 »*** avec la commande ***« php --version »*** :

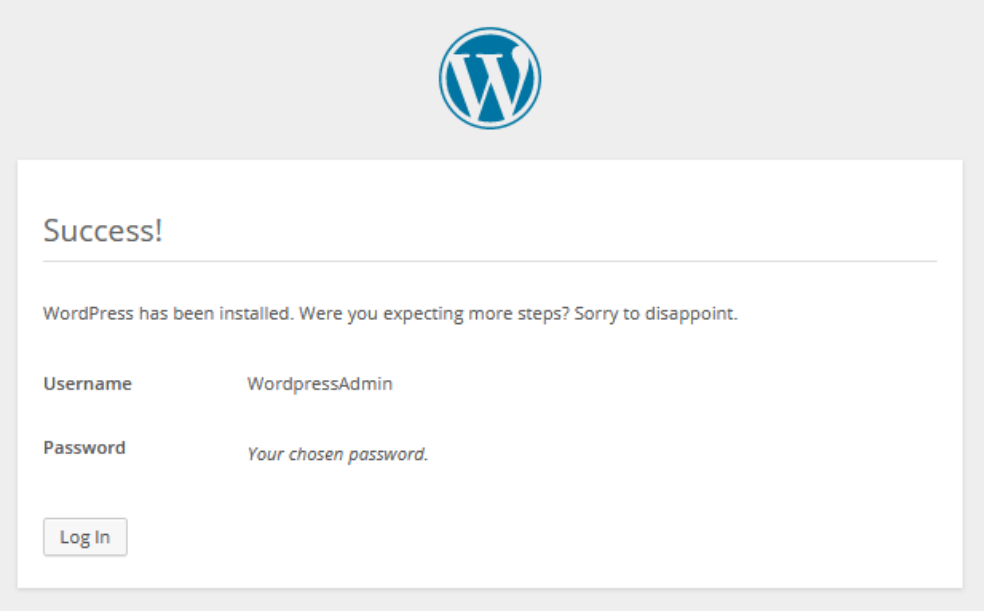
```
root@puppetapachewp:~# php --version
PHP 5.6.30-0+deb8u1 (cli) (built: Feb  8 2017 08:50:21)
Copyright (c) 1997-2016 The PHP Group
Zend Engine v2.6.0, Copyright (c) 1998-2016 Zend Technologies
    with Zend OPcache v7.0.6-dev, Copyright (c) 1999-2016, by Zend Technologies
```

→ Puis via le navigateur web graphique de la machine ***« puppetgraph.puppet.test »*** on va tester la bon fonctionnement de Wordpress en essayant d'accéder à l'URL ***« http://puppetapachewp.puppet.test »*** :



On peut voir qu'on est bien redirigé vers le script d'installation. Il faut donc remplir le formulaire puis cliquer sur « Install Wordpress ».

→ Enfin si tout ce passe correctement la page ci-dessus devrait s'afficher :



Cela implique donc le bon fonctionnement du module ***« php5 »*** ainsi que la présence du paquet ***« php5-mysql »***.

→ Pour finir je vais me connecter directement à la base de données pour vérifier que la base de données ***« wordpress »*** à été utilisée :

```
mysql> show tables;
+-----------------------+
| Tables_in_wordpress   |
+-----------------------+
| wp_commentmeta        |
| wp_comments           |
| wp_links              |
| wp_options            |
| wp_postmeta           |
| wp_posts              |
| wp_term_relationships |
| wp_term_taxonomy      |
| wp_terms              |
| wp_usermeta           |
| wp_users              |
+-----------------------+
11 rows in set (0.00 sec)
```

C'est tout pour le déploiement complet du CMS Wordpress, cependant cela ne suffit pas en milieu de production.

En effet, il peut être intelligent de coupler cela à un script de sauvegarde de la base de données afin de re-deployer une sauvegarde de la base de données Wordpress une fois le CMS restauré en cas de problème, pour de récupérer un site totalement fonctionnel.

8. Déploiement d'un service DHCP :

La dernière partie de ce tutoriel va concerner le déploiement du service DHCP ***« isc-dhcp-server »*** toujours via Puppet.

8.1. Isc-dhcp-server :

→ Comme tous les modules vus précédemment, on commence par installer le module Puppet adéquate, ici j'utilise ***« theforeman-dhcp »*** :

```
root@PuppetMaster:/etc/puppet/manifests# puppet module install theforeman-dhcp
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└─┬ theforeman-dhcp (v4.0.0)
  └── puppetlabs-concat (v2.2.1)
```

→ Ensuite on passe à l'édition du manifest dans lequel sera déclaré le déploiement du service DHCP ***« isc-dhcp-server »*** :

```
#Manifest DHCP
class dhcpserv {
#Installation du paquet isc-dhcp-server
        class { 'dhcp':
                dnsdomain       => [ 'puppet.test' ],
                interfaces      => [ 'eth0' ],
        }
#Declaration d'un pool
        dhcp::pool { 'pool1.puppet.test':
                network => '192.168.1.0',
                mask    => '255.255.255.0',
                range   => '192.168.1.100 192.168.1.110',
                gateway => '192.168.1.1'
        }
}
```

Donc la classe ***« dhcpserv »*** est créée, dans laquelle on va déclarer la classe de référence ***« dhcp »*** permettant l'installation du service. Via les attributs ***« dnsdomain »*** et ***« interfaces »*** on indique le nom de domaine et l'interface réseau utilisée. En rouge on déclare le pool d'adresses nommé ***« pool1.puppet.test »*** nous permettant ainsi d'établir :

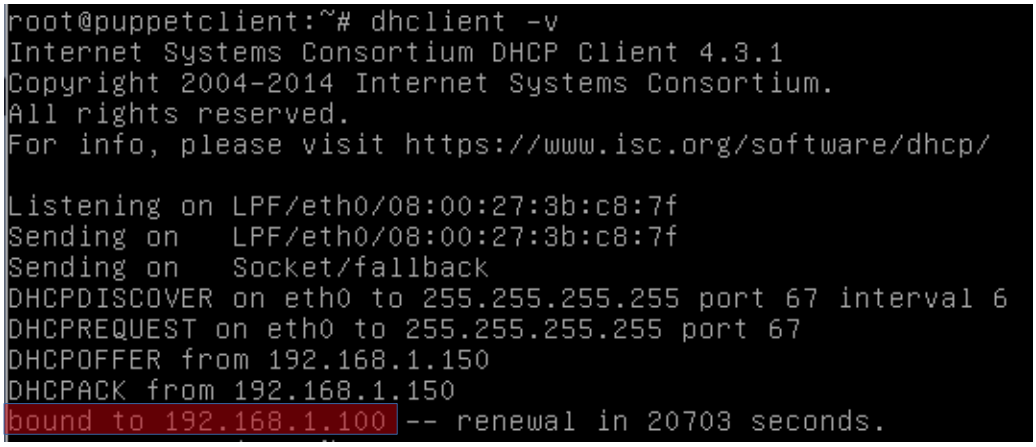
- ***network :*** Adresse IP du réseau.
- ***mask :*** Le masque IP.
- ***range :*** les adresses IP pouvant être allouées aux clients DHCP.
- ***gateway :*** Adresse IP de la passerelle du réseau donc du routeur.

C'est tout pour ce manifest, il faut maintenant le faire remonter sur le node approprié.

8.2. Test :

Maintenant que le service ***« isc-dhcp-server »*** est déployé et configuré, il faut tester son bon fonctionnement.

→ Pour cela j'ai configuré le node ***« puppetclient.puppet.test »*** en DHCP et j'ai lancé la demande de configuration IP :



Donc la commande ***« dhclient -v »*** permet de solliciter un serveur DHCP sur le réseau. On peut voir que le node ***« puppetdhcp.puppet.test »*** nous a bien délivré une configuration IP.*(en rouge)*