Eines per a l'administració remota de servidors

14 d'abril de 2016

1 Synctool

La missió principal d'aquesta eina és la de sincronitzar fitxers de configuració d'un conjunt de servidors remots en base a un repositori local. A l'igual que pssh utilitza eines com ssh o rsync per a la distribució dels fitxers que s'han de sincronitzar. Un cop sincronitzats en els servidors, synctool permet executar comandes del tipus `service dimoni restart`. Pots, a més, customitzar synctool per a executar més comandes[1].

Per treballar amb synctool hem de fer-ho per ssh sense password. O sigui, del node mestre fins als nodes del clúster ha d'haver autenticació per claus assimètriques. D'igual manera, si volem protegir les claus privades haurem de fer servir l'ssh-agent.

2 Puppet

2.1 Instal·lació

Ens hem de baixar la versió depenent de la nostra distribució:

```
root@master:~# wget https://apt.puppetlabs.com/puppetlabs-release-jessie.deb
root@master:~# dpkg -i puppetlabs-release-jessie.deb
```

La instal·lació del paquet que ens hem baixat ens configura nous repositoris on finalment estàn allotjats els binaris i el codi font de puppet.

```
root@master:/etc/apt/sources.list.d# apt-get install puppetmaster
```

Al final, a part de depèndencies requerides:

```
root@master:/etc/apt/sources.list.d# dpkg -l | grep puppet
ii  puppet-common          3.7.2-4   all   configuration management system
ii  puppetlabs-release     1.0-11    all   ‘‘Package to install Puppet ...’’
ii  puppetmaster           3.7.2-4   all   configuration management, master service
ii  puppetmaster-common    3.7.2-4   all   configuration management, master common files
```

Pel client fem el mateix procès però en comptes d'instal·lar el paquet `puppetmaster` instal·lem `puppet`. Després configurem el fitxer `/etc/hosts` pròpiament per a que tots dos es vegin per el nom de la màquina.

[1] **node** és un host dintre d'un grup de computadores o **cluster**.

2.2 Hello World

Hem de crear un fitxer sota de `/etc/puppet/manifiest` amb extensió `.pp`. Aquest fitxer és un fitxer de codi declaratiu de puppet on definim els nostres recursos. Cada recurs és d'un tipus determinat i té un nom. Això és el mínim que ha de tenir. Veurem que pot tenir propietats amb valors (parell de claus separades per `=>`). El codi té la següent aparença:

```
notify{'Hello world':
}
```

Aquest és un recurs de tipus *notify* Ara el que hem de fer és executar-lo:

```
$ puppet apply hello.pp
Notice: Compiled catalog for master.home in environment production in 0.05 seconds
Notice: Hello world
Notice: /Stage[main]/Main/Notify[Hello world]/message: defined 'message' as 'Hello world'
Notice: Finished catalog run in 0.04 seconds
```

Un altra tipus de recurs és *file* el qual serveix per crear un fitxer amb un determinat contingut. En aquest exemple anem a crear un fitxer a `/tmp` que es diu *demo.txt*. El fitxer s'ha de crear si no existeix i el seu contingut ha de ser 'Hello World':

```
usuario@master:/etc/puppet/manifests$ cat demo.pp
file {'/tmp/demo.txt':
    # si el fitxer no exixteix, que el crei
    ensure => present,
    # contingut del fitxer
    content => 'Hello World',
}
usuario@master:/etc/puppet/manifests$ puppet apply demo.pp
Notice: Compiled catalog for master.home in environment production in 0.21 seconds
Notice: Finished catalog run in 0.03 seconds
usuario@master:/etc/puppet/manifests$ ls /tmp
demo.txt
usuario@master:/etc/puppet/manifests$ cat /tmp/demo.txt
Hello Worldusuario@master:/etc/puppet/manifests$
```

2.3 Test Model Client-Servidor

Anem a fer que un agent apliqui un .pp del servidor. Per defecte, el .pp s'ha de dir *site.pp*. És el manifest per defecte que els clients llegeixen del master server. En arrancar l'agent, si tot va bé el procès és el següent (el primer cop que s'executa l'agent s'estableixen els certificats):

- Agent genera claus i envia al màster la seva clau pública.
- Master li torna la clau signada i la torna al client. Ho farem a mà.
- Agent es baixa .pp i executa la configuració

```
root@client0:~# puppet agent --server master.home --verbose --no-daemonize
 --waitforcert 10
Info: Creating a new SSL key for client0.home
Info: csr_attributes file loading from /etc/puppet/csr_attributes.yaml

Info: Certificate Request fingerprint (SHA256):
  6D:AD:C0:57:4E:82:67:79:0F:25:35:39:CF:C1:35:67:ED:12:6C:27:32:87:43:E1:...
Notice: Did not receive certificate
Notice: Did not receive certificate
Notice: Did not receive certificate
Notice: Did not receive certificate
Notice: Did not receive certificate
Notice: Did not receive certificate
Info: Caching certificate for client0.home
Notice: Starting Puppet client version 3.7.2
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for client0.home
Info: Applying configuration version '1457863005'
Notice: /Stage[main]/Main/File[/tmp/demo2.txt]/ensure: created
Notice: Finished catalog run in 0.12 seconds
^CNotice: Caught INT; calling stop
root@client0:~# ls /tmp
demo2.txt
root@client0:~#
```

En el master hem de tenir el *site.pp*:

```
file {'/tmp/demo2.txt':
  # si el fitxer no exixteix, que el crei
  ensure => present,
  # contingut del fitxer
  content => 'Hello World',
  owner => 'root',
  mode => 777,
}
```

En el client, l'avís `Notice: Did not receive certificate` vol dir que no ha rebut el seu certificat signat per el master. Per fer-ho, des del master:

```
root@master:/etc/puppet/manifests# puppet cert sign client0.home
```

Per veure els certificats dels clients:

```
root@master:/etc/puppet/manifests# puppet cert list
```

Al final, en aquest exemple, se'ns haurà creat un fitxer a tmp amb propietari i permisos com els configurats en el *site.pp*.

Problemes

1. que no coincideixin els noms de les màquines amb els noms que s'han posat en els certificats. Podem tocar el paràmetre certname a la configuració del master, o afegir en el `/etc/hosts` com a nom a les màquines el que hem posat en el certificat i en la comanda de l'agent,

   `--server <posem el nom que surt al certificat>`

2. que no s'envii el certificat des de l'agent. Tornem a començar: des del master, esborrem el certificat de l'agent i, des del client, igual:

   - Master:

     `root@master:/etc/puppet/manifests# puppet cert clean client0.home`

   - client:

     `root@client0:~# find /var/lib/puppet/ssl -name client0.home.pem -delete`

     I tornem a executar la comanda en l'agent.

3. en el client, l'agent no s'arrenca donant un error. Amb això, hem de tenir clar les següents coses:

   (a) En instal·lar *puppet* en el client tenim un servei *puppet agent* iniciat per defecte.

   (b) Si hi ha un *agent puppet* iniciat no es pot iniciar cap altre, per la qual cosa, hem de parar el que hi hagi arrancat i hem de treure el bloqueig de que s'inicii qualsevol agent amb la comanda `puppet agent --enable`

   (c) Un cop desbloquejat, ja podem executar la comanda per rebre la configuració del master.

2.4 Nodes

Els nodes permeten a un servidor *Puppet* tenir diferents recursos diferenciats per clients. Els nodes es defineixen en el manifest *site.pp*. Exemple del fitxer *site.pp*:

```
# el nom que li donem al node correspon al nom de la màquina
node client0 {
  package {'nginx':
    # ens hem d'assegurar que el paquet estigui instal·lat
    # si no ho està, s'instal·la
    ensure => installed,
  }
}

# nodes no definits explícitament
node default {
  notify {'Default node':
  }
}
```

Agent *client0*, aleshores, en recuperar la configuració, haurà instal·lat *nginx* (abans no hi era):

```
root@client0:/home/usuario# puppet agent --server master.home --verbose
                                          --no-daemonize --waitforcert 10
Notice: Starting Puppet client version 3.7.2
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for client0.home
Info: Applying configuration version '1458290609'
Notice: /Stage[main]/Main/Node[client0]/Package[nginx]/ensure:
                          ensure changed 'purged' to 'present'
Notice: Finished catalog run in 60.09 seconds
^CNotice: Caught INT; calling stop
root@client0:/home/usuario# dpkg -l nginx
ii  nginx          1.6.2-5+deb8 all          small, powerful, scalable web/prox
```

Si a la propietat *ensure* li donem el valor *absent*, succeirà el contrari.

2.5 Recursos

- **notify**
- **file**
- **package**: defineix un paquet de software a instal·lar o desinstal·lar en un node.
- **service**: Possibles propietats:
  - `ensure`, que pot prendre els valors *stopped* o *running*
  - `enable`, per configurar el seu arranc automátic, que pot prendre els valors *true* o *false*.
  - *require*. Si volem fer esmena d'una altre recurs. Veure la següent subsecció sobre dependències de recursos del recurs *service*
- **exec**: per executar comandes.
- **group**: gestionar grups *Unix like*.
- **user**: gestionar usuaris *Unix like*.

2.6 Dependències entre recursos en el recurs *service*

Si tenim un recurs de tipus *service* el que volem, per exemple, és que estigui iniciat però sempre i quan tinguem el paquet instal·lat, o que es notifiqui cada canvi en el fitxer de configuració d'aquest servei ja que no es reflectiràn aquests canvis si no es fa, com a mínim, un *reload* del servei.

Qualsevol crida que fem a un recurs des d'un altre recurs, com a regla de sitaxi, hem de començar a nomenar el tipus de recurs amb lletra majúscula i, com si fos un array, entre corxets i cometes, especificar el nom del recurs al que fem referència:

```
root@master:/etc/puppet/manifests# cat nodes.pp
node 'web1'{
        package {'nginx':
                ensure => installed,
        }
        service {'nginx':
                ensure => running,
                enable => true,
                require => Package['nginx'],
        }
}

node default{
        notify {'Default node':
        }
}
root@master:/etc/puppet/manifests#
```

2.7 Modulització: Classes i definició de recursos

Els mòduls es creen sota el directori `/etc/puppet/modules`. Cada mòdul es crea sota un directori dintre del directori modules i la seva estructura bàsica porta els següents directoris: `manifests`, `files` i `templates`. Al directori `manifests` hi ha d'haver un fitxer que és el que s'executa primer quan es fa referència a un mòdul, `init.pp`. Per incloure un mòdul en el catàlog d'un node:

```
node 'web1'{
        include apache
}

node default{
        notify {'Default node':
        }
}
```

El directori *files* serveix per contenir els fitxers de configuració dels serveis, etc.

La diferència entre classes i definicions és que les classes les inclouem en els llocs on les volem fer servir i les definicions simplement les cridem. Vindria a ser com la diferència entre *includes* i *funcions* en un llenguatge estructurat.

2.7.1 Classes

Les classes venen a ser com includes. Són blocs de codi *puppet*. No afegeixen cap recurs al catàlog de cap node. Per a que sí formin part d'un catàlog, han de ser declarades. Poden contenir subclasses, les quals es criden amb ::. Exemple de classe:

```
#dintre de class definim els nostres recursos:
class base{
  file {'/etc/passwd':
    owner => 'root',
    group => 'root',
    mode => '0644',
  }
  file {'/etc/shadow':
    owner => 'root',
    group => 'root',
    mode => '0440',
  }
```

```
}
```

Un altre exemple de classe que, a més, té dependéncies:

```
class apache{
        package {'apache2':
                        ensure => installed,
        }
        service {'apache2':
                        ensure => running,
                        require => Package['apache2']
        }
        file {'/etc/apache2/conf/apache2.conf':
                        ensure => present,
                        require => Package['apache2']
                        notify => Service['apache2']
        }

}
```

Subclasses

Com es pot resoldre el problema de les dependències amb subclasses? Podem crear, en el mateix directori on hem creat el fitxer *init.pp* al mòdul apache (`/etc/puppet/modules/apache/manifests`) el fitxer *install.pp* per definir la subclasse *install* que representa el recurs package:

```
class apache::install {
  package {
    ensure => installed,
  }
}
```

I, de la mateixa manera, el fitxer *service.pp* i *config.pp*:

```
class apache::service{
        service {'apache2':
                ensure => running,
                require => [Class["apache::install"]]
}
```

```
class apache::config{
        file {'/etc/apache2/conf/apache2.conf':
                ensure => present,
                require => [Class["apache::install"]],
                notify => [Class["apache::service"]]
        }
}
```

Fixar-se que quan fem referència a un element, un recurs, una classe, etc, ho fem amb la primera lletra en majúscules.

```
class apache{
        include apache::install,
                apache::service,
                apache::config

}
```

Exemple obsolet. S'han de treure les comes i potser cal ficar *include* en les tres línies

2.7.2 Definitions

A diferència de les classes, una vegada creada una definició, ja actua com un tipus de recurs. Es pot parametritzar i és avaluada cada vegada en funció d'aquests paràmetres.

Anem a veure un exemple seguint l'exemple d'abans del mòdul d'apache. Crearem la definició que serà per fer servir recursos que faràn referència a virtualhosts. Primer de tot creem el fitxer *vhost.pp* dintre del mateix directori *manifests* on hem creat el fitxer init del módul apache:

```
#entre parèntesi declarem els arguments.
#hi han dos arguments que sempre hi són, que són $title i $name.
define apache::vhost($port, $docroot){
  ‘‘/etc/apache2/conf/$name’’:
    # fisicament aquest fitxer font haurà d'estar al directory files
    # dintre del mòdul.
    source => ‘‘puppet:///modules/apache/apache2.conf’’,
    owner => ‘‘apache‘‘,
    group => ’’apache‘‘,
    mode => 755,
    require => [Class[’’apache::install‘‘]],
    notify => [Class[’’apache::service‘‘]]
}
```

Després la fem servir a *install.pp* d'apache:

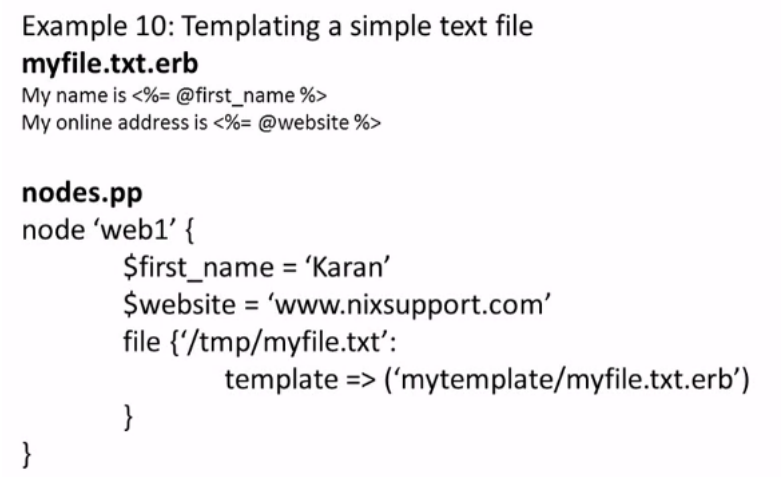
```
class apache {
  include apache::install,
  include apache::service,
  include apache::config,

  # el primer argument \'es el $name, el nom del recurs.
  # faig servir un recurs de tipus apache\:\:vhost que te per nom apache2.conf
  # t\'e per paràmetres $port i $docroot
  apache::vhost {‘‘apache2.conf’’:
    $port => 80,
    $docroot => ‘‘/var/www/html/mysite’’,
  }
}
```

2.8 Ordre d'avaluació entre els recursos

Per defecte serà en l'ordre en la que s'han definit aquests recursos. Sinó, es pot control·lar o alterar l'ordre gràcies a les propietats *require* i *before* dintre dels recursos i fent referència (o sigui, donant com a valor d'aquestes propietats) als recursos que li acompanyen.

2.9 Templates

Les plantilles es fan servir per crear fitxers amb valors dinàmics que passem als clients o agents. Es creen sota els mòduls, en el directori *templates* i poden fer ús de variables, expressions, llistats, etc. que es calculen en temps d'execució. Les plantilles van dintre dels directoris dels mòduls, en els subdirectoris *templates*. Per fer ús d'un template s'ha de cridar la funció *templates*. Anem veure un exemple de com crear i fer servir plantilles. Anem a crear una classe que es diu *linuxbase* la qual farem servir per definir els recursos de client_0. Aquesta classe



tindrà una subclasse, que és a la que cridarà directament, *tmp* que definirà un recurs de tipus file amb un contingut que estarà definit per una plantilla:

1. Creem els subdirectoris necessaris a `/etc/puppet/modules`: `linuxbase`, `linuxbase/manifests` i `linuxbase/templates`.

2. Definim la plantilla a `linuxbase/templates`. Creen un fitxer que es diu myfile.txt.erb. Les plantilles s'han de crear amb l'extensió `.erb` (llenguatge *erb*, *ruby standard library*):

   ```
   my name is <%= @first_name %>
   my website is <%= @website %>
   ```

3. Fem ús de la plantilla a la subclasse `linuxbase::tmp`. Per això definim la subclasse en `linuxbase/manifests/tmp.pp`:

   ```
   class linuxbase::tmp{
     $first_name = ''Juan''
     $website = ''www.aguileraj.com''
   # la plantilla agafa totes les variables definides en aquest scope
     file {'/tmp/file.txt':
       content => template (''linuxbase/myfile.txt.erb''),
     }
   }
   ```

4. I ara definim la classe `linuxbase` que anirà dintre del fitxer `linuxbase/manifests/init.pp`:

   ```
   class linuxbase {
     include linuxbase::tmp
   }
   ```

5. en executar puppet en mode client en client0 s'hauria d'haver creat el fitxer a `/tmp/file.txt` amb el contingut:

   ```
   my name is Juan
   my website is www.aguileraj.com
   ```

2.10 Facts

Variables predefinides del sistema i amb les quals podem també crear estructures de control de fluxe com *if* o *switch* o *if ... else*. Per veure la llista de *facts* definides en el sistema, `facter`. Anem a veure un exemple en el que, depenent de la variable d'entorn o *facter* `$osfamily`, instal·lem el paquet *apache2* si el sistema és *Debian* o el paquet *httpd* si és *Red Hat*. Ho farem com en l'exemple anterior, a *templates*, creant una subclasse de la classe *linuxbase* que es digui *install*. Fitxer `/etc/puppet/modules/linuxbase/manifests/install.pp`:

```
class linuxbase::install {
  if $osfamily ==  'Debian' {
    package {'apache2':
      ensure => installed,
    }
  } elsif $osfamily ==  'RedHat' {
    package {'httpd':
      ensure => installed,
    }
  }
}
```

2.11 Gestionar usuaris i grups Unix like des de puppet

Els recursos associats són *group* i *user*. En el següent exemple podem visualitzar les propietats que poden tenir:

```
group {'db-admins':
  ensure => present,
gid => 4002,
}
user {
  uid => 2014,
home => '/home/karan'
# si la següent propietat no està a true,
# no podrem crear la home o gestionar la
# home adecuadament (amb els permisos correctes, per exemple)
  managehome => true,
gid => 1000,
shell => '/bin/bash',
groups => ['db-admins'],
}
```

2.12 Com executar comandes Linux amb puppet

En aquest cas el recurs és *exec* les propietats que pot portar:

```
exec {'Download Putty':
   command => 'wget http://tartarus.org/~simon/putty-
snapshots/x86/putty.zip',
    cwd => '/opt',
    path => '/bin:/usr/bin',
    creates => '/opt/putty.zip',
}
```

2.13 Síntesi

2.13.1 Instal·lació de Lamp server en un servidor amb puppet

Instal·lació manual de Lamp en un servidor

1. Actualitzar les llistes dels repositoris del sistema

   ```
   # apt-get update
   ```

2. Ens instal·lem *apache2*

   ```
   # apt-get install apache2
   ```

3. Ara ens instal·lem el paquet *php5*. Instal·la automàticament altres paquets.

   ```
   # apt-get install php5
   ```

4. *php5-mysql*. Requereix *php5*. Instal·la automàticament altres paquets.

   ```
   # apt-get install php5-mysql
   ```

5. Reiniciem apache2.

6. instal·lem el paquet *mysql-server* i li donem un password a root (*mysql*)

7. Idem amb libmysqlclient-dev. Suposo que serà necessari tenir prèviament mysql-server.

2.14 TODOs

```
}
file {'/etc/apache2/conf.d/apache2.conf':
        ensure => present,
        notify => Service['httpd'],
}
```

Per exemple, el que fa aquesta propietat *notify* és avisar de qualsevol canvi en el fitxer de configuració per a que el servei HTTP reinicii.

Per a què serveix la comanda puppet apply -e 'comanda'

3 bcfg2

[2]

4 Sobre aquest document

Referències

[1] `http://walterdejong.github.io/synctool/`

[2] `http://docs.bcfg2.org/index.html`

[3] `https://en.wikipedia.org/wiki/Comparison_of_open-source_configuration_management_software`

[4] `https://forge.puppetlabs.com/`