Projet Tuteuré :
Configuration automatique d'un cluster de calcul avec Puppet

André Dimitri
Quentin Dexheimer
Riwan Blonde

22 mars 2012

Table des matières

Chapitre 1

Introduction

1.1 Introduction au Projet Tuteuré

1.1.1 blabla made in Riwan

Chapitre 2

Grid5000

2.1 Introduction au Grid 5000

2.1.1 blabla made in Dimitri

Chapitre 3

Puppet Labs

3.1 Introduction à Puppet

3.1.1 Qu'est ce que Puppet Labs ?

Puppet est un logiciel open source comportant les outils nécessaires à la configuration de systèmes informatiques. Il est basé sur le langage de programmation « Ruby », et est sous licence GPL v2. Puppet a principalement été développé par Luke Kanies et son entreprise Puppet Labs.

Kanies a développé puppet grâce à son expérience dans les systèmes Unix et les systèmes d'administration depuis 1997. Non satisfait des outils de configuration existants, il a commencé à travailler avec des outils de développement en 2001, et a fondé Puppet Labs en 2005, une entreprise de développement open source basée sur les outils d'automatisation. Peu de temps après, Puppet Labs sort son nouveau produit phare : Puppet. Il peut être utilisé pour gérer la configuration d'application sous Unix et OSX, ainsi que Linux et Windows depuis peu de temps.

Son modèle est basé sur 3 piliers :

- Le déploiement
- Un langage de configuration et une couche d'abstraction
- Sa couche transactionnelle

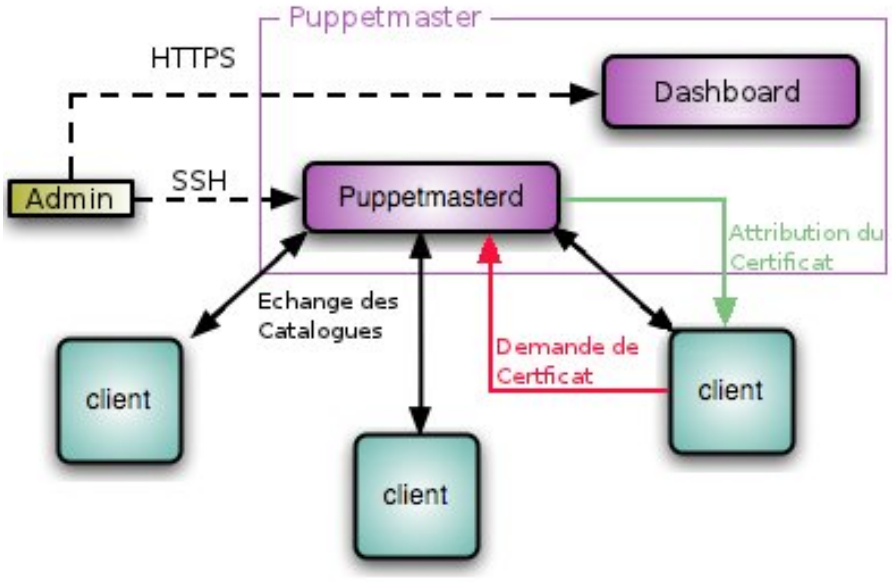


FIGURE 3.1 – Principe d'utilisation de Puppet

3.1.2 Contexte d'utilisation

La gestion au quotidien des configurations systèmes et applicatives d'une entreprise représente un travail très fastidieux. Puppet simplifie grandement la vie des administrateurs : plus de contrôles et d'interventions à réaliser régulièrement. Puppet se charge d'imposer sur les machines des utilisateurs les configurations "modèles" définies par l'administrateur. Puppet est un outil de déploiement et de gestion centralisée de configurations pour les environnements Linux, Unix et Windows ; les machines gérées pouvant être physiques ou virtualisées.

3.1.3 Fonction principale

Puppet est un outil de déploiement et de gestion automatisés de configurations et de systèmes informatiques (serveurs, postes de travail..).
Il repose sur un modèle client/serveur : un serveur central sert de dépôt de configurations, les systèmes clients (nœuds) se mettant à jour de manière manuelle ou automatique.
Avec Puppet, l'administrateur n'écrit pas un ensemble d'opérations à exécuter sur les différents nœuds sous la forme d'un script, l'administrateur décrit l'état final de la machine dans un Manifest, ce qui l'affranchit de la connaissance des commandes propres à chaque système d'exploitation pour arriver à cet état. Le client Puppet peut être exécuté plusieurs fois, les changements seront opérés seulement si l'état de la machine ne correspond pas à celui désiré.

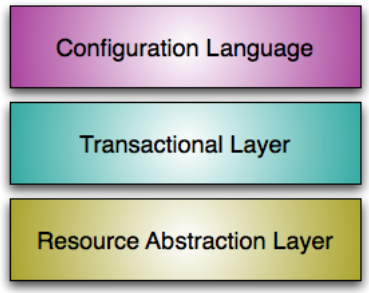


FIGURE 3.2 – Fonction principale de Puppet

Déploiement

Puppet est habituellement déployé sur un simple modèle client/serveur. Le serveur est appelé « Puppet Master », le logiciel client est appelé un « agent », et l'hôte lui-même ainsi que les agents sont définit comme des « noeuds ». Le Master s'exécute comme un démon sur un hôte et contient la configuration requise pour l'environnement. Les agents se connectent au Master via une connexion cryptée qui utilise le standard SSL. Il est important de préciser que si l'agent a déjà la configuration requise, puppet ne fera rien. Cela signifie que puppet appliquera seulement les changements sur l'environnement s'ils sont nécessaires. L'ensemble de ces processus est appelé une « exécution de configuration ». Par défaut, l'agent puppet vérifie toutes les 30 minutes le Master afin de voir si des modifications doivent êtres effectuées. Cet intervalle de temps est bien sur paramétrable.

Configuration Lang/Rsrc

Puppet utilise son propre langage de déclaration pour définir les points de configuration qui sont écrits dans une « Ressource ». Ce qui permet de distinguer puppet de beaucoup d'autres outils de configuration. Ce langage permet de déclarer si un package doit être installé ou si un service doit être lancé par exemple.

La plupart des outils de configuration (scripts shell ou pearl par exemple) sont procéduraux. Ils décrivent comment les choses doivent êtres faites plutôt que de se focaliser sur l'état final attendu.

Les utilisateurs de puppet ont juste besoin de déclarer l'état final voulu de ses hôtes : les packages à installer, les services à exécuter etc. Avec puppet, l'administrateur système n'attache pas d'importance sur comment ces actions vont être faites.

Transaction Layer

Le moteur de puppet est sa couche transactionnelle. Une transaction puppet :

- Interprète et compile la configuration,
- Communique la configuration compilée à l'agent,
- Applique la configuration sur l'agent,
- Rapporte le résultat de cette application au Master.

La 1ère étape de puppet est d'analyser la configuration et de calculer comment l'appliquer sur l'agent. Pour cela, puppet crée un graphique représentant toutes les ressources, ainsi que leurs relations entre elles et chaque agents. Puis puppet applique chaque ressource sur un hôte. Ce qui en fait une des caractéristiques les plus puissante de puppet.

Ensuite puppet se sert des ressources et les compile dans un catalogue pour chaque agent. Le catalogue est envoyé à l'hôte et appliqué par l'agent puppet. Les résultats de cette application sont renvoyés au Master sous forme de rapport.

La couche transactionnelle permet aux configurations d'être créées et appliquées indéfiniment sur un hôte. Ceci est appelé « idempotent », cela signifie que de multiples applications de la même opération donneront le même résultat. Une configuration de puppet peut être exécutée plusieurs fois sur un hôte en toute sécurité avec le même résultat, assurant ainsi à la configuration de rester compatible.

3.2 Commencer avec Puppet

3.2.1 Installation

Nous allons à présent passer à l'installation de Puppet sur une machine serveur et sur une machine cliente. Nous utiliserons le système d'exploitation Debian Squeeze, puisque c'est celui que nous utiliserons plus tard lors du travail effectué.

Cependant, il faut savoir que Puppet est disponible sur toutes les distributions Linux et Unix (OpenSolaris, BSD, RedHat, Debian et Ubuntu) ainsi que sur Windows Server 2008.

Côté Serveur

Les pré-requis pour installer Puppet sont les packages **ruby** et **libshadow-ruby1.8**.

Nous allons utiliser le package **Puppetmaster** disponible dans l'APT, nous n'aurons donc pas à devoir gérer les dépendances liées à l'utilisation de Puppet. Ainsi pour installer Puppet, nous entrons la commande suivante :

```
sudo apt-get install puppet puppetmaster facter
```

En installant le paquêt Puppet sur la machine serveur, nous lui permettons de s'auto-gérer. Pour les machines clientes, il suffit d'installer les mêmes paquêts a l'exeptions du Puppetmaster.

Les packages installés

Liste des différents paquets installés et leur utilités :

Package puppet : Ce paquet contient le script de démarrage et les scripts de compatibilité pour l'agent puppet, qui est le processus responsable de la configuration du nœud local.

Package puppetmaster : Ce paquet contient le script de démarrage et les scripts de compatibilité pour le processus maître puppet, qui est le serveur hébergeant les manifests et les fichiers pour les nœuds puppet.

Package facter : Bibliothèque dit « cross-plateform », qui permet de collecter des informations sur le système. Particulièrement utile dans la collecte d'un nom de système, d'adresse IP et/ou MAC, clé SSH.

3.2.2 Configuration

Côté Serveur

Pour configurer le serveur, il faudra éditer le fichier de configuration « puppet.conf », ainsi que le fichier de hosts :

```
sudo nano /etc/puppet/puppet.conf
sudo nano /etc/hosts
```

Dans le premier, on y ajoute à la fin les lignes suivantes :

```
# /etc/puppet/puppet.conf
[master]
certname=puppetlabs
```

Et dans le seconde fichier :

```
# /ect/hosts
127.0.0.1 localhost
192.168.1.67 puppetlabs
```

Cela permet de spécifier le nom du « **Puppet Master** ».

Côté Client

Comme pour le serveur il faudra éditer les fichiers **puppet.conf** et **hosts** :

```
# /etc/puppet/puppet.conf
[main]
server=puppetlabs

# /ect/hosts
127.0.0.1 localhost
192.168.1.188 puppetclient
192.168.1.67 puppetlabs
```

Cela permet de définir le serveur maitre pour l'agent Puppet.

3.2.3 Synchronisation

Une fois l'installation et les configurations terminées, on lance le serveur maitre via la commande :

```
sudo /etc/init.d/puppetmaster {start | stop | restart}
```

Lors de la première installation le serveur est déjà démarrer. Sur la machine cliente, on lance une deuxième commande :

```
sudo puppet agent --test
```

On obtient alors une demande de validation de certificat :

```
info: Creating a new certificate request for puppetclient
info: Creating a new SSL key at /var/lib/puppet/ssl/private_keys/puppetclient.pem
warning: peer certificate won't be verified in this SSL session
notice: Did not receive certificate
```

Sur le serveur il faut alors délivrer le certificat pour la machine client :

```
sudo puppet cert --list
sudo puppet cert --sign puppetclient
```

Pour avoir la liste complète des commandes : **puppet help**.

Une fois le certificat distribué, on relance la commande précédente et cette fois apparait un nouveau message :

```
notice: Got signed certificate
notice Stating Puppet client
err: Could not retrieve catalog: Could not find default node or by name
with 'puppetclient' on node puppetclient.
```

L'agent du client a bien été connecté avec le serveur mais n'a pas pu trouver un catalogue pour le dit client.

3.3 Manifests

Pour que Puppet fonctionne nous devons lui dire quoi faire. Ceci se fait grâce aux manifests.

3.3.1 Qu'est ce qu'un manifest ?

Les programmes de Puppet sont appelés «manifest», et ils utilisent l'extension de fichier .pp. Le coeur du language Puppet est la déclaration de ressources, qui représente l'état désiré d'une ressource. Les Manifests peuvent également utiliser des instructions conditionnelles, un groupe de ressources dans des collections, générer du texte avec des fonctions, référencer du code dans d'autres manifests, et bien d'autres choses, mais tout revient en dernière analyse à faire en sorte que les bonnes ressources soient gérées de la bonne façon.

Avant d'être appliqués, les manifests sont compilés dans le catalogue, qui est un graphe acyclique dirigé qui ne représente que les ressources et l'ordre dans lequel elles doivent être synchronisées. Toute la logique conditionnelle, la recherche de données, l'interpolation de variables, et le regroupement de ressources calcule l'écart lors de la compilation.

Parmis les manifests deux sont importants. Lorsque le client consultera le Puppet Master, l'agent lira les fichiers dans un certain ordre :

- le fichier /etc/puppet/manifests/site.pp qui dit à Puppet où et quelle configuration charger pour les clients. C'est dans ce fichier que nous ferons la déclaration des ressources, on spécifiera également d'importer les "nodes".

```
1 import 'nodes.pp'
```

- le fichier /etc/puppet/manifest/node.pp est lu lors de l'import. Dans ce fichier, nous renseignons les FQDN des clients ainsi que les modules qui leur sont associés.

```
1 node 'oar-server.ptut-grid5000.lan' { include oar-server, apache, mysql }
```

3.3.2 Ressources

Une ressource est un élément que Puppet sait configurer :

- file (contenu, permissions, propriétaire)
- package (présence ou absence)

- service (activation/désactivation, démarrage/arrêt)
- exec(exécution commande)
- cron, group, host, user etc ...

Dans Puppet chaque ressource est identifiée par un nom, elle est composée d'un certain nombre d'attributs qui ont chacun une valeur. Le language de Puppet represente une ressource comme ceci :

```
user { 'dave':
  ensure     => present,
  uid        => '507',
  gid        => 'admin',
  shell      => '/bin/zsh',
  home       => '/home/dave',
  managehome => true,
}
```

3.4 Modules

3.4.1 Qu'est ce qu'un module ?

Un module est une collection de manifests, ressources, fichiers, templates, class et définitions. Un simple module contiendra tout ce qui est requis pour configurer une application particulière. Par exemple il pourra contenir toutes les ressources spécifiées dans le manifest, fichiers et configuration associée pour Apache. Chaque module à besoin d'une structure de répertoire spécifique et d'un fichier appelé init.pp. Cette structure autorise Puppet à charger automatiquement les modules. Pour accomplir ce chargement automatique Puppet vérifie une série de répertoires appelée le chemin de module (the module path en anglais). Le module path est configuré avec l'option de configuration du module path dans la section [main] de puppet.conf.

Par défault, Puppet recherche des modules dans /etc/puppet/modules et /var/lib/puppet/modules. Si nécessaire on peut rajouter d'autres chemins.

3.4.2 Structure d'un module

Un module est définit de la façon suivante :

1. /etc/puppet/nom-du-module/manifests/ :
   - Le répertoire manifests contiendra notre fichier init.pp et tout autre configuration.
   - Le fichier init.pp est le coeur de notre module et chaque module doit en avoir un.
   - Dans le fichier init.pp nous allons retrouver des classes qui seront instanciées lors de l'appel d'un agent. Dans ces classes on retrouve les configurations de référence.
2. /etc/puppet/nom-du-module/files/ :
   - Le répertoire files contiendra tous les fichiers que ne souhaitons utiliser comme une partie de notre module, par exemples des fichiers à "uploader".
3. /etc/puppet/nom-du-module/templates/ :
   - Le répertoire "templates" contiendra tous les templates que notre module pourrait utiliser.

3.4.3 Exemple concret de module

Premièrement création de l'arborescence :

- `mkdir -p /etc/puppet/modules/mysql/{files,templates,manifests}`

Ensuite nous allons créer le fichier init.pp avec la commande suivante :

- `touch /etc/puppet/modules/mysql/manifests/init.pp`

Une fois l'arborescence créée nous devons configurer le fichier init.pp comme ceci :

```
1 import 'install.pp'
2 import 'config.pp'
3 import 'service.pp'
4
5 class mysql {
6   include mysql::install, mysql::config, mysql::service
7 }
```

Notre init.pp contient une seule class appelée mysql. A cela, on y ajoute trois fichiers install.pp, config.pp et service.pp contenant respectivement les class apache : :install et apache : :service. La class apache dit au puppetmaster d'aller executer les class inclusent.

Les trois fichiers doivent se trouver dans le même dossier que le init.pp.

Le premier fichier, install.pp, s'assure que les paquets MySQL-server, MySQL-client et Phpmyadmin sont bien installés.

```
class mysql::install {
  package { [ "mysql-server", "mysql-client", "phpmyadmin" ]:
    ensure => present,
    require => User["mysql"],
  }

  user {
    "mysql":
    ensure => present,
    comment => "MySQL user",
    gid => "mysql",
    shell => "/bin/false",
    require => Group["mysql"],
  }
  group {
    "mysql":
    ensure => present,
  }
}
```

Il permet aussi de créé un utilisateur MySQL et un groupe MySQL auquel on donne les droits administrateurs sur MySQL.

Le second fichier, config.pp, utilise la class mysql : :config. Elle est une class de type "ressource fichier". Elle permet de copier un fichier de configuration de MySQL prédéfinie.

```
class mysql::config {
  file {
    "/etc/mysql/my.cnf":
      source  => "puppet:///mysql/mysql-conf",
      ensure  => file,
      owner   => root,
      group   => root,
      mode    => 644,
      notify  => Service["mysql"],
      require => Class["mysql::install"];
  }
}
```

L'option "source" indique a Puppet où est situé le fichier. Généralement, il est situé dans le dossier files du module. Cependant, on ne spécifie pas tout le chemin d'accès. Juste le nom du module et nom du fichier. Puppet sait qu'il faudra directement chercher dans le dossier files.

Les autres options permettent de définir le propriétaire, le groupe et les permissions du fichier.

Le troisième fichier importé, service.pp, sert a vérifié que le service récemment installé.

```
class mysql::service {
  service { "mysql":
    ensure     => running,
    hasstatus  => true,
    hasrestart => true,
    enable     => true,
    require    => Class["mysql::install"],
  }
}
```

3.5 Complément à Puppet

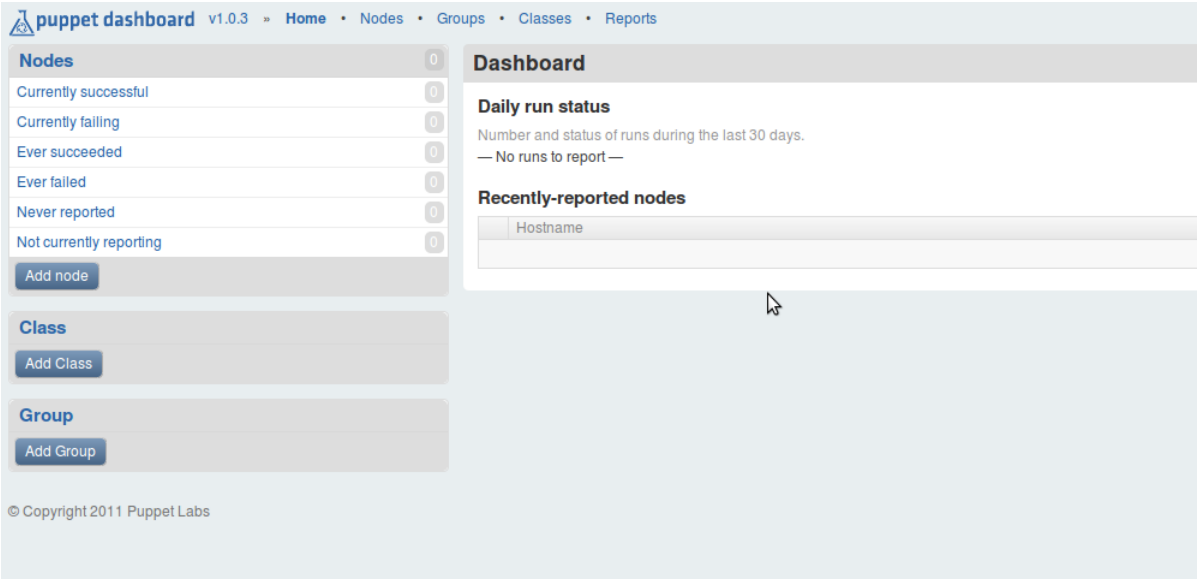
3.5.1 Puppet Dashboard

Aperçu

Puppet Dashboard est une application avec interface web permettant d'afficher des informations sur le serveur Puppet et son environnement (Master et Agents). Il permet de voir sur des graphes, les données répertoriées d'un serveur Puppet ou de plusieurs (multi-puppet).

Il fait aussi l'inventaire de données à partir des agents (à la façon d'un OCS inventory).

Enfin, il peut être utilisé comme console de configuration pour le serveur Puppet et de ses nodes (désignation des classes et/ou paramètres, etc...).



Installation

Comme tout Puppet, son installation est relativement simple puisqu'une ligne de commande via l'APT est requise. Cependant le dépôt "Puppet Labs APT" doit être ajouté dans le fichier sources.list :

```
#/etc/apt/sources.list
deb http://apt.puppetlabs.com/ubuntu lucid main
deb-src http://apt.puppetlabs.com/ubuntu lucid main
```

A fortiori, la clé GPG doit aussi être ajoutée :

```
sudo gpf --recv-key 4BD6EC30
sudo gpg -a --export 4BD6EC30 > /tmp/key
sudo apt-key add /tmp/key
```

Finalement, la commande pour l'installer via l'APT est :

```
sudo apt-get install puppet-dashboard
```

Pour se connecter à l'interface web on exécute la commande suivante puis dans la barre navigation de son navigateur web, l'url qui suit :

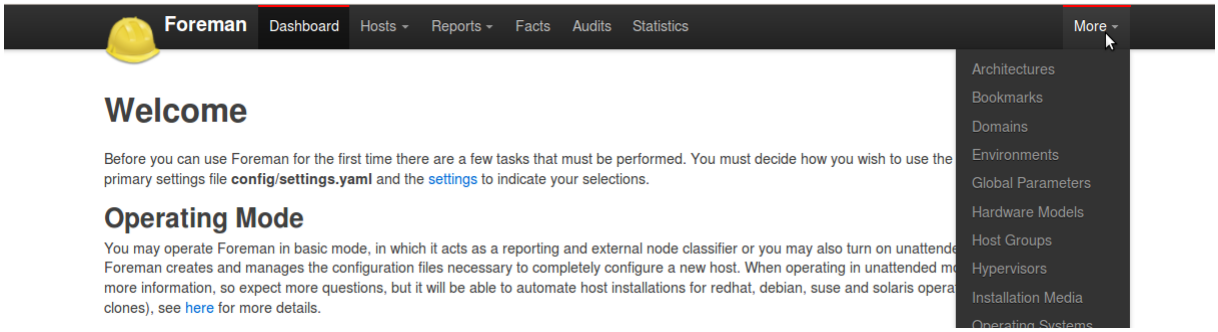
```
sudo /etc/init.d/puppet-dashboard start
http://puppetlabs:3000/
```

3.5.2 The Foreman

Introduction

Une autre interface web pour Puppet est "The Foreman" ou "Foreman". Elle aussi est réalisée en Ruby. Cette interface permet de montrer l'inventaire du parc informatique basé sur Facter et donne des informations en temps réel sur le statut des agents basées sur les "Puppet reports".

Remarque : Contrairement à Puppet-Dashboard, developpé par l'équipe officielle Puppetlabs, The Foreman a été conçu et réalisé par un fan indépendant bien avant Dashboard.



Installation

l'installation est semblable aux autres avec là aussi un ajout de répertoire/clé pour l'APT.

```
deb http://deb.theforeman.org/ stable main
wget http://deb.theforeman.org/foreman.asc
sudo apt-key add foreman.asc
sudo apt-get install foreman-mysql

sudo /etc/init.d/foreman start
http://puppetlabs:3000/
```

3.5.3 Puppet Module Tool

Introduction

Outre les deux interfaces web, « Puppet Forge », propose énormément de modules qui peuvent permettre de faciliter le déploiement rapide d'une infrastructure complexe. Ces modules sont configurés de telle sorte qu'il vous est facile de les modifier s'il le faut.

Les Modules peuvent être téléchargés ou installés manuellement, mais il existe un "module manager", Puppet-module tool.

Puppet-module tool permet de :
- Créer des squelettes pour vos futurs modules.
- Chercher des modules déjà existants.
- Installer et configurer des modules déjà existants.

Installation

Le package de Puppet Module Tool a la particularité d'être exclusivement dans le « RubyGems repository », cela rend plus simple son installation sur différentes plateformes où RubyGems est installé.

```
sudo gem install puppet-module
```

Utilisation

Pour chercher un module déjà existant, on utilise la commande « search » :

```
puppet-module search mysql
```

On obtient alors un résultat comme ceci :
Pour installer un des résultats trouvés, on utilise la command « install » :

```
puppet-module install ghoneycutt/mysql
```

nous obtenons ainsi un répertoire mysql :

Chapitre 4

Travail effectué

4.1 Travail effectué

4.1.1 blabla made in Quentin

Chapitre 5

Conclusion