

DAWAN Paris
DAWAN Nantes
DAWAN Lyon

11,rue Antoine Bourdelle, 75015 PARIS
32, Bd Vincent Gâche, 5e étage - 44200 NANTES
Bt Banque Rhône Alpes, 2ème étage - 235 cours Lafayette 69006 LYON



Formation Puppet: Initiation & Approfondissement

Plus d'info sur <http://www.dawan.fr> ou **0810.001.917**



Présentation de la formation

Qui suis-je? « mon parcours » « mes compétences »

Exprimer votre besoin:

- Le contexte **IT**
- Votre environnement de travail
- Vos **objectifs** pour cette formation.

Objectifs

Comprendre le mouvement **DevOps** et l'industrialisation

Gestion de version : **Git**

La philosophie **Puppet**

Installation et configuration **Puppet Master & Puppet agent**

Configurations avancées / modules / templates / hiera

Avec ces acquis vous serez en mesure de :

- ✓ Installer et configurer une plateforme Puppet
- ✓ Centraliser l'administration d'un environnement hétérogène
- ✓ Modéliser et déployer une infrastructure type infra-as-code

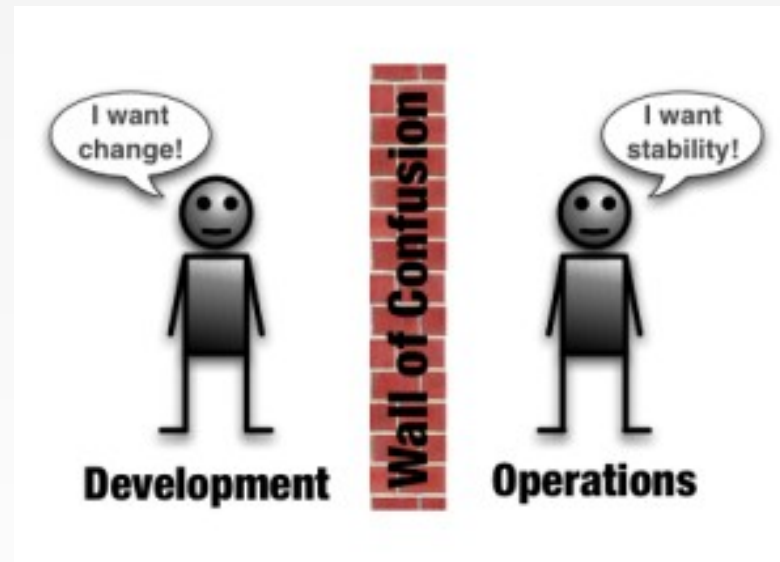




Que vous évoque le terme **DevOps**?
Fonctionnalités **Puppet** ?
Les **Facts** ?

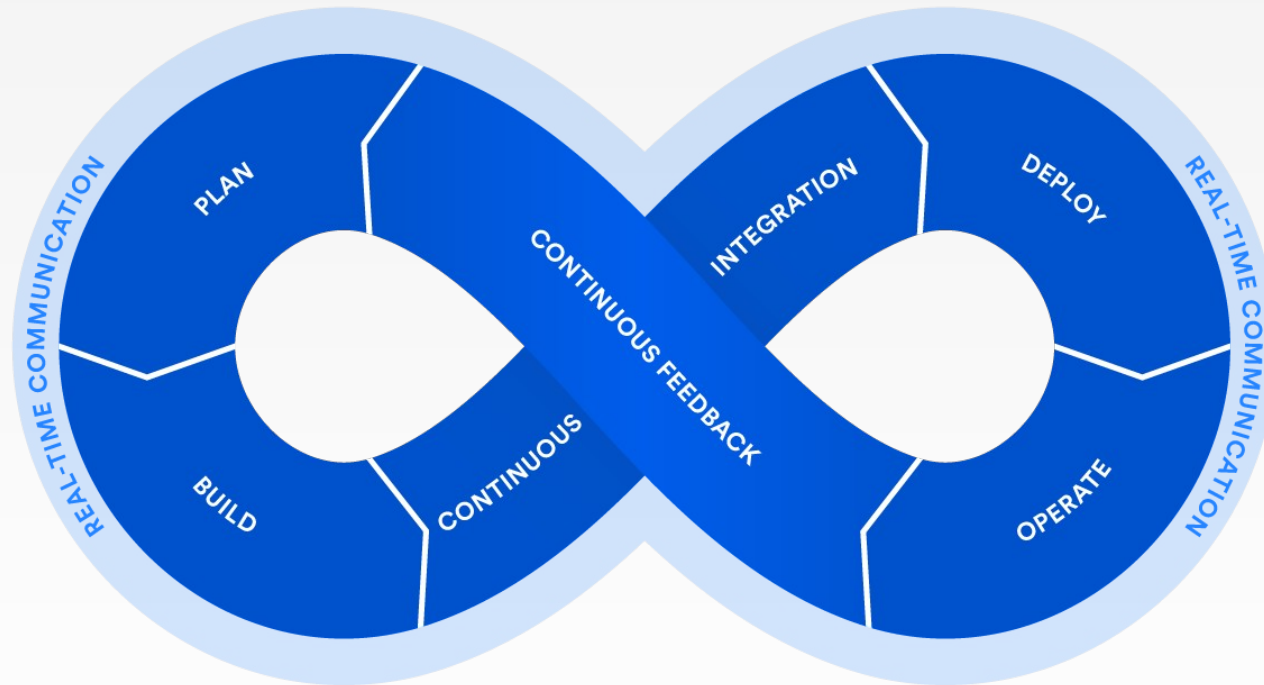
Introduction DevOps

- L'origine du mouvement DevOps



Introduction DevOps

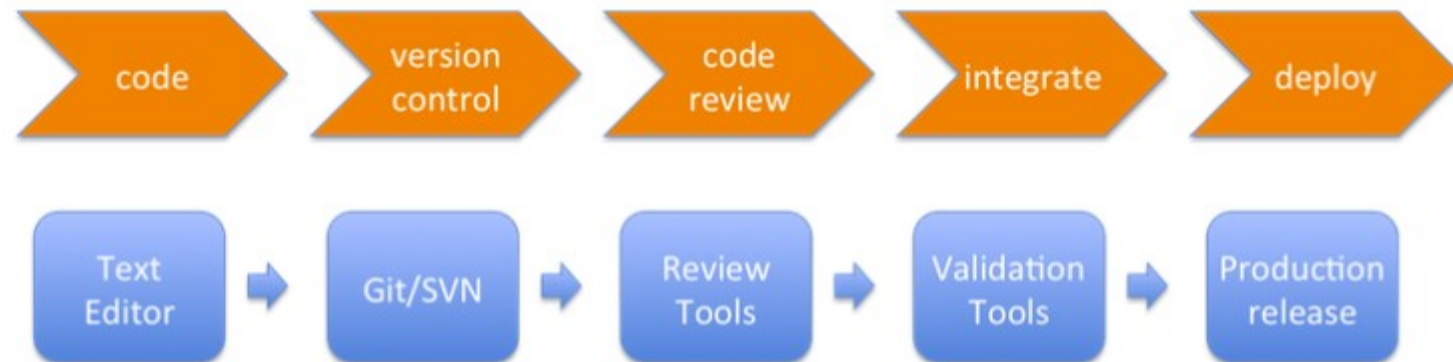
- Les besoins d'industrialisation pour l'exploitation



Infrastructure as Code

- Infrastructure as Code

- vs configuration manuelle
- les + : gestion de version, CI/CD, collaboratif

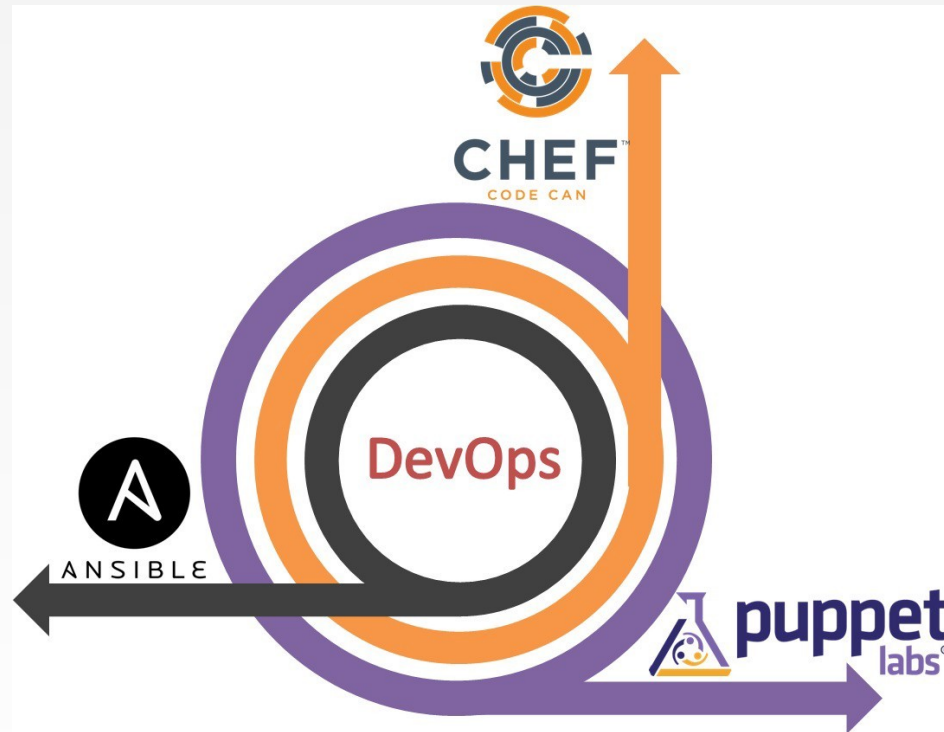


"It's all software"



Introduction DevOps

- Les outils



Présentation de git

- Créé en avril 2005 par Linux Torvalds
 - (gestion du workflow d'intégration des patches du noyau Linux)
- Version Control System
 - Enregistre et maintient les modifications de fichiers
 - Permet de revenir sur une version spécifique d'un fichier (passé ou future)
- Facilite le développement parallèle
 - Branches (dev, master, hotfix)
 - Merges, pull request
- Gère des projets de taille importante (Linux, Android, Gnome...)
- Architecture simple et distribuée

Présentation de git

- Version Control System
 - Enregistre et maintient les modifications de fichiers
 - Permet de revenir sur une version spécifique d'un fichier (passé ou future)
- Facilite le développement parallèle
 - Branches (dev, master, hotfix)
 - Merges, pull request
- Gère des projets de taille importante (Linux, Android, Gnome...)
- Architecture simple et distribuée



Introduction DevOps



- Positionnement de Puppet dans la paysage
 - Figure de proue du DevOps (rattrapé par Ansible)
 - Fonctionne de préférence avec un agent (et donc un maître)
 - Écosystème extrêmement important
 - Développé en Ruby
 - Permet de gérer des déploiements système & applicatifs

Présentation de Puppet

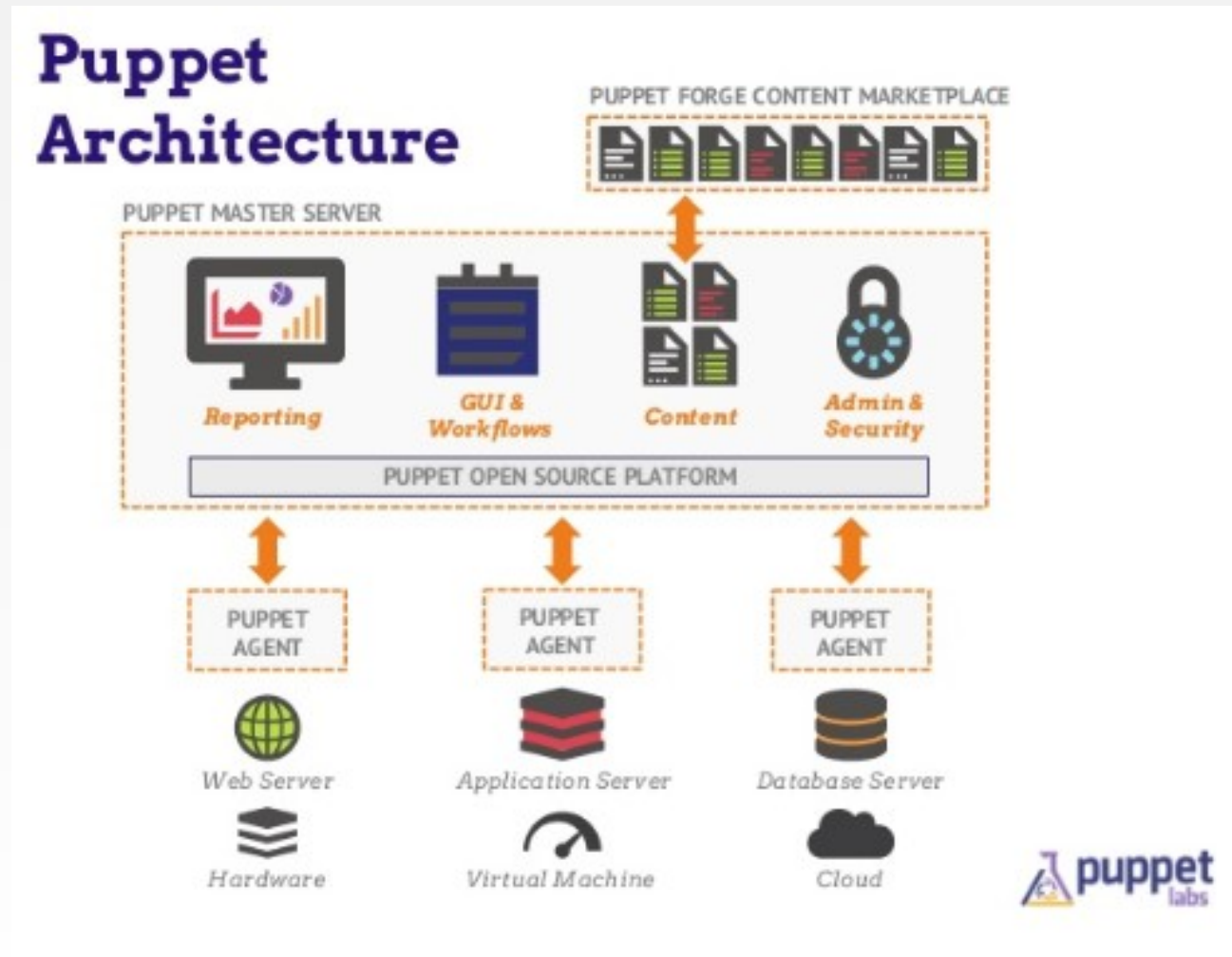
- Système de gestion centralisée des configurations
- Relation Serveur - Agents
- Projet mature (depuis 2005)
- 2021 : puppet 7
- PuppetLabs : Enterprise et Open source
- 2013 - 2014 : 30 millions \$ et 40 millions \$ (vmware, cisco, google)
- Encore très présent dans les grands SI qui ont fortement investi dessus



EnterpriseVSOpen

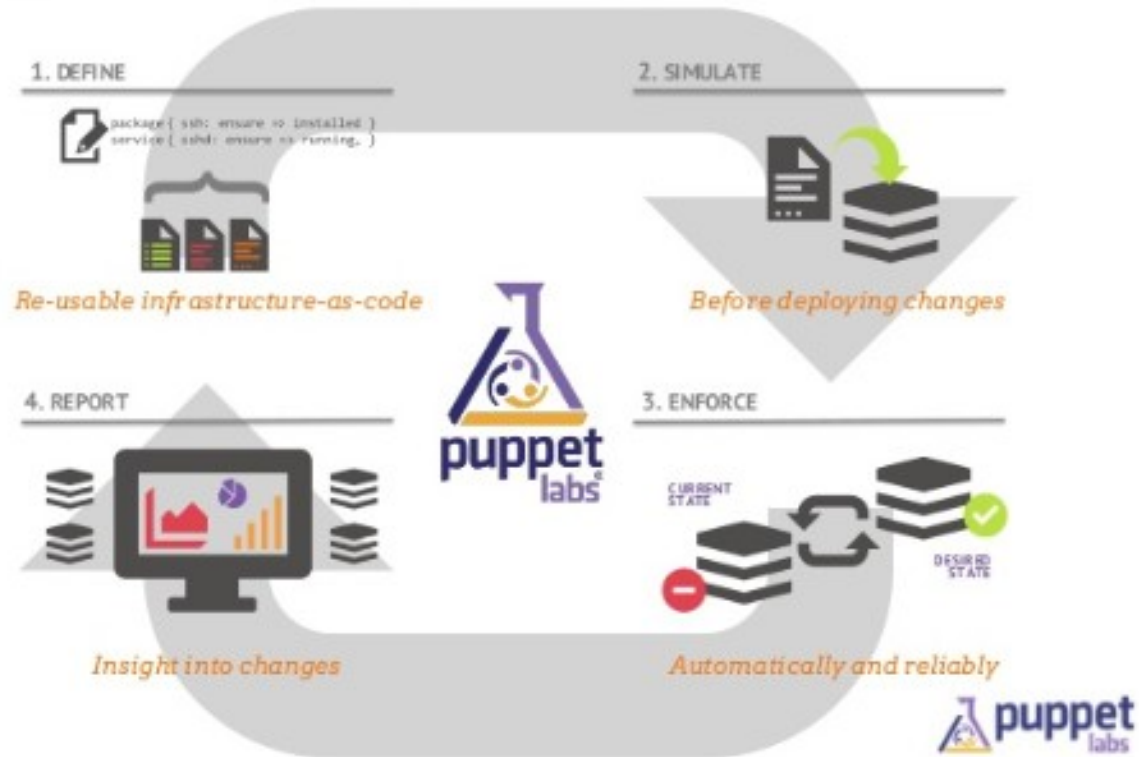


Puppet architecture



Puppet architecture

New Approach: Software Defined Infrastructure



Déclaratif vs Impératif

Impératif

Dessine moi un mouton (avec un crayon bien taillé sur une feuille blanche en papier recyclé...)

Comment

Déclaratif

Apporte moi le dessin d'un mouton

Quoi

Déclaratif vs Impératif

Comparison

Imperative Shell Code

```
if [ 0 -ne $(getent passwd elmo > /dev/null)$? ]
then
    useradd elmo --gid sysadmin -n
fi

GID=$(getent passwd elmo | awk -F: '{print $4}')
GROUP=$(getent group $GID | awk -F: '{print $1}')

if [ "$GROUP" != "$GID" ] && [ "$GROUP" != "sysadmin" ]
then
    usermod --gid $GROUP $USER
fi
```

```
if [ "`getent group sysadmin | awk -F: '{print $1}'`" == "" ]
then
    groupadd sysadmin
fi
```

Declarative Puppet Code

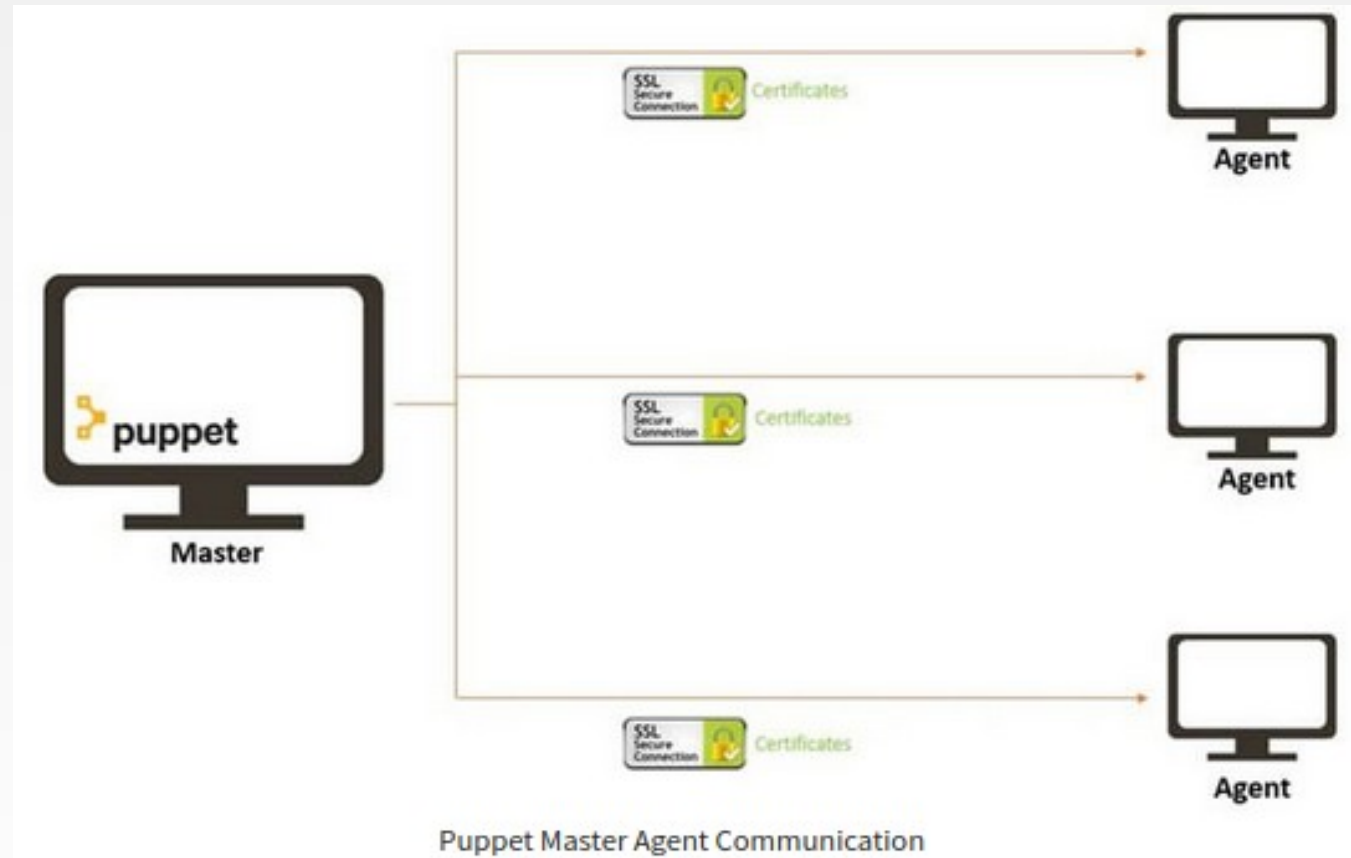
```
user { 'elmo':
  ensure => present,
  gid    => 'sysadmin',
}
```

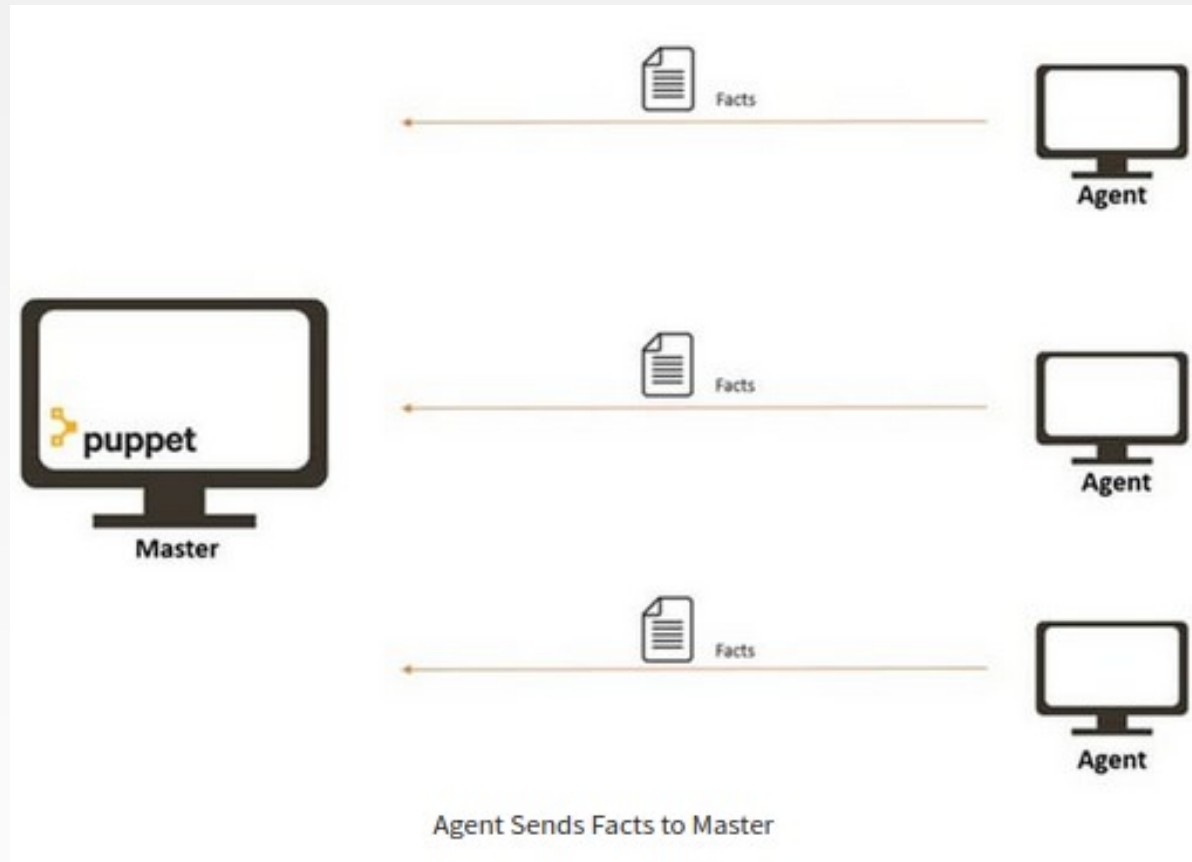
```
group { 'sysadmin':
  ensure => present,
}
```




Comment fonctionne Puppet ?

- Basé sur un modèle de déploiement « **pull** »
- Les agent s'enregistrent régulièrement ou à la demande avec le nœud maître pour voir si quelque chose doit être mis à jour dans l'agent
- Si action nécessaire, l'agent reçoit le code compilé (catalogue) et exécute
- La communication entre maître et agent est établie via des certificats sécurisés
- Un agent s'installe sur Windows/Linux/MacOS/Solaris
- Un master/maître s'installe sur Linux







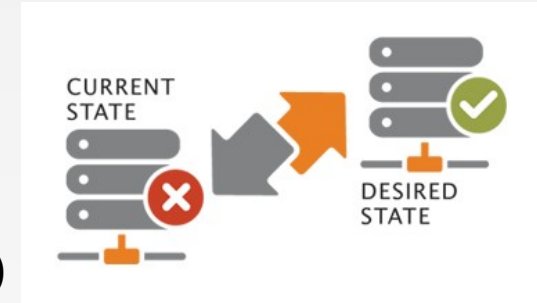
Master sends a catalog to Agent



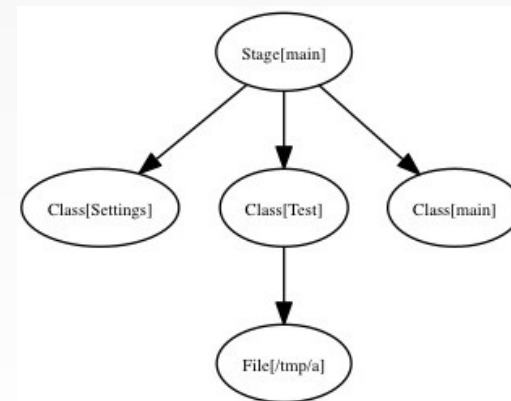
Agent applies configuration

Fonctionnement du catalogue

- Le catalogue : décrire un état désiré
- Fonctionnement : autonome ou via un agent (préféré)
- Deux étapes : compilation puis exécution

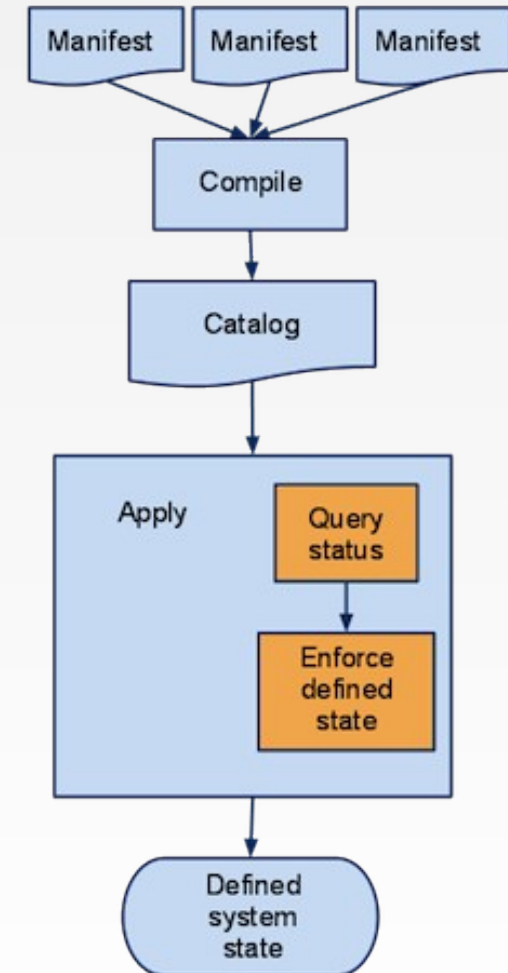


```
class test {  
  file {  
    "/tmp/a": content => "test!"  
  }  
}  
  
include test
```



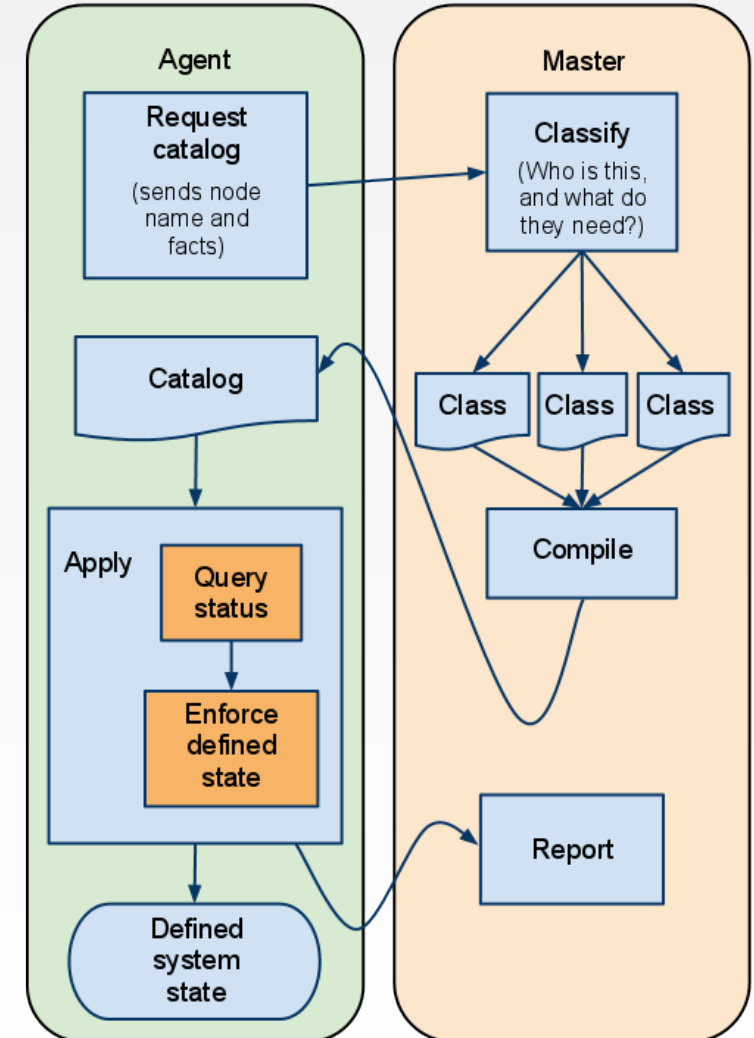
Architecture autonome

- Manifests compilés pour construire le catalogue de ressources
- Utilisation d'un manifest principal : site.pp
- Stand-Alone : puppet apply
- Pilotage planifié : crontab ou ordonnanceur
- Bootstrapping, ou cas particulier (pas d'accès réseau)
- Accès à toutes les ressources
- compilation en local du catalogue
- Intéressant pour débiter



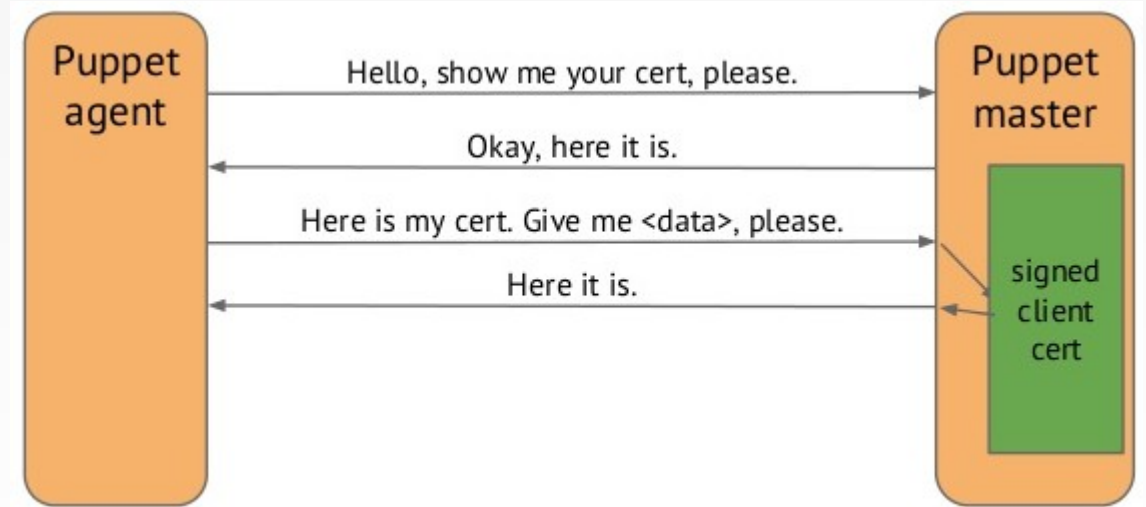
Architecture Agent/Master

- C'est le fonctionnement canonique
- Compilation du catalogue sur le maître, diffusé, enregistré en local sur les agents
- Remontée des faits des agents vers le maître
- Rapports d'exécutions renvoyés sur le master



Implications sécuritaires

- Gestion de certificats/clés
- Communication agent/maître via HTTPS
- Principe du moindre privilège
- Reporting centralisé



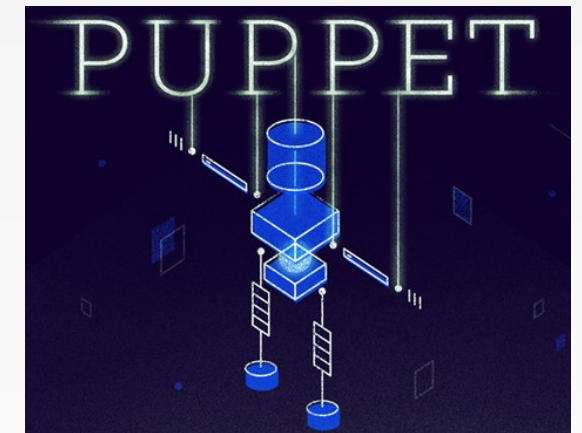
Installation, présentation

- Pré-requis matériels pour le maître
- Pré-requis logiciels : Ruby, Facter, Hiera, gem, json
- Distribution supportées : facilité d'installation



- Pré-requis externes : DNS, NTP

System requirements





Installation Agent



- Installation du dépôt PC1
 - <https://apt.puppetlabs.com/>
- Installation du puppet master
 - puppetserver, puppet, puppet-agent
 - /etc/puppetlabs/puppet/puppet.conf
- Installation du paquet puppet-agent sur un nœud
- Création d'un premier manifeste : premier_manifeste
- Mise en place du manifeste

Atelier pratique

- Installation de puppet, mise en place d'un catalogue trivial
 - Création de 4 Vms : 3 Ubuntu, 1 Centos via VirtualBox
 - Nommage de VMS, résolution de noms (puppet, node1 et node2,,)





Vocabulaire puppet

- **Noeud** (Node) : serveur ou poste de travail administré par Puppet
- **Site** : ensemble des noeuds gérés par le Puppet Master
- **Ressource** (Resource): objet que Puppet peut manipuler (fichier, utilisateur, service, package, etc.)
- **Classe** : moyen dans Puppet de séparer des morceaux de code
- **Module** : unité de code Puppet qui est réutilisable et pouvant être partagé
- **Facter** : librairie multi-plateforme qui fournit à Puppet sous forme de variables les informations propres au système (nom, adresse ip, système d'exploitation, etc.)
- **Manifeste** (Manifest) : regroupe un ensemble de ressource.
- **Catalogue** : ensemble des classes de configuration à appliquer à un nœud



Découverte du langage

- Les ressources dans Puppet
- La couche d'abstraction des ressources
- Les manifestes
- Idempotence, états souhaités
- Ordonnancement des ressources
- Variables, faits (facts avec facter)
- Conditions
- Classes, types définis, modules



Découverte du langage

- Fichiers
 - manifests .pp : langage Puppet
 - Templates
 - utilisés par les manifests
 - langage Puppet .epp ou Ruby .erb
- langage Puppet
 - Ressources
 - Variables
 - opérateurs, structures de contrôle : expressions conditionnelles, boucles
 - fonctions ⇒ opérations complexes : fonctions built-in, fonctions des modules - ex. module stdlib



Les ressources dans Puppet



- La ressource : unité atomique de configuration
- Caractéristiques : déclaratives, idempotentes, uniques
- 1 ressource Puppet = 1 "chose" gérée sur la machine cible
- abstraction selon l'OS (RAL - Resource Abstraction Layer)
- Type de base : package, file, service, user...
- Listing des ressources :
 - `# puppet describe -l`
 - `# puppet ressource service`

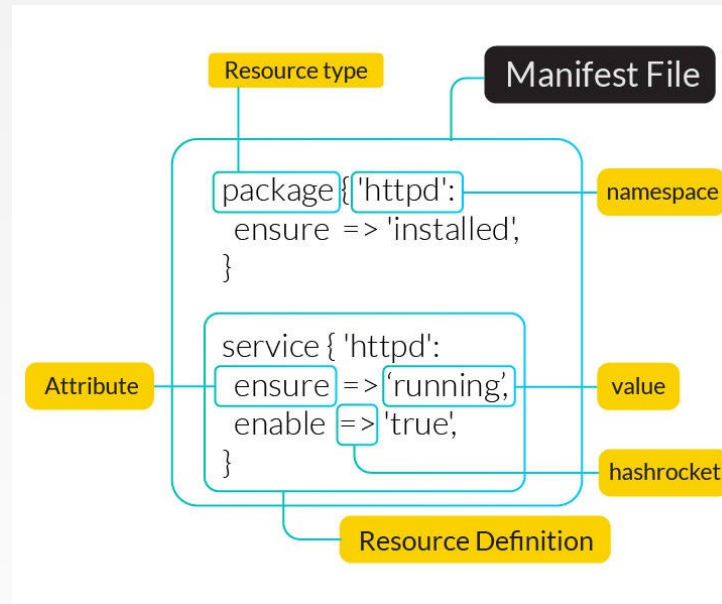
La couche d'abstraction des ressources (RAL)

- Abstraction des ressources : typage, description, état désiré
- Langage de description Puppet :

```
type { 'title':  
  argument => value,  
  other_arg => value,  
}
```

Les manifestes

- Organisation des manifestes (.pp) : liste de ressources



Attention à la casse :

=> tout les déclarations se font en minuscule

Protéger les chaîne par des guillemets

- Application via puppet apply manifeste.pp
- Affichage de messages avec la ressource notify

Atelier pratique

- Récupérer le détail de la ressource user 'root'
- Comment décrire un service démarré ?
- Écrire la description d'une ressource utilisateur :
 - Prénom
 - Shell
 - Gecos





Idempotence, états souhaités

- Puppet ne décrit pas des actions, mais des états désirés
 - « ensure »
- Exercice :
 - Mettez l'utilisateur décrit dans un manifest user.pp et l'appliquer sur un node
 - Supprimez-le, en modifiant l'état souhaité de la ressource

```

user { 'joe':
  # (namevar) The user name
  name      => 'joe',
  # The user's status: 'present','absent','role'
  ensure    => 'present',
  # The user's id
  uid       => '1001',
  # The user's primary group id
  gid       => '1001',
  # Eventual user's secondary groups (use array for many)
  groups    => [ 'admins' , 'developers' ],
  # The user's password. As it appears in /etc/shadow
  # Use single quotes to avoid unwanted evaluation of $* as variables
  password  => '$6$ZFS5JFFRZc$FFDSvPZSSFGVdXDlHe?',
  # Typical users' attributes
  shell     => '/bin/bash',
  home      => '/home/joe',
  mode      => '0644',
}

```

Ordonnancement des ressources

- Défaut : ordre de déclaration dans le manifest
- Dépendance entre les ressources :
 - before/require – notify/subscribe (pour provoquer un redémarrage)
- Le classique « trifecta » : package/file/service

```
package { 'exim':  
  before => File['exim.conf'],  
}  
  
file { 'exim.conf':  
  notify => Service['exim'],  
}  
  
service { 'exim':  
}
```



Casse !

Majuscule au type de la ressource

Déclaration : minuscule

Appel : majuscule

- Notation -> / ~> : pour marquer la dépendance / le notify

```
Package['exim'] -> File['exim.conf'] ~> Service['exim']
```


Ordonnancement des ressources

Ansible *impératif concret séquentiel*

```
- name: install apache
  yum: name=httpd state=latest
  become_user: root
  become: true
- name: start and enable apache service
  service: name=httpd state=started enabled=yes
  become_user: root
  become: true
```

Puppet *déclaratif abstrait et dépendant*

```
package {'httpd':
  ensure => 'latest'
}
service {'httpd':
  ensure => 'running',
  enable => true,
  require => Package['httpd'],
}
```



```
package { 'exim':  
}  
  
file { 'exim.conf':  
  require => Package['exim'],  
}  
  
service { 'exim':  
  subscribe => File['exim.conf'],  
}
```


Atelier pratique

- Écrire un manifeste qui définit une clé SSH publique pour l'utilisateur root
- Appliquer le manifeste avec puppet apply ssh
- Bonus : reprendre sshd_config et exiger une clé SSH pour root (après s'être assuré qu'elle est en place), modifier le manifeste pour pousser la configuration et déclencher un redémarrage du service



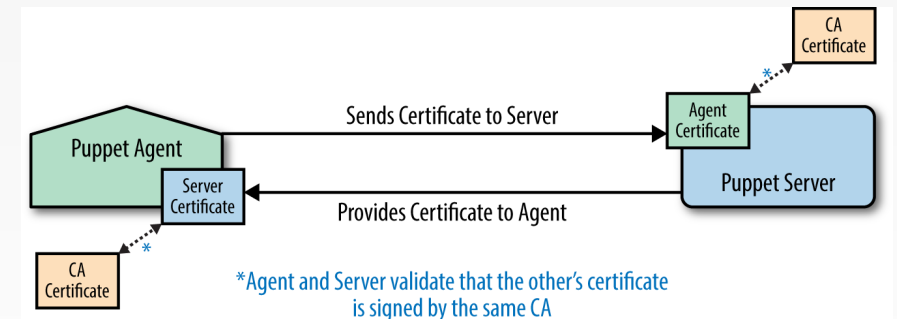


Mise en place maître/agent

- Configuration et démarrage puppetserver
 - `# systemctl start puppetserver`
- Configuration Agent et contact avec le master
 - `# puppet agent --test`
- Méthode de démarrage des agents : crontab, démon
- Manifest puppet.pp ?
- Configuration basique

Certificats et configuration

- Lister les certificats en attente :
 - `# puppetserver ca --list`
- Signer les certificats :
 - `# puppetserver ca sign --certname srv01.formation.1an`
 - *autosign*
- Arborescence puppet master :
 - Logs
 - Manifests
 - `# /etc/puppetlabs/code/environments/production/manifests/*.pp`



Atelier pratique

- Lancer l'agent puppet sur les nœuds
- Signer les certificats sur le maître
- Intégrer ssh.pp et user.pp dans le manifeste global (répertoire)





Variables, faits (facts avec Facter)

- Définition des variables :
 - `$variable = valeur`

Redéfinition des variables

Les variables ne peuvent pas être redéfinies.

- Variable indéfinie : `undef`



Type de base



- Nombres : `$variable = 1`
- Chaîne : `$variable = "toto"`
 - Interpolation : `$variable = "toto$autrevariable\n"`
 - Ou mieux : `$var = "toto${variable}suite"`
 - Sans interpolation : `$variable = '[0-9a-z]+$var'`
 - Caractère d'échappement : `\`

Autres types

- Booléens : true ou false

Valeurs des chaînes et nombres

La chaîne vide vaut faux, toutes les autres, y compris "false" valent vrai.
Tous les nombres sont vrais, y compris 0 et les nombres négatifs.

- Tableau : `$tab = ["element1"; "element2"]`
 - Indexation : `$tab[0]`
- Hashes : `$dic = {'cle' => 'valeur', 'cle2' => 'valeur2'}`
 - Indexation : `$dic['cle']`

Faits : facts

- Variable déterminées pour un hôte particulier via "facter"
- Exécution manuelle via facter (dépendance de puppet)
- Utilisation dans le manifeste :
 - \$::fact
 - Ex : \$::fqdn

```
$ puppet facts
{
  "facterversion": "3.2.0",
  "interfaces":
  "cpu,et1,et2,et3,et4,fabric,lo,ma1,vnnicet1,vnnicet2,vnnicet3,vnnicet4",
  "os": {
    "architecture": "x86_64",
    "family": "Linux",
    "hardware": "x86_64",
    "name": "AristaEOS",
    "release": {
      "full": "4.14.9",
      "major": "4",
      "minor": "14"
    },
    "selinux": {
      "enabled": false
    }
  },
  "osfamily": "Linux",
```


Atelier pratique

- Reprendre l'exemple des clés SSH, en mettant la clé dans une variable pour la définir pour les utilisateurs root et user.pp
- Dans un fichier /etc/motd, préciser le nom de l'OS et de la distribution





Conditions



if elsif else

```
if $::osfamily == 'Debian' {  
    $package_name = 'apache2'  
} elsif $::osfamily == 'RedHat' {  
    $package_name = 'httpd'  
} else {  
    notify { "Operating system $::operatingsystem not supported" }  
}
```

Conditions

Selector for variable's assignement

```
$package_name = $osfamily ? {  
    'RedHat' => 'httpd',  
    'Debian' => 'apache2',  
    default  => undef,  
}
```

Conditions

case

```
case $::osfamily {  
    'Debian': { $package_name = 'apache2' }  
    'RedHat': { $package_name = 'httpd' }  
    default: { notify { "Operating system $::operatingsystem not sup  
ported" } }  
}
```

Conditions

```
if $::osfamily == 'Debian' { [ ... ] }  
  
if $::kernel != 'Linux' { [ ... ] }  
  
if $::uptime_days > 365 { [ ... ] }  
  
if $::operatingsystemrelease <= 6 { [ ... ] }
```

Conditions

```
if ($::osfamily == 'RedHat')  
and ($::operatingsystemrelease == '5') {  
    [ ... ]  
}  
  
if ($::osfamily == 'Debian') or ($::osfamily == 'RedHat') {  
    [ ... ]  
}
```



Conditions



```
if '64' in $::architecture  
  
    if $monitor_tool in [ 'nagios' , 'icinga' , 'sensu' ]
```

Notion de classe

- Une classe : un regroupement de ressources
- Notion de définition vs déclaration
- Déclaration des ressources : ajout au catalogue
 - attention à l'unicité

```
class puppet {  
  file { ['/etc/puppet/puppet.conf':  
    content => template('puppet/client/puppet.conf'),  
  }  
}
```


Classe : syntaxe

- Définition :

```
class puppet {  
  file { ['/etc/puppet/puppet.conf':  
    content => template('puppet/client/puppet.conf'),  
  }  
}
```

- Déclaration :
 - include
 - require
- Inherit

Paramètre de classe

- La classe peut définir des paramètres, qui pourront être passés comme attributs lors de la déclaration

```
# Définition de la classe
class unFichier ( String $contenu = 'contenu par défaut' ) {
  file { '/tmp/fichier1':
    content => $contenu,
  }
}

# Déclaration
class { 'unFichier' :
  contenu => 'contenu défini a la déclaration' ,
}
```



Types définis



- Certaines classes ont parfois vocation à être incluses plusieurs fois avec des paramètres différents, mais ceci n'est pas possible
- On transforme alors la class en type, en s'assurant que toutes les ressources définies seront uniques lors de la déclaration

Les modules

- Les modules sont des classes bien rangées !
 - Défaut : /etc/puppet/modules
 - Classe du même nom que le module dans manifest/init.pp
 - Include de la même manière

```
mysql/          # Main module directory

mysql/manifests/ # Manifests directory. Puppet code here. Required
.
mysql/lib/       # Plugins directory. Ruby code here
mysql/templates/ # ERB Templates directory
mysql/files/     # Static files directory
mysql/spec/      # Puppet-rspec test directory
mysql/tests/     # Tests / Usage examples directory

mysql/Modulefile # Module's metadata descriptor
```

Les modules

Serveur de fichier

En utilisant les modules, on gagne la possibilité, en mettant les fichiers dans un répertoire files de préciser une source du type
puppet:///modules/monmodule/fichier.txt afin que le maître distribue le fichier aux nœuds !

```
source => 'puppet:///modules/mysql/my.cnf'  
# File is in: $modulepath/mysql/files/my.cnf
```

```
content => template('mysql/my.cnf.erb'),  
# Template is in: $modulepath/mysql/templates/my.cnf.erb
```

Les modules



- Installation :

Welcome to Puppet Development Kit



Atelier pratique



- Modulariser les classes
- Créer un module agent pour pousser puppet.conf
- Utiliser les sélecteurs et conditions pour rendre les classes multi-distributions
- Tester



Puppet dans l'infrastructure

- La définition des modèles (templates) avec Embedded Puppet (EPP)
- Classification des nœuds
- Classification des nœuds avec Hiera
- Ressources virtuelles, ressources exportées
- Utilisation de la forge, industrialisation des modules
- Modules plus complets, librairie standard

La définition des modèles (templates EPP)

- Permet de pousser un fichier qui contient des variables
- Les templates sont écrits dans un langage de modélisation :
 - Embedded Puppet (EPP)
 - Syntaxe : `<%= @variable %>`
- On dépose les fichiers dans le répertoire templates des modules

Exemple de template Puppet

Exemple de template Puppet

```
file {'/etc/motd':  
  content => epp(  
    'motd/arch.epp',  
    {  
      arch => $facts['os']['architecture'],  
      host => $facts['hostname'],  
    }  
  )  
}
```

+

```
#####  
bonjour  
  
bienvenue sur <%=host%> (<%=arch%>)  
#####
```



Atelier pratique



- Ré-écrire les ressources issues des manifestes sous forme de classes
- Créer une nouvelle classe apache qui :
 - Installe apache
 - Déploie un fichier index.html indiquant la distribution et le nom du nœud
- Vérifier que le manifeste est bien appliqué sur les nœuds
- Créer un fichier params.pp pour y stocker toutes les variables, il doit être pris en compte par héritage
- Vérifier que la manifeste intègre bien les variables du params.pp



Atelier pratique



- Remplacer la méthode de création du fichier `/etc/motd` par l'utilisation d'un template avec la structure suivante :
 - Commentaire
 - HOST:
 - OS:
 - IP:
 - Condition if
 - INFRA: 64 Bits
 - CPU:

Classification des nœuds

- But : déclarer certaines classes sur certains nœuds

```
node 'web01' {  
  include apache  
}
```

```
node 'web01' , 'web02' , 'web03' {  
  include apache  
}
```

- Nœud default : si on ne trouve pas la définition précise du nœud

Obligation de définition

Si on définit un nœud, alors il doit y avoir un bloc pour chacun des nœuds.
D'où le nœud default...

- Expressions régulières :
 - `node /^db [1 - 5] \. puppet \. lan $ / { ... }`



Hiera



Puppet with Hiera
Using **hiera** for **variable management**

- Concept d'ENC (external node classifier), présentation de Hiera et langage **YAML**
- Hiérarchie, répertoire hieradata, common.yml et nodes/*
- Fonctions lookup : hiera()



Hiera



- Fusion de dictionnaires : `hiera_hash()` et tableaux : `hiera_array()`
- Paramètres de classe surchargé par Hiera
- Création dynamique de ressources via un hash
- Définition de classes dans hiera
- Fonction `hiera_include`

La forge

- [Forge Puppet](#)
- puppet module list
- puppet module search
- puppet module install





Configuration avancée



- Environnements Puppet : configuration choix
- Liaison avec git
- Gestion des certificats
- PuppetDB : facts et catalogue en cache, reporting