Version : **2022.01**

Dernière mise-à-jour : 2023/09/13 14:09

LDF801 - Puppet en Mode Sans Maître

Contenu du Module

Qu'est-ce Puppet ?

Puppet est :

- un logiciel libre écrit partiellement en Ruby permettant la gestion de la configuration de serveurs esclaves (GNU/Linux, Mac OS X et Windows),
- diffusé sous licence Apache 2.0 pour les versions récentes de Puppet.

La version libre :

- permet de gérer les déploiements système et applicatif,
- accepte les machines virtuelles de type Amazon EC2.

La version commerciale de Puppet permet aussi :

- de gérer les machines virtuelles VMware,
- d'avoir une interface graphique de gestion,
- d'automatiser et d'orchestrer les déploiements,
- d'avoir une plateforme de développement pour tous les environnements,
- de gérer individuellement les droits utilisateurs.

Démarrer avec Puppet

Puppet est un outil de gestion de la configuration de systèmes. Pour résumer le flux de travail avec Puppet, on peut dire que :

- vous spécifiez l'état de configuration voulu en éditant des fichier textes et un modèle des ressources du système,
- Puppet compare l'état actuel avec l'état voulu et procède aux modifications nécessaires.

Plus particulièrement Puppet est :

- un langage pour spécifier l'état désiré,
- un moteur qui interprète le code écrit dans ce langage et qui l'applique aux nœuds pour arriver à l'état désiré.

Par exemple :

```
package { 'curl':
 ensure => installed,
}
```

Ce **manifest** indique que le paquet **curl** doit être installé. Quand ce code est appliqué, Puppet vérifie si le paquet **curl** est installé dans le nœud puis :

- si c'est le cas, ne fait rien,
- sinon l'installe.

Voici un autre exemple :

```
user { 'trainee':
 ensure => present,
}
```

Dans ce cas, l'utilisateur doit être présent dans le nœud concerné, sinon Puppet va créer l'utilisateur.

La force de Puppet est qu'il est capable d'implémenter ce code quelque soit le système d'exploitation du nœud, parmi les systèmes d'exploitation compatibles :

- vous décrivez la configuration désirée en termes de ressources et d'attributs,
- Puppet crée les ressources dans les différents systèmes d'exploitation en utilisant le même **manifest**.

Il existe deux façons d'utiliser Puppet :

- Stand-alone Puppet ou “Sans Maître”,
  - Puppet est exécuté sur **chaque** nœud et n'a pas donc besoin de contacter un Maître. Cette configuration utilise Git, SFTP ou rsync pour mettre à jour les manifests sur chaque nœud,
- Architecture Agent/Maître,
  - un nœud est dédié à l'exécution de Puppet et tous les autres nœuds doivent le contacter pour connaître la configuration à appliquer.

> **Important** - Ce cours commence avec l'utilisation de Puppet en mode **Sans Maître** en utilisant Git et termine avec l'étude de l'architecture Agent/Maître.

Utiliser des Manifests

LAB #1 - Gestion des Fichiers

Connectez-vous à votre machine virtuelle et démarrez la VM Puppet :

```
trainee@debian10:~$ cd puppet-beginners-guide-3/

trainee@debian10:~/puppet-beginners-guide-3$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'ubuntu/xenial64' version '20210422.0.0' is up to date...
==> default: A newer version of the box 'ubuntu/xenial64' for provider 'virtualbox' is
==> default: available! You currently have version '20210422.0.0'. The latest is version
==> default: '20211001.0.0'. Run `vagrant box update` to update.
...
trainee@debian10:~/puppet-beginners-guide-3$ vagrant ssh
Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.4.0-210-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable

New release '18.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Tue Apr 27 17:02:35 2021 from 10.0.2.2
```

Mettez à jour les dépôts de paquets :

```
vagrant@ubuntu-xenial:~$ sudo dpkg --configure -a

vagrant@ubuntu-xenial:~$ sudo apt update
Hit:1 http://archive.ubuntu.com/ubuntu xenial InRelease
Hit:2 http://archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:3 http://archive.ubuntu.com/ubuntu xenial-backports InRelease
Hit:4 http://apt.puppetlabs.com xenial InRelease
Hit:5 http://security.ubuntu.com/ubuntu xenial-security InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
37 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Créez ensuite le fichier **file_hello.pp**

```
vagrant@ubuntu-xenial:~$ vi file_hello.pp

vagrant@ubuntu-xenial:~$ cat file_hello.pp
file { '/tmp/hello.txt':
  ensure  => file,
  content => "hello, world\n",
}
```

Dans ce fichier nous pouvons constater la déclaration d'une ressource de type fichier avec **file**. Une ressource est une configuration que vous souhaitez être gérée par Puppet par exemple, un fichier, un utilisateur, un compte ou bien un paquet. Le format de notre manifest est donc :

```
TYPE_RESSOURCE { TITRE:
ATTRIBUT => VALEUR,
...
}
```

La ressource est identifiée par un titre. Chaque ressource doit avoir un titre unique. Dans le cas d'un fichier c'est le chemin complet vers le fichier -

/tmp/hello.txt.

Le reste du code est une liste d'attributs pour la ressource. Dans le cas d'un fichier les attributs disponibles pour sont, par exemple :

- le contenu,
- le propriétaire,
- le groupe,
- le mode,
- etc ...

> **Important** - Bien que les attributs soient différents selon le type de ressource, l'attribut **ensure** est commun à toutes les ressources. Par contre, la valeur de cet attribut diffère selon le type de ressource. Dans le cas de notre code, il est stipulé un fichier et non un répertoire ou un liens symbolique.

Le contenu de ce fichier est stipulé par l'attribut **content** qui est ici une chaîne **hello, world** suivie par un caractère de nouvelle ligne **\n**.

L'application du manifest par Puppet se résume ainsi :

- Puppet lit le manifest ainsi que la liste des ressources,
- Puppet compile les ressources en un **catalogue**,
- Puppet lit le catalogue et applique chaque ressource à tour de rôle.

Le nom du manifest n'est pas important, par contre l'extension doit être **.pp**

Appliquez ce fichier avec la commande suivante :

```
vagrant@ubuntu-xenial:~$ sudo puppet apply file_hello.pp
Notice: Compiled catalog for ubuntu-xenial in environment production in 0.06 seconds
Notice: /Stage[main]/Main/File[/tmp/hello.txt]/ensure: defined content as '{md5}22c3683b094136c3398391ae71b20f04'
Notice: Applied catalog in 0.06 seconds
```

Vérifiez que Puppet a écrit le fichier **/tmp/hello.txt** :

```
vagrant@ubuntu-xenial:~$ cat /tmp/hello.txt
hello, world
```

1.1 - Modification d'un Fichier Existant sur le Serveur

Dans le cas où le fichier existe déjà et son contenu est différent, Puppet écrasera son contenu avec celui du manifest :

```
vagrant@ubuntu-xenial:~$ sudo sh -c 'echo "goodbye, world" > /tmp/hello.txt'
vagrant@ubuntu-xenial:~$ cat /tmp/hello.txt
goodbye, world
vagrant@ubuntu-xenial:~$ sudo puppet apply file_hello.pp
Notice: Compiled catalog for ubuntu-xenial in environment production in 0.06 seconds
Notice: /Stage[main]/Main/File[/tmp/hello.txt]/content: content changed '{md5}767887814e925822027f4fe63fb69ce2'
to '{md5}22c3683b094136c3398391ae71b20f04'
Notice: Applied catalog in 0.11 seconds
vagrant@ubuntu-xenial:~$ cat /tmp/hello.txt
hello, world
```

> **Important** - Des modifications manuelles faites donc à un fichier géré par Puppet seront perdues la prochaine fois que Puppet s'exécute, sauf dans le cas où le manifest reflète les mêmes modifications. Pour cette raison, il est une bonne pratique d'insérer un commentaire en début de fichier comme avertissement.

1.2 - Effectuer un Dry Run avec Puppet

Il est possible de demander à Puppet, grâce à l'utilisation de l'option **–noop**, de nous informer des modifications qu'il aurait fait en appliquant un manifest, sans que ces modifications soient réellement effectuées :

```
vagrant@ubuntu-xenial:~$ sudo sh -c 'echo "goodbye, world" > /tmp/hello.txt'
vagrant@ubuntu-xenial:~$ sudo puppet apply --noop file_hello.pp
Notice: Compiled catalog for ubuntu-xenial in environment production in 0.07 seconds
Notice: /Stage[main]/Main/File[/tmp/hello.txt]/content: current_value '{md5}767887814e925822027f4fe63fb69ce2',
should be '{md5}22c3683b094136c3398391ae71b20f04' (noop)
Notice: Class[Main]: Would have triggered 'refresh' from 1 event
Notice: Stage[main]: Would have triggered 'refresh' from 1 event
Notice: Applied catalog in 0.07 seconds
vagrant@ubuntu-xenial:~$ cat /tmp/hello.txt
goodbye, world
```

Comme vous pouvez constater, Puppet décide si un fichier doit être modifié en fonction de la valeur du hash md5.

Pour constater les modifications que Puppet aurait effectué, utilisez l'option **–show_diff** :

```
vagrant@ubuntu-xenial:~$ sudo puppet apply --noop --show_diff file_hello.pp
Notice: Compiled catalog for ubuntu-xenial in environment production in 0.07 seconds
Notice: /Stage[main]/Main/File[/tmp/hello.txt]/content:
--- /tmp/hello.txt  2018-05-26 13:31:30.480333595 +0000
+++ /tmp/puppet-file20180526-27193-1pjhkk9  2018-05-26 13:36:45.039195308 +0000
@@ -1 +1 @@
-goodbye, world
+hello, world

Notice: /Stage[main]/Main/File[/tmp/hello.txt]/content: current_value '{md5}767887814e925822027f4fe63fb69ce2',
should be '{md5}22c3683b094136c3398391ae71b20f04' (noop)
Notice: Class[Main]: Would have triggered 'refresh' from 1 event
Notice: Stage[main]: Would have triggered 'refresh' from 1 event
Notice: Applied catalog in 0.11 seconds
```

LAB #2 - Gestion des Paquets

Puppet est capable de gérer les paquets grâce à la ressource **package**. Dans le manifest nous trouvons donc cette ressource, le nom du paquet ainsi que l'attribut **ensure** :

```
package { 'package_name':
ensure => installed,
}
```

Ce manifest aura comme résultat d'assurer que le package **package_name** soit installé en utilisant le gestionnaire des paquets du système d'exploitation.

Créez le fichier **package.pp** suivant :

```
vagrant@ubuntu-xenial:~$ sudo vi package.pp
vagrant@ubuntu-xenial:~$ cat package.pp
package { 'cowsay':
  ensure => installed,
}
```

Important - Le titre de la ressource de type paquet est **cowsay**.

Appliquez le manifest :

```
vagrant@ubuntu-xenial:~$ sudo puppet apply package.pp
Notice: Compiled catalog for ubuntu-xenial in environment production in 1.51 seconds
Notice: /Stage[main]/Main/Package[cowsay]/ensure: created
Notice: Applied catalog in 8.76 seconds
```

Le résultat de l'application de ce manifest est l'installation du paquet **cowsay** :

```
vagrant@ubuntu-xenial:~$ dpkg --get-selections | grep cowsay
cowsay                      install
cowsay-off                  install
vagrant@ubuntu-xenial:~$ dpkg -s cowsay
Package: cowsay
Status: install ok installed
Priority: optional
Section: games
Installed-Size: 90
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Architecture: all
Version: 3.03+dfsg1-15
Depends: libtext-charwidth-perl, perl
Recommends: cowsay-off
Suggests: filters
Description: configurable talking cow
 Cowsay (or cowthink) will turn text into happy ASCII cows, with
 speech (or thought) balloons. If you don't like cows, ASCII art is
 available to replace it with some other creatures (Tux, the BSD
 daemon, dragons, and a plethora of animals, from a turkey to
 an elephant in a snake).
Original-Maintainer: Tony Maillefaud <maltouzes@gmail.com>
Homepage: https://web.archive.org/web/20120527202447/http://www.nog.net/~tony/warez/cowsay.shtml
```

Pour voir la version du paquet que Puppet pense être installé, utilisez la commande **puppet resource** en spécifiant le paquet concerné :

```
vagrant@ubuntu-xenial:~$ sudo puppet resource package openssl
package { 'openssl':
  ensure   => '1.0.2g-1ubuntu4.15',
  provider => 'apt',
}
vagrant@ubuntu-xenial:~$ dpkg -s openssl
Package: openssl
Status: install ok installed
```

```
Priority: optional
Section: utils
Installed-Size: 934
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Architecture: amd64
Version: 1.0.2g-1ubuntu4.15
Depends: libc6 (>= 2.15), libssl1.0.0 (>= 1.0.2g)
Suggests: ca-certificates
Conffiles:
 /etc/ssl/openssl.cnf 7df26c55291b33344dc15e3935dabaf3
Description: Secure Sockets Layer toolkit - cryptographic utility
 This package is part of the OpenSSL project's implementation of the SSL
 and TLS cryptographic protocols for secure communication over the
 Internet.
 .
 It contains the general-purpose command line binary /usr/bin/openssl,
 useful for cryptographic operations such as:
  * creating RSA, DH, and DSA key parameters;
  * creating X.509 certificates, CSRs, and CRLs;
  * calculating message digests;
  * encrypting and decrypting with ciphers;
  * testing SSL/TLS clients and servers;
  * handling S/MIME signed or encrypted mail.
Original-Maintainer: Debian OpenSSL Team <pkg-openssl-devel@lists.alioth.debian.org>
```

L'utilisation de cette commande sans spécifier un paquet permet de voir la liste de tous les paquets :

```
vagrant@ubuntu-xenial:~$ sudo puppet resource package | more
package { 'accountsservice':
  ensure   => '0.6.40-2ubuntu11.3',
  provider => 'apt',
}
package { 'acl':
  ensure   => '2.2.52-3',
```

```
  provider => 'apt',
}
package { 'acpid':
  ensure   => '1:2.0.26-1ubuntu2',
  provider => 'apt',
}
package { 'adduser':
  ensure   => '3.113+nmu3ubuntu4',
  provider => 'apt',
}
package { 'apparmor':
  ensure   => '2.10.95-0ubuntu2.11',
  provider => 'apt',
}
package { 'apport':
  ensure   => '2.20.1-0ubuntu2.21',
  provider => 'apt',
--More--
```

Puppet resource a aussi un mode interactif :

```
vagrant@ubuntu-xenial:~$ sudo puppet resource -e package openssl
```

Cette commande génère un manifest pour l'état actuel de la ressource et l'ouvre dans un éditeur :

```
package { 'openssl':
  ensure   => '1.0.2g-1ubuntu4.15',
  provider => 'apt',
}
```

Si vous modifiez ce manifest, lors de l'enregistrement du fichier, Puppet appliquera les modifications.

LAB #3 - Gestion des Services

Le manifest pour la gestion d'un service contient la ressource **service**. Créez le fichier suivant :

```
vagrant@ubuntu-xenial:~$ sudo vi service.pp
vagrant@ubuntu-xenial:~$ cat service.pp
service { 'sshd':
  ensure => running,
  enable => true,
}
```

Les attributs d'une ressource peuvent être consultés par la commande **puppet describe** :

```
vagrant@ubuntu-xenial:~$ sudo puppet describe service

service
=======
Manage running services.  Service support unfortunately varies
widely by platform --- some platforms have very little if any concept of a
running service, and some have a very codified and powerful concept.
Puppet's service support is usually capable of doing the right thing, but
the more information you can provide, the better behaviour you will get.
Puppet 2.7 and newer expect init scripts to have a working status command.
If this isn't the case for any of your services' init scripts, you will
need to set `hasstatus` to false and possibly specify a custom status
command in the `status` attribute. As a last resort, Puppet will attempt to
search the process table by calling whatever command is listed in the `ps`
fact. The default search pattern is the name of the service, but you can
specify it with the `pattern` attribute.
**Refresh:** `service` resources can respond to refresh events (via
`notify`, `subscribe`, or the `~>` arrow). If a `service` receives an
event from another resource, Puppet will restart the service it manages.
The actual command used to restart the service depends on the platform and
```

```
can be configured:
* If you set `hasrestart` to true, Puppet will use the init script's restart
command.
* You can provide an explicit command for restarting with the `restart`
attribute.
* If you do neither, the service's stop and start commands will be used.


Parameters
----------

- **binary**
    The path to the daemon.  This is only used for
    systems that do not support init scripts.  This binary will be
    used to start the service if no `start` parameter is
provided.

- **control**
    The control variable used to manage services (originally for HP-UX).
    Defaults to the upcased service name plus `START` replacing dots with
    underscores, for those providers that support the `controllable`
    feature.

- **enable**
    Whether a service should be enabled to start at boot.
    This property behaves differently depending on the platform;
    wherever possible, it relies on local tools to enable or disable
    a given service. Default values depend on the platform.
    Valid values are `true`, `false`, `manual`, `mask`, `delayed`.
    Requires features enableable.

- **ensure**
    Whether a service should be running. Default values depend on the
    platform.
```

```
    Valid values are `stopped` (also called `false`), `running` (also called
    `true`).

- **flags**
    Specify a string of flags to pass to the startup script.
    Requires features flaggable.

- **hasrestart**
    Specify that an init script has a `restart` command.  If this is
    false and you do not specify a command in the `restart` attribute,
    the init script's `stop` and `start` commands will be used.
    Valid values are `true`, `false`.

- **hasstatus**
    Declare whether the service's init script has a functional status
    command. This attribute's default value changed in Puppet 2.7.0.
    The init script's status command must return 0 if the service is
    running and a nonzero value otherwise. Ideally, these exit codes
    should conform to [the LSB's specification][lsb-exit-codes] for init
    script status actions, but Puppet only considers the difference
    between 0 and nonzero to be relevant.
    If a service's init script does not support any kind of status command,
    you should set `hasstatus` to false and either provide a specific
    command using the `status` attribute or expect that Puppet will look for
    the service name in the process table. Be aware that 'virtual' init
    scripts (like 'network' under Red Hat systems) will respond poorly to
    refresh events from other resources if you override the default behavior
    without providing a status command.
Valid values are `true`, `false`.

- **manifest**
    Specify a command to config a service, or a path to a manifest to do so.

- **name**
```

```
    The name of the service to run.
    This name is used to find the service; on platforms where services
    have short system names and long display names, this should be the
    short name. (To take an example from Windows, you would use "wuauserv"
    rather than "Automatic Updates.")

- **path**
    The search path for finding init scripts.  Multiple values should
    be separated by colons or provided as an array.

- **pattern**
    The pattern to search for in the process table.
    This is used for stopping services on platforms that do not
    support init scripts, and is also used for determining service
    status on those service whose init scripts do not include a status
    command.
    Defaults to the name of the service. The pattern can be a simple string
    or any legal Ruby pattern, including regular expressions (which should
    be quoted without enclosing slashes).

- **restart**
    Specify a *restart* command manually.  If left
    unspecified, the service will be stopped and then started.

- **start**
    Specify a *start* command manually.  Most service subsystems
    support a `start` command, so this will not need to be
specified.

- **status**
    Specify a *status* command manually.  This command must
    return 0 if the service is running and a nonzero value otherwise.
    Ideally, these exit codes should conform to [the LSB's
    specification][lsb-exit-codes] for init script status actions, but
```

```
    Puppet only considers the difference between 0 and nonzero to be
    relevant.
    If left unspecified, the status of the service will be determined
    automatically, usually by looking for the service in the process
    table.
    [lsb-exit-codes]:
    http://refspecs.linuxfoundation.org/LSB_4.1.0/LSB-Core-generic/LSB-Core-
    generic/iniscrptact.html

- **stop**
    Specify a *stop* command manually.

- **timeout**
    Specify an optional minimum timeout (in seconds) for puppet to wait when
    syncing service properties
Requires features configurable_timeout.

Providers
---------
    base, bsd, daemontools, debian, freebsd, gentoo, init, launchd, openbsd,
    openrc, openwrt, rcng, redhat, runit, service, smf, src, systemd,
    upstart, windows
```

Pour consulter la liste des types de ressources, utilisez la commande suivante :

```
vagrant@ubuntu-xenial:~$ sudo puppet describe --list
These are the types known to puppet:
augeas          - Apply a change or an array of changes to the  ...
cron            - Installs and manages cron jobs
exec            - Executes external commands
file            - Manages files, including their content, owner ...
filebucket      - A repository for storing and retrieving file  ...
group           - Manage groups
host            - Installs and manages host entries
```

```
mount           - Manages mounted filesystems, including puttin ...
notify          - Sends an arbitrary message to the agent run-t ...
package         - Manage packages
resources       - This is a metatype that can manage other reso ...
schedule        - Define schedules for Puppet
scheduled_task  - Installs and manages Windows Scheduled Tasks
selboolean      - Manages SELinux booleans on systems with SELi ...
selmodule       - Manages loading and unloading of SELinux poli ...
service         - Manage running services
ssh_authorized_key - Manages SSH authorized keys
sshkey          - Installs and manages ssh host keys
stage           - A resource type for creating new run stages
tidy            - Remove unwanted files based on specific crite ...
user            - Manage users
whit            - Whits are internal artifacts of Puppet's curr ...
yumrepo         - The client-side description of a yum reposito ...
zfs             - Manage zfs
zone            - Manages Solaris zones
zpool           - Manage zpools
```

Les différents types de ressources peuvent être regroupés dans le même manifest. Créez le fichier **package_file_service.pp** :

```
vagrant@ubuntu-xenial:~$ sudo vi package_file_service.pp
vagrant@ubuntu-xenial:~$ cat package_file_service.pp
package { 'mysql-server':
  ensure => installed,
  notify => Service['mysql'],
}

file { '/etc/mysql/mysql.cnf':
  source => '/examples/files/mysql.cnf',
  notify => Service['mysql'],
}
```

```
service { 'mysql':
  ensure => running,
  enable => true,
}
```

Notez que dans ce manifest se trouve l'attribut **notify**. Cet attribut notifie le service mysql lors d'un changement de son fichier de configuration **mysql.cnf**. L'action par défaut de Puppet dans ce cas est de re-démarrer le service.

L'ordre de la déclaration des ressources dans ce manifest est suivi par Puppet :

- Puppet install le paquet **mysql-server**,
- Puppet copie le fichier **/examples/files/mysql.cnf** vers **/etc/mysql/mysql.cnf**,
- Puppet démarre le service **mysql**.

Cet ordre est logique. Il est évident que le manifest suivant ne donnera pas un résultat satisfaisant :

```
package { 'mysql-server':
  ensure => installed,
  notify => Service['mysql'],
}

service { 'mysql':
  ensure => running,
  enable => true,
}

file { '/etc/mysql/mysql.cnf':
  source => '/examples/files/mysql.cnf',
  notify => Service['mysql'],
}
```

L'ordre de l'application des ressources dans un manifest peut cependant être fixé en utilisant l'attribut **require**. Voici le même manifest avec l'utilisation de cet attribut :

```
package { 'mysql-server':
  ensure => installed,
}

file { '/etc/mysql/mysql.cnf':
  source  => '/examples/files/mysql.cnf',
  notify  => Service['mysql'],
  require => Package['mysql-server'],
}

service { 'mysql':
  ensure  => running,
  enable  => true,
  require => [Package['mysql-server'], File['/etc/mysql/mysql.cnf']],
}
```

Gérer du code Puppet avec Git

Git est :

- un logiciel de **gestion de versions** décentralisé,
- un logiciel libre créé par Linus Torvalds,
- distribué selon les termes de la licence publique générale GNU version 2.

Quand le code d'un projet est modifié par un développeur, celui-ci procède à un **commit** pour rendre le code disponible aux autres. Un commit est un **snapshot** ou instantanée du repository qui est gardé pour toujours ce qui implique la possibilité de **rollbacks**.

Important - Pour apprendre comment écrire un message de commit Git, consultez ce lien : **https://chris.beams.io/posts/git-commit** (en anglais).

Important - Avant de poursuivre, créez-vous un compte sur le site https://github.com ainsi qu'un **Personal Access Token**. Pour savoir comment créer votre Personal Access Token, consultez ce lien : **https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens#creating-a-personal-access-token-classic** (en anglais).

LAB #4 - Créer un Repository Local

Un jeu de fichiers sous Git est appelé un **repository** ou simplement un **repo**. Pour créer un repo local pour puppet, utilisez les commandes suivantes :

```
vagrant@ubuntu-xenial:~$ sudo mkdir puppet
vagrant@ubuntu-xenial:~$ sudo chown vagrant:vagrant puppet
vagrant@ubuntu-xenial:~$ cd puppet
vagrant@ubuntu-xenial:~/puppet$ git init
Initialized empty Git repository in /home/vagrant/.git/
```

Git a besoin de savoir qui vous êtes. Saisissez donc les commandes suivantes en remplaçant les coordonnées d'identification avec les vôtres :

```
vagrant@ubuntu-xenial:~/puppet$ git config --global user.name "ittrainingdev"
vagrant@ubuntu-xenial:~/puppet$ git config --global user.email "infos@i2tch.co.uk"
```

Un repo doit contenir un fichier README qui contient des informations concernant le repo. Créez donc ce fichier avec un minimum d'informations :

```
vagrant@ubuntu-xenial:~/puppet$ echo "Coming soon!" > README.md
```

Consultez le statut du repo avec la commande **git status** :

```
vagrant@ubuntu-xenial:~/puppet$ git status
```

```
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README.md

nothing added to commit but untracked files present (use "git add" to track)
```

Pour informer Git de la présence du fichier **README.md**, il convient d'utiliser la commande **git add** :

```
vagrant@ubuntu-xenial:~/puppet$ sudo git add README.md
vagrant@ubuntu-xenial:~/puppet$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   README.md
```

Notez que le fichier est maintenant sous la ligne **Changes to be committed:**. Procédez donc à un commit :

```
vagrant@ubuntu-xenial:~/puppet$ sudo git commit -m 'Add README file'
[master (root-commit) 9139685] Add README file
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

Pour voir l'historique des commits, utilisez la commande **git log** :

```
vagrant@ubuntu-xenial:~/puppet$ git log
```

```
commit 913968526d9748f8d92e6eacea03ac0a6d1ac901
Author: ittrainingdev <infos@i2tch.co.uk>
Date:   Sun May 27 04:53:07 2018 +0000

    Add README file
```

4.1 - Les Branches avec Git

Une Branche sous Git est une copie parallèle du code où les modifications sont indépendantes. Ces modification peuvent ensuite être fusionner avec la Branche Maître. Ceci permet :

- de connecter un nœud à une branche spécifique pour tester du code, sans l'implémenter sur tous le nœuds,
- à deux ou plusieurs développeurs de travailler d'une manière indépendante puis d'échanger des commits entre les Branches.

Vous pouvez trouver d'avantage d'informations concernant les Branches de Git à cet URL **https://git-scm.com/book/fr/v1/Les-branches-avec-Git-Brancher-et-fusionner%C2%A0%3A-les-bases** ainsi que l'ebook gratuit ici - **https://git-scm.com/book/fr/v2**.

LAB #5 - Créer un Repository Distant

Commencez par vous connecter à votre compte sur **https://github.com**. Créez ensuite un nouveau repo **puppet** à l'adresse **https://github.com/new**.

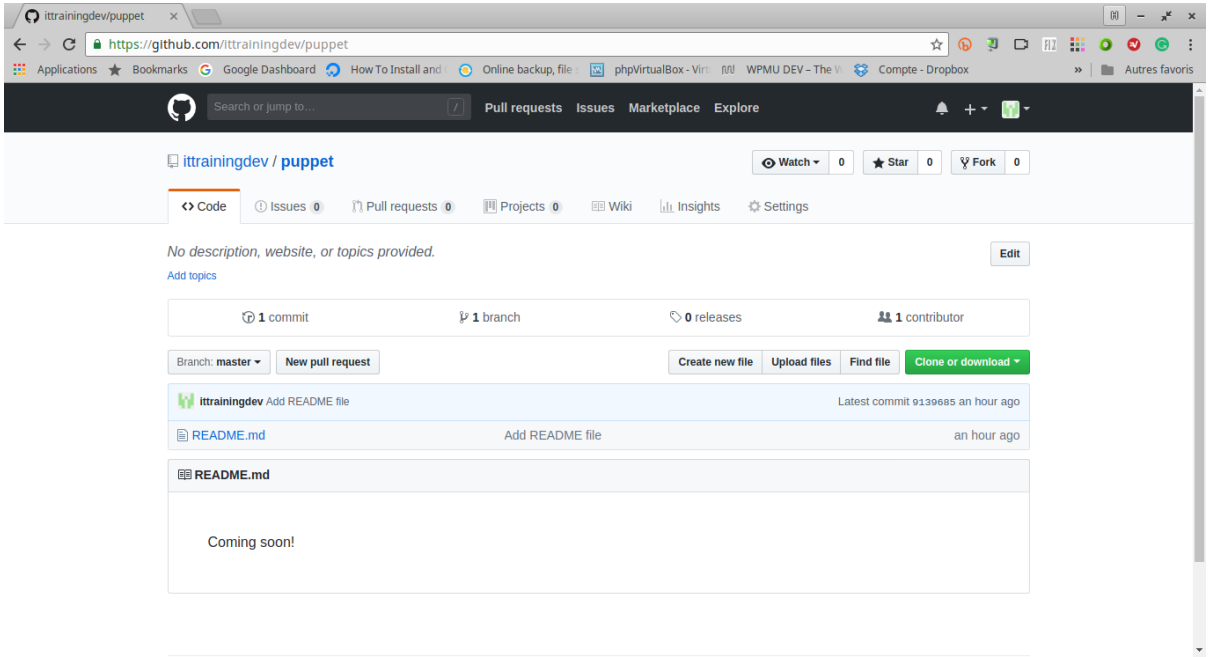
Copiez l'URL du repo, par exemple : **https://github.com/ittrainingdev/puppet.git**.

Vous devez maintenant pousser votre repo local vers Github. Placez-vous dans le répertoire puppet de votre machine virtuelle et saisissez la commande suivante en collant **à la place** de l'URL ci-dessous, l'URL que vous avez copié ci-dessus et **en utilisant vos coordonnées de connexion** :

```
vagrant@ubuntu-xenial:~/puppet$ sudo git remote add origin https://github.com/ittrainingdev/puppet.git
vagrant@ubuntu-xenial:~/puppet$ sudo git push -u origin master
Username for 'https://github.com': ittrainingdev
Password for 'https://ittrainingdev@github.com': collez votre personal-access-token-classic ici
```

```
Counting objects: 3, done.
Writing objects: 100% (3/3), 232 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/ittrainingdev/puppet.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

Vérifiez maintenant votre repo sur Github en visitant l'URL de votre repo. Vous devez voir le fichier README.md :



LAB #7 - Cloner un Repository

Le répertoire par défaut de stockage des manifests de Puppet est **/etc/puppetlabs/code/environments/**.

Placez-vous donc dans ce répertoire :

```
vagrant@ubuntu-xenial:~/puppet$ cd /etc/puppetlabs/code/environments/
```

Consultez son contenu :

```
vagrant@ubuntu-xenial:/etc/puppetlabs/code/environments$ ls
pbg  production
```

Le répertoire **production** contient des exemples de manifests de production installés par défaut :

```
agrant@ubuntu-xenial:/etc/puppetlabs/code/environments$ cd production/
vagrant@ubuntu-xenial:/etc/puppetlabs/code/environments/production$ ls
data  environment.conf  hiera.yaml  manifests  modules
```

Git refuse de cloner vers un répertoire non-vide. Renommez donc le répertoire production :

```
vagrant@ubuntu-xenial:/etc/puppetlabs/code/environments/production$ cd ..
vagrant@ubuntu-xenial:/etc/puppetlabs/code/environments$ sudo mv production production.sample
```

Cloner maintenant le repo sur Github vers un **nouveau** répertoire **/etc/puppetlabs/code/environments/production** :

```
vagrant@ubuntu-xenial:/etc/puppetlabs/code/environments$ sudo git clone
https://github.com/ittrainingdev/puppet.git production
Cloning into 'production'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
```

LAB #7 - Appliquer des Modifications Automatiquement

En production, chaque nœud doit automatiquement télécharger les modifications du code dans le **repo** distant puis les appliquer avec Puppet.

Pour arriver à cette fin, il convient de créer un script bash dans le répertoire **/home/vagrant/puppet/files/** :

```
vagrant@ubuntu-xenial:/etc/puppetlabs/code/environments$ cd /home/vagrant/puppet
vagrant@ubuntu-xenial:~/puppet$ mkdir manifests files
vagrant@ubuntu-xenial:~/puppet$ vi files/run-puppet.sh
vagrant@ubuntu-xenial:~/puppet$ cat files/run-puppet.sh
#!/bin/bash
cd /etc/puppetlabs/code/environments/production && git pull
/opt/puppetlabs/bin/puppet apply manifests/
```

Créez ensuite le manifest **run-puppet.pp** dans le répertoire **/home/vagrant/puppet/manifests/** :

```
vagrant@ubuntu-xenial:~/puppet$ vi manifests/run-puppet.pp
vagrant@ubuntu-xenial:~/puppet$ cat manifests/run-puppet.pp
# Set up regular Puppet runs
file { '/usr/local/bin/run-puppet':
  source => '/etc/puppetlabs/code/environments/production/files/run-puppet.sh',
  mode   => '0755',
}

cron { 'run-puppet':
  command => '/usr/local/bin/run-puppet',
  hour    => '*',
  minute  => '*/15',
}
```

Important - Ce manifest copie le script **/etc/puppetlabs/code/environments/production/files/run-puppet.sh** dans le répertoire **/usr/local/bin/** du nœud en le nommant **run-puppet** puis crée un cron job qui appel ce script toutes les 15 minutes.

Ajoutez maintenant les fichiers à Git :

```
vagrant@ubuntu-xenial:~/puppet$ sudo git add manifests files
vagrant@ubuntu-xenial:~/puppet$ sudo git commit -m 'Add run-puppet script and cron job'
[master 756859c] Add run-puppet script and cron job
 2 files changed, 14 insertions(+)
 create mode 100644 files/run-puppet.sh
 create mode 100644 manifests/run-puppet.pp

vagrant@ubuntu-xenial:~/puppet$ sudo git push origin master
Username for 'https://github.com': ittrainingdev
Password for 'https://ittrainingdev@github.com':
Counting objects: 6, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 688 bytes | 0 bytes/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To https://github.com/ittrainingdev/puppet.git
   9139685..756859c  master -> master
```

LAB #8 - Mise en Place sur un Noeud

Votre nœud se trouve dans le répertoire **/etc/puppetlabs/code/environments/production** :

```
vagrant@ubuntu-xenial:~/puppet$ cd /etc/puppetlabs/code/environments/production
vagrant@ubuntu-xenial:/etc/puppetlabs/code/environments/production$ ls
README.md
```

Commencez par faire un **git pull** :

```
vagrant@ubuntu-xenial:/etc/puppetlabs/code/environments/production$ sudo git pull
remote: Counting objects: 6, done.
```

```
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 6 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), done.
From https://github.com/ittrainingdev/puppet
   9139685..756859c  master     -> origin/master
Updating 9139685..756859c
Fast-forward
 files/run-puppet.sh     |  3 +++
 manifests/run-puppet.pp | 11 ++++++++++
 2 files changed, 14 insertions(+)
 create mode 100644 files/run-puppet.sh
 create mode 100644 manifests/run-puppet.pp
```

Vérifiez le résultat :

```
vagrant@ubuntu-xenial:/etc/puppetlabs/code/environments/production$ ls
files  manifests  README.md
```

Appliquez maintenant le mainifest sur le nœud :

```
vagrant@ubuntu-xenial:/etc/puppetlabs/code/environments/production$ sudo puppet apply manifests/
Notice: Compiled catalog for ubuntu-xenial in environment production in 0.20 seconds
Notice: /Stage[main]/Main/File[/usr/local/bin/run-puppet]/ensure: defined content as
'{md5}dbfba978957e90ebb47e3a266b89231b'
Notice: /Stage[main]/Main/Cron[run-puppet]/ensure: created
Notice: Applied catalog in 0.22 seconds
```

Vérifiez que le script a été créé sur le nœud :

```
vagrant@ubuntu-xenial:/etc/puppetlabs/code/environments/production$ cat /usr/local/bin/run-puppet
#!/bin/bash
cd /etc/puppetlabs/code/environments/production && git pull
/opt/puppetlabs/bin/puppet apply manifests/
```

ainsi que le cron job :

```
vagrant@ubuntu-xenial:/etc/puppetlabs/code/environments/production$ sudo crontab -l
# HEADER: This file was autogenerated at 2020-02-11 10:08:47 +0000 by puppet.
# HEADER: While it can still be managed manually, it is definitely not recommended.
# HEADER: Note particularly that the comments starting with 'Puppet Name' should
# HEADER: not be deleted, as doing so could cause duplicate cron jobs.
# Puppet Name: run-puppet
*/15 * * * * /usr/local/bin/run-puppet
```
