Creació d'una plataforma LAMP utilitzant Puppet

Memòria final

Carles Amigó Martínez
Enginyeria tècnica d'informàtica de gestió
Universitat Oberta de Catalunya
Treball de fi de carrera

Índex

Descripció del TFC

Introducció

Avui en dia la plataforma coneguda com a LAMP, s'ha convertit en el líder indiscutible de la immensa majoria dels projectes web. D'aquesta manera, el software lliure s'ha consolidat com la primera alternativa a tenir en compte a l'hora de planificar qualsevol tipus de projecte basat en web.

De totes maneres, si un projecte té èxit, és inevitable que creixi al cap de poc temps, fent necessari un número de màquines cada cop més gran i en conseqüència cada cop més feina per administrar-les.

El projecte de codi obert Puppet, neix precisament per intentar solucionar aquesta problemàtica, de manera que un cop ben configurat i amb una bona programació, el cost de mantenir una granja de centenars o milers de servidors no és gaire més elevat que el de mantenir-ne desenes.

Sobre LAMP

LAMP és un acrònim creat per descriure la plataforma estàndard, que utilitza eines de codi obert, sobre la que funcionen la majoria d'aplicacions web d'avui en dia.

Tot i que ara mateix el concepte és molt més ampli, aquest terme definia la utilització del següent programari:

- **L de Linux.** Tot i que també es poden fer servir altres sistemes operatius basats en UNIX com Solaris, FreeBSD, OpenBSD, MacOS o fins i tot altres sistemes com Microsoft Windows

- **A d'Apache.** Tot i que recentment estan apareixent altres servidors web com Nginx (http://nginx.net/) o Lighttpd (http://www.lighttpd.net/), cap d'ells arriba al compromís d'Apache entre flexibilitat i rendiment

- **M de MySQL.** MySQL és escollit en la majoria de projectes web per la seva simplicitat, lleugeresa i rapidesa en contraposició d'altres bases de dades a priori més potents com Oracle o PostgreSQL

- **P de PHP.** Llenguatge creat originàriament per a la web. De totes maneres, també es poden fer servir altres llenguatges com Perl o Python.

Sobre Puppet

Puppet va néixer per ajudar a administrar granges de servidors de manera fàcil fent servir un llenguatge declaratiu que descriu d'una manera senzilla i clara l'estat en que han d'estar els servidors que gestiona.

Tot i que quan va ser creat ja existien eines similars per assolir aquesta tasca[1], cap d'elles no era lo suficientment flexible, ràpida i àgil per adaptar-se al cambiant món de l'administració de sistemes informàtics.

Per poder gestionar un seguit de servidors amb Puppet, es necessiten les següents parts:

1. Un servidor (o puppetmaster) que contindrà tot el codi i mitjançant una API REST[2] els clients es comunicarà amb ell obtenint tota la informació necessària per poder-se configurar.

2. Un client (o puppetclient), instal·lat a tots els servidors, i que periòdicament farà una consulta al puppetmaster per obtenir el estat desitjat en que ha d'estar configurat.

Avui en dia Puppet és utilitzat per grans empreses com Google, Apple, Zinga o la mateixa Wikipedia[3] i el seu us és cada cop més generalitzat.

1 Per exemple, CFEngine http://cfengine.com/

2 REST (Representational State Transfer) és una arquitectura de programari pensada per sistemes distribuïts basats en hipermèdia, com ara el web.

3 Wikipedia va publicar fa poc tot el seu codi de Puppet amb el que gestiona totes les seves màquines: http://blog.wikimedia.org/2011/09/19/ever-wondered-how-the-wikimedia-servers-are-configured.

Objectius generals i específics

Els objectius a complir amb aquest projecte seran:

1. Documentació sobre l'us de Puppet en la plataforma que ens pertoca amb els seus avantatges i inconvenients.

2. Instal·lació i configuració dels servidors que formaran part de la plataforma:

    1. Servidor web

    2. Servidor MySQL

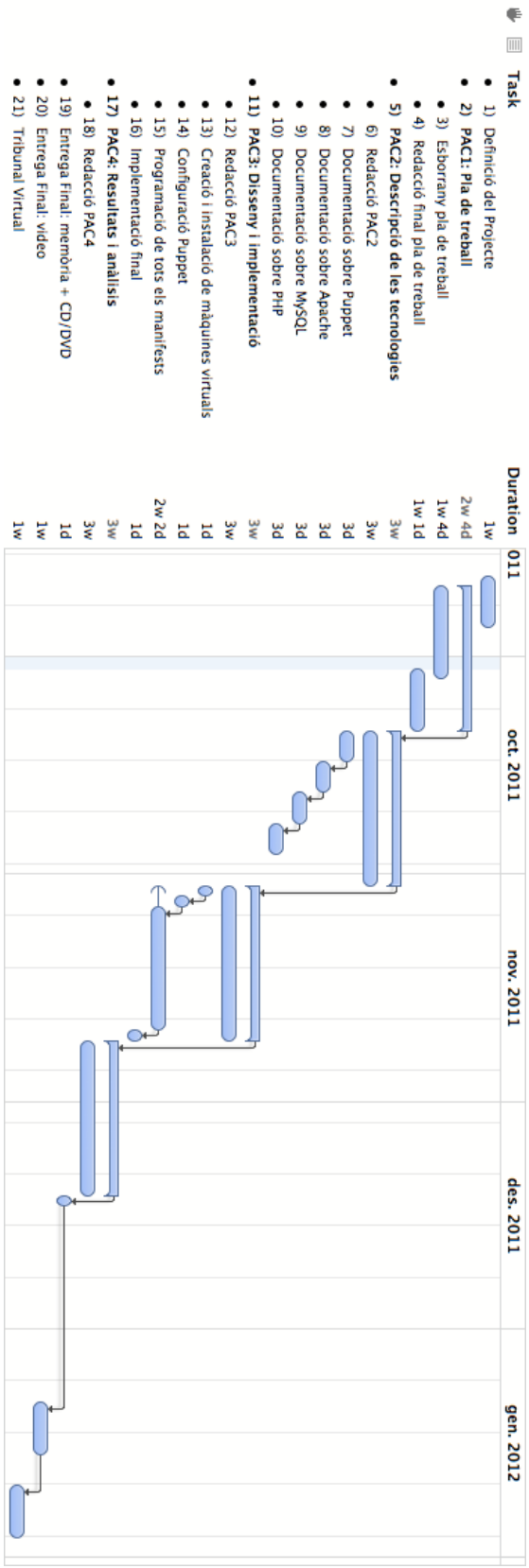
    3. Servidor Puppet

3. Programació de tots el codi de Puppet per poder administrar les màquines que formaran part de la plataforma.

4. Documentar el codi amb la eina Puppetdoc[4] per poder generar documentació automàtica. Aquesta documentació es farà en Anglès per poder publicar el codi a Puppetforge[5] si fos escaient.

4 Sistema intern de documentació que incorpora Puppet similar a javadoc
5 Plataforma lliure de distribució de mòduls de Puppet

Planificació amb fites i temporalització



Tasca	**Inici**	**Fi**	**Durada**
0) TFC	21/09/11	27/01/12	18w 3d
1) Definició del Projecte	21/09/11	27/09/11	1w
2) PAC1: Pla de treball	22/09/11	11/10/11	2w 4d
3) Esborrany pla de treball	22/09/11	04/10/11	1w 4d
4) Redacció final pla de treball	04/10/11	11/10/11	1w 1d
5) PAC2: Descripció de les tecnologies	12/10/11	01/11/11	3w
6) Redacció PAC2	12/10/11	01/11/11	3w
7) Documentació sobre Puppet	12/10/11	14/10/11	3d
8) Documentació sobre Apache	17/10/11	19/10/11	3d
9) Documentació sobre MySQL	20/10/11	24/10/11	3d
10) Documentació sobre PHP	25/10/11	27/10/11	3d
11) PAC3: Disseny i implementació	02/11/11	22/11/11	3w
12) Redacció PAC3	02/11/11	22/11/11	3w
13) Creació i instalació de màquines virtuals	02/11/11	02/11/11	1d
14) Configuració Puppet	03/11/11	03/11/11	1d
15) Programació de tots els manifests	04/11/11	21/11/11	2w 2d
16) Implementació final	22/11/11	22/11/11	1d
17) PAC4: Resultats i anàlisis	23/11/11	13/12/11	3w
18) Redacció PAC4	23/11/11	13/12/11	3w
19) Entrega Final: memòria + CD/DVD	14/12/11	14/12/11	1d
20) Entrega Final: video	11/01/12	17/01/12	1w
21) Tribunal Virtual	23/01/12	27/01/12	1w

Material

Per poder dur a terme aquest projecte, serà necessari el següent material:

Programari

CentOS

- http://www.centos.org/

Sistema operatiu basat en RedHat. En concret farem servir la versió 6, ja que es tracta de la última que hi ha al mercat i que incorpora les darreres versions del programari que farem servir.

En principi qualsevol distribució seria vàlida per a aquest projecte, però hem escollit CentOS per ser totalment lliure i gratuïta i per la seva gran extensió en el mon laboral, havent-se convertit en un estàndard "de facto".

Puppet

- http://puppetlabs.com/

Puppet és un programa de codi obert que s'encarrega d'administrar la configuració de servidors. Fa servir un llenguatge descriptiu per definir l'estat ideal en que han d'estar totes les màquines.

Mitjançant una arquitectura client-servidor Puppet s'encarrega de comparar de manera periòdica l'estat actual de cada una de les màquines amb l'estat ideal. En el cas de que hi hagi diferències, executarà les comandes necessàries i modificarà els fitxers de configuració pertinents per assegurar que l'estat actual i l'estat ideal son exactament iguals.

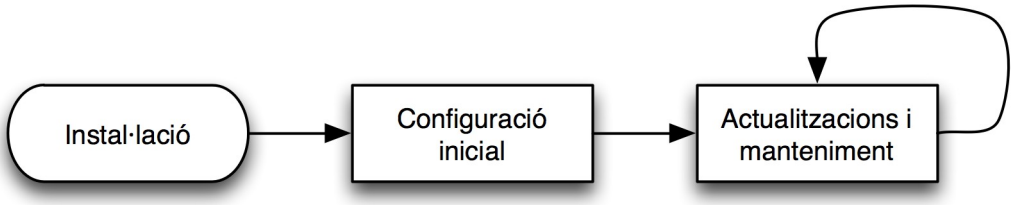
Puppet està escrit en el llenguatge de programació Ruby, i el seu llenguatge descriptiu, tot i ser diferent i molt més senzill, ha heretat bastants característiques d'aquest. Està disponible per a la gran majoria de sistemes operatius basats en Unix (sobretot Linux) i fins i tot té suport bàsic per a plataformes Windows.

Cicle de vida d'un sistema

Des que es posa en marxa un sistema fins que es retira del seu funcionament, podem dir que segueix un cicle de vida específic i que es repeteix per a qualsevol altre sistema que vulguem posar en funcionament, encara que sigui per a realitzar una altra tasca.

Aquest cicle de vida es podria resumir en tres etapes:

1. Instal·lació
2. Configuració inicial
3. Actualitzacions i manteniment



Instal·lació

La instal·lació d'un sistema comporta des de la compra del maquinari i el seu muntatge i configuració (per exemple la configuració del RAID o el número de processadors o la quantitat de memòria) fins a la instal·lació del sistema operatiu.

Automatitzar les tasques relacionades amb el maquinari està fora de l'abast d'aquest projecte. No obstant, la automatització de la instal·lació del sistema operatiu pot ser realitzada amb relativa facilitat mitjançant programari específic. Algun d'aquest programari és específic de la plataforma en la que estiguem treballant.

Per exemple, en els sistemes basats en RedHat, es pot fer servir kickstart, pero altres sistemes tenen eines equivalents.

Per realitzar una instal·lació automatitzada mitjançant kickstart, hem de configurar la bios perquè arranqui un petit sistema operatiu mitjançant la xarxa (amb un protocol anomenat PXE).

Un cop arrancat aquest petit sistema operatiu, llançarà el programa de instal·lació, que intentarà llegir un fitxer a on haurem especificat prèviament tots els paràmetres de configuració del sistema a instal·lar.

Aquest fitxer, és l'anomenat "kickstart" i tot i no tractar-se d'un script pròpiament dit, consta d'unes comandes ben definides amb les que podem especificar qualsevol paràmetre de configuració, des de tota la part de xarxa fins al particionat del disc o els paquets que voldrem instal·lar.

Exemple de fitxer kickstart per a una instal·lació bàsica:

```
# Llenguatge del sistema
lang en_US.UTF-8

# Quins mòduls de llenguatge instal·larem
langsupport --default=en_US.UTF-8 en_US.UTF-8

# Teclat del sistema
keyboard es

# Configurarem el ratolí
mouse

# Configurem la zona horaria
timezone --utc Europe/Madrid

# Contrasenya de root
rootpw --iscrypted $1$LZXhyRqY$m0tAX6fn2ZeOyMCXT4lvW1

# Realitzarem un reinici després de la instal·lació
reboot

# Farem servir la instal·lació en mode text
text

# Instal·larem el sistema operatiu en comptes de fer una actualització
install

# Agafarem els paquets de un servidor NFS
nfs --server=172.20.1.1  --dir=/export/RedHat

# Configurem l'arranc
bootloader –location=mbr

# Borrarem el Master Boot Record
zerombr yes

# També borrarem totes les particions existents
clearpart --all –initlabel

# Informació de les particions
part /boot --fstype ext3 --size 100
part swap --recommended
part / --fstype ext3 --size 1 –grow

# Configuració de la autentificació del sistema
auth  --useshadow  --enablemd5

# Configuració de xarxa
```

```
network --bootproto=static –ip=192.168.1.2 --netmask=255.255.255.0 –gateway=192.168.1.1 –nameserver=8.8.8.8 --device=eth0

# Desactivem el firewall i selinux
firewall --disabled
selinux –disabled

# No configurarem les X windows
skipx

# Selecció dels paquets a instal·lar
%packages --resolvedeps
@ editors
@ text-internet
@ admin-tools
@ system-tools
-psgml
-dovecot
-spamassassin
-emacs
-emacspeak
-emacs-leim
-cadaver
-fetchmail
-mutt
e2fsprogs
-postfix
-sendmail-cf
net-snmp
openssh-server
openssh-clients
-cups
-bluez-libs
-sendmail
-xinetd

# un cop acabada la instal·lació, executarem aquestes comandes
%post
/usr/bin/yum -y install puppet
/usr/bin/yum -y update

# desactivem el mouse de consola
/etc/init.d/gpm stop
/sbin/chkconfig --del gpm
```

Realment, la instal·lació inicial del sistema no ha de ser massa complicada, ja que tot i ser un sistema flexible, kickstart no permet tanta flexibilitat com aporta puppet, i un cop tinguem instal·lat el sistema operatiu base amb els mínims paquets necessaris, podem

fer servir puppet per realitzar la resta de tasques de configuració.

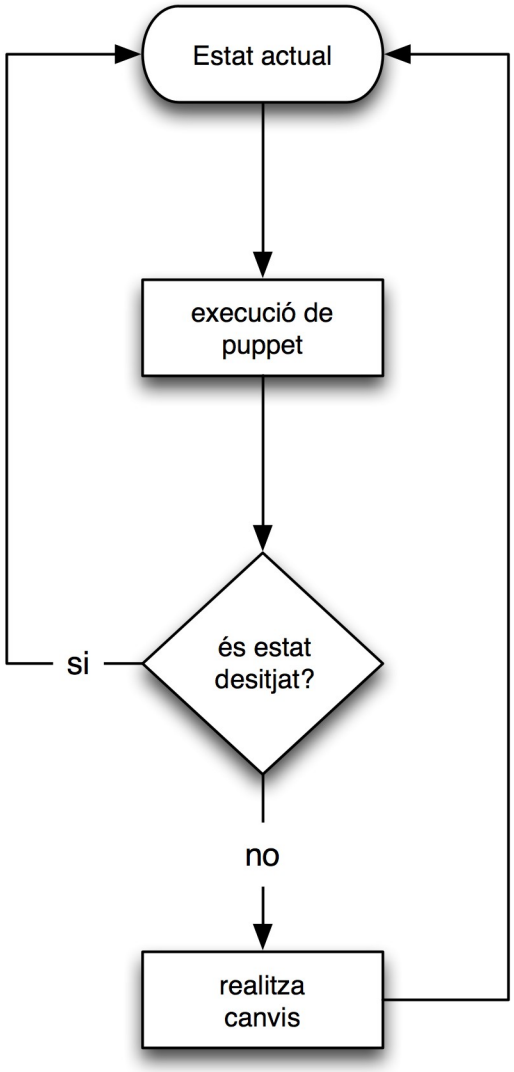
Configuració inicial

Un cop tenim un sistema operatiu mínim, el nostre maquinari ja pot arrancar i executar la per primer cop puppet. Aquesta tasca també es pot arribar a automatitzar mitjançant el kickstart que hem creat abans.

En aquesta primera execució, puppet s'encarregarà de comprovar que l'estat actual de la màquina es correspon amb l'estat desitjat, tal i com l'hem especificat nosaltres en els nostres fitxers de "manifests" en el llenguatge descriptiu de puppet.

En el cas que l'estat actual no correspongui amb l'estat desitjat, puppet executarà totes les comandes i modificacions necessàries per arribar a l'estat que nosaltres hem especificat.

El cicle d'execució de puppet es podria definir en el següent diagrama de flux:

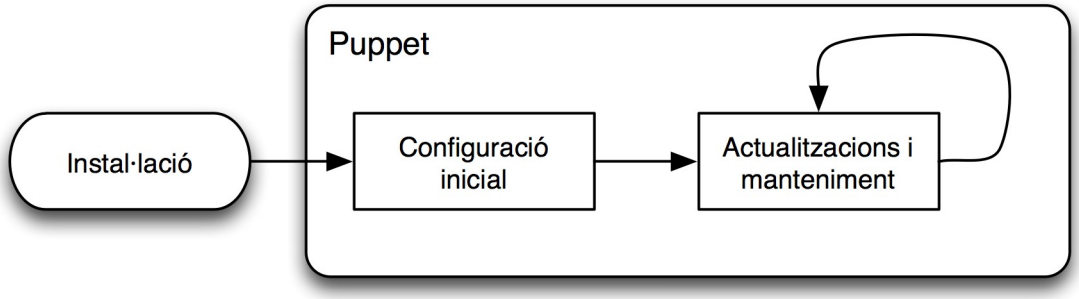


Actualitzacions i manteniment

Un cop hem arribat al nostre estat desitjat però, puppet seguirà comprovant de manera periòdica que l'estat actual segueix corresponent amb l'estat desitjat.

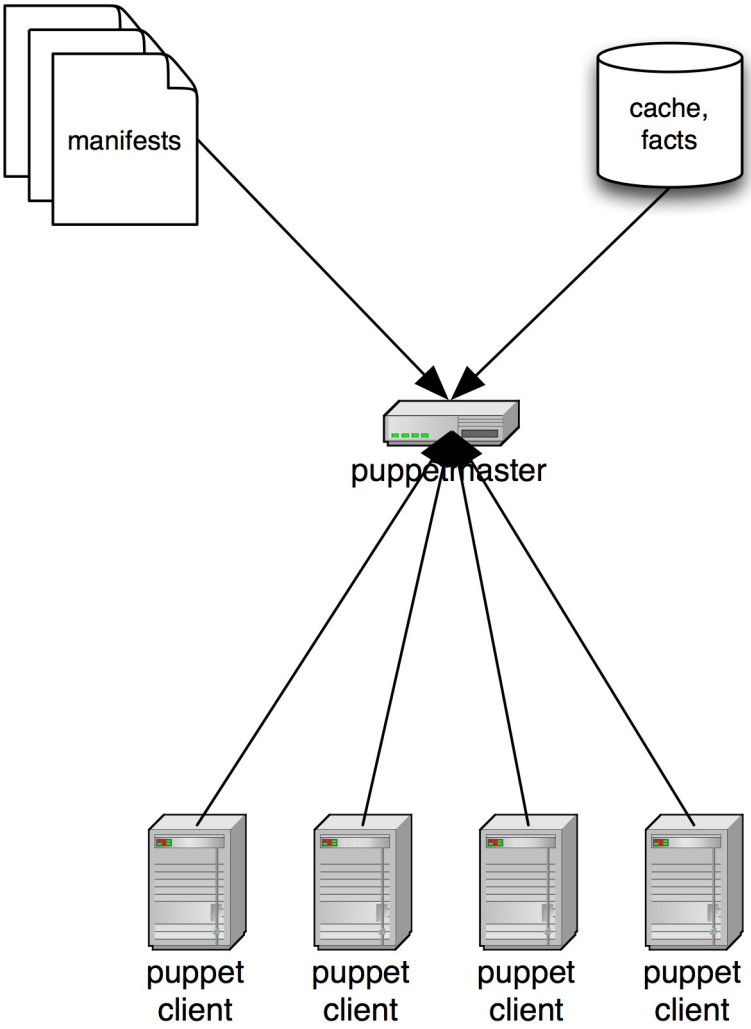
Per exemple, es pot donar el cas que factors externs com per exemple una interacció humana modifiquin l'estat actual de la màquina i el deixin en un estat no desitjat.

Un altre cas possible es podria donar si per alguna raó l'estat desitjat canvia i aixi ho manifestem als nostres fitxers de "manifests" de puppet. Per exemple, es podria donar el cas que per especificacions noves en el nostre projecte, necessitéssim instal·lar i configurar un nou programari.



Arquitectura interna de Puppet

Puppet funciona en una arquitectura client-servidor:



Puppet client

El client de puppet és un servei que s'està executant de manera constant a tots els nostres sistemes. Periòdicament, en un interval de normalment cada 30 minuts, es conectarà amb el nostre servidor de Puppet per demanar-li el catàleg que especifica l'estat desitjat en que ha d'estar el nostre sistema.

Prèviament però, el nostre client de puppet li haurà enviat al servidor el que anomenem "facts" sobre aquest sistema. Un fact no és més que una variable a on s'especifica una característica única d'aquest servidor. Amb el client de puppet hem d'instal·lar un altre programa anomenat "facter" que serà l'encarregat de recollir tots aquests "facts" i també ens permetrà visualitzar-los des de la linea de comandes.

Tots aquests "facts" els podrem fer servir en els nostres manifests a l'hora de definir l'estat desitjat dels nostres sistemes ja que puppet els incorpora com a variables. Com que cada "fact" acostuma a ser diferent en cada un dels nostres sistemes, son de gran utilitat per poder escriure "manifests" diferents depenent de quin sistema es tracti.

Per defecte, amb facter venen bastants "facts" de gran utilitat, però en cas de necessitat podem crear els nostres propis, ja que és fàcilment extensible mitjançant el llenguatge de programació Ruby.

Alguns dels "facts" que acostumen a ser més útils podrien ser aquests:

```
architecture => i386
domain => fr3nd.lan
facterversion => 1.6.2
fqdn => tfc-web.fr3nd.lan
hardwareisa => i686
hardwaremodel => i686
hostname => tfc-web
id => root
interfaces => eth0,lo
ipaddress => 192.168.1.101
ipaddress_eth0 => 192.168.1.101
ipaddress_lo => 127.0.0.1
is_virtual => false
kernel => Linux
kernelmajversion => 2.6
kernelrelease => 2.6.32-71.el6.i686
kernelversion => 2.6.32
macaddress => 08:00:27:37:78:F1
macaddress_eth0 => 08:00:27:37:78:F1
memoryfree => 450.09 MB
memorysize => 499.63 MB
netmask => 255.255.255.0
netmask_eth0 => 255.255.255.0
netmask_lo => 255.0.0.0
```

```
network_eth0 => 192.168.1.0
network_lo => 127.0.0.0
operatingsystem => CentOS
operatingsystemrelease => 6.0
osfamily => RedHat
path                                                                    =>
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
physicalprocessorcount => 1
processor0 => Intel(R) Pentium(R) 4 CPU 3.00GHz
processorcount => 1
puppetversion => 2.6.12
swapfree => 1023.99 MB
swapsize => 1023.99 MB
timezone => CET
uniqueid => a8c06501
uptime => 0:02 hours
uptime_days => 0
uptime_hours => 0
uptime_seconds => 134
virtual => physical
```

El client un cop ha rebut el catàleg del servidor, s'encarregarà de comparar l'estat desitjat amb l'estat actual del sistema. Com que puppet és multi-plataforma i pot executar-se en una gran varietat de sistemes operatius (tant en basats en unix com en Windows), molts cops la comprovació d'un cert recurs pot haver de fer-se de maneres molt diferents.

Per exemple, per verificar l'existència d'un servei i que està executant-se constantment, es farà de manera completament diferent en sistemes basats en RedHat que en sistemes basats en Debian o sobretot en sistemes basats en Windows.

Per poder superar aquest problema, puppet delega la comprovació d'aquests recursos als anomenats "providers", que seràn els encarregats de realitzar una tasca o una altra depenent del sistema operatiu en el que s'estàn executant.

Hi ha "providers" per a una gran quantitat de sistemes ja creats i en la gran majoria dels casos nosaltres no ens hem de preocupar de res ja que puppet detectarà automàticament quin ha de fer servir en cada moment. De totes maneres, també podem especificar manualment quin "provider" volem fer servir en cada moment en cas de necessitat.

Puppet server

El servidor de puppet tindrà el servei de puppet executant-se contínuament escoltant

peticions al port 8140, que serà per on entraran totes les connexions dels nostres clients.

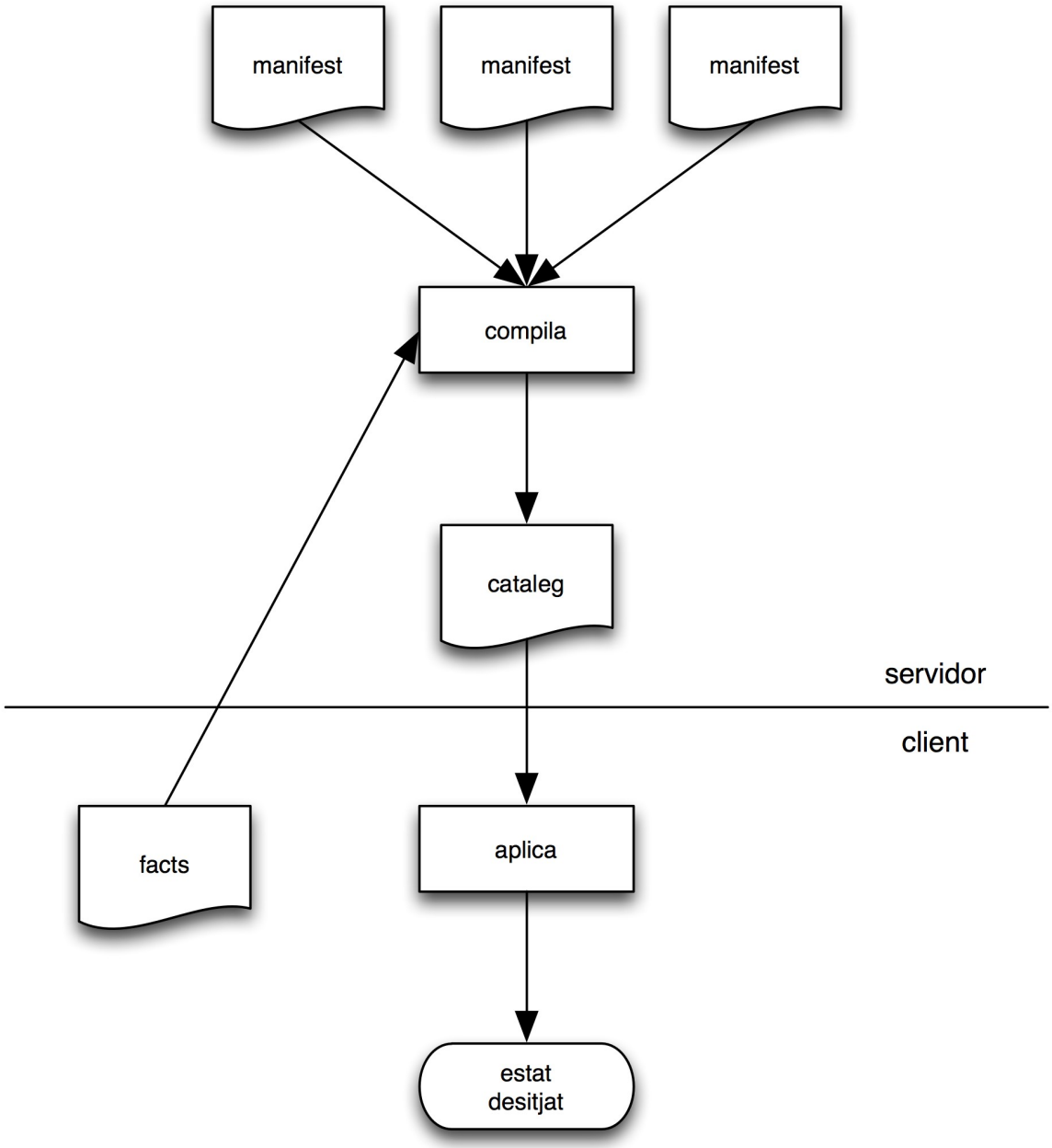
Per a la comunicació es fa servir un protocol REST[6] sobre un canal encriptat mitjançant SSL. També es verifica el certificat del client per a una major seguretat, de manera que és molt difícil "impersonar" un servidor, ja que prèviament haurem d'haver signat el certificat del client amb la nostra pròpia CA.

Al connectar per primer cop, el client enviarà tots els facts disponibles al servidor, i un cop s'han rebut, el servidor començarà la compilació del catàleg. Aquesta compilació pot trigar des de uns segons a uns varis minuts, depenent de la complexitat i de la potència del nostre servidor.

Aquest catàleg no és més que un fitxer en format YAML[7] comprimit en el que apareixen tots els recursos que hem especificat en els nostres "manifests" a l'hora de dissenyar l'estat desitjat del nostre sistema. Un cop el client ha rebut el catàleg, començarà a comparar-lo amb l'estat actual i a realitzar els canvis pertinents en cas de necessitat.

6 Representational State Transfer. És un protocol basat en http

7 Format de serialització de dades inspirat en XML però dissenyat per facilitar la seva lectura per a un humà



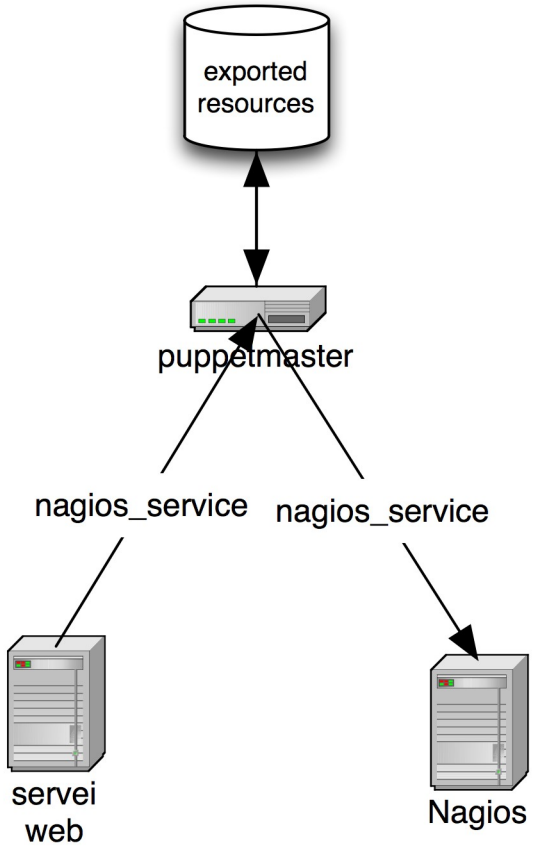
Una altra característica una mica més avançada del servidor de puppet son els anomenats "exported resources". Un "exported resource" no és més que un mètode per poder crear recursos des de nodes diferents. És un concepte una mica complex en que la millor manera d'explicar-lo és mitjançant un exemple.

Imaginem que volem monitoritzar mitjançant Nagios[8] els serveis que estem configurant en els nostres sistemes. Podríem per una banda configurar els serveis necessaris (per exemple un servei web) i per l'altre en el nostre node a on executaríem Nagios configurar de manera manual tots els serveis que hem arrancat.

8 Nagios és un programari de monitorització de serveis i hosts molt estès degut a la seva gran versatilitat

Això implica una doble feina ja que per cada servei hauríem de configurar el servei en sí i la monitorització per una altra banda, fent que a la llarga, i a mesura que el número de màquines i de serveis augmenta, el manteniment d'aquesta infraestructura es faci més pesat i requereixi més temps.

Puppet soluciona aquesta problemàtica amb els "exported resources". D'aquesta manera, des de la màquina a on s'executarà el nostre servei web serà la mateixa encarregada de crear a la màquina de Nagios la monitorització adequada.



D'aquesta manera el mateix node a on executem el nostre servei web definirà el recurs de monitorització de Nagios i puppetmaster el guardarà a la seva base de dades interna.

Quan el client de puppet que s'està executant al node de Nagios es conecti al servidor, aquest li enviarà tots els recursos definits en altres nodes i passaran a formar part de l'estat desitjat d'aquest sistema.

Per declarar un recurs com a "exported resource" s'ha de fer amb dues arrobes (@@) davant del nom del recurs.

Exemple extret de la documentació de Puppet:

- Part del node que definirà el "exported resource"

```
class nagios-target {
   @@nagios_host { $fqdn:
        ensure => present,
        alias => $hostname,
        address => $ipaddress,
        use => "generic-host",
   }
   @@nagios_service { "check_ping_${hostname}":
        check_command => "check_ping!100.0,20%!500.0,60%",
        use => "generic-service",
        host_name => "$fqdn",
        notification_period => "24x7",
        service_description => "${hostname}_check_ping"
   }
}
```

- Part del node del servidor de Nagios

```
class nagios-monitor {
    package { [ nagios, nagios-plugins ]: ensure => installed, }
    service { nagios:
        ensure => running,
        enable => true,
        #subscribe => File[$nagios_cfgdir],
        require => Package[nagios],
    }
    # collect resources and populate /etc/nagios/nagios_*.cfg
    Nagios_host <<||>>
    Nagios_service <<||>>
}
```

Llenguatge de Puppet

Per a la definició de manifests en Puppet es fa servir un llenguatge propi. Tot i que incorpora alguns conceptes com classes, herències o variables, no es tracta d'un llenguatge de programació pròpiament dit sinó més bé un llenguatge descriptiu.

Mitjançant aquest llenguatge, definirem l'estat desitjat de cada un dels nostres nodes/sistemes

Recursos

Un recurs és l'element més petit que podem trobar en un manifest de Puppet. Pot definir una gran varietat de coses, començant per un simple fitxer en un determinat directori del disc dur o un servei que ha d'estar executant-se constantment.

Puppet incorpora de sèrie una gran quantitat de recursos que seran útils per a la gran majoria dels casos, però en el cas que necessitem fer alguna cosa que no està disponible amb tots els recursos que puppet proveeix, sempre podem crear els nostres propis mitjançant Ruby[9].

Cada recurs té un tipus, un títol i una llista d'atributs. Aquests atributs poden ser específics de cada recurs o pot ser un dels "meta atributs" que estàn disponibles per a tots i cada un dels recursos existents.

Aquesta és la llista actual amb una breu descripció de tots els recursos disponibles:

- Augeas – Aplica un canvi fent servir l'aplicació augeas[10]
- computer – Gestiona un objecta "computer" en el servei de directori de Mac OS X
- cron – Crea i manté una entrada en el servei cron per executar comandes a intervals definits
- exec – Executa una comanda desde la linea de comandes si es compleixen certes condicions
- file – Gestiona un fitxer determinat, els seus permisos i el seu contingut.
- filebucket – És un repositori per a fer backup de fitxers. Cada cop que puppet modifica o elimina un fitxer al nostre disc, es guarda una copia a aquí
- group – Crea un grup de sistema
- host – Gestiona una entrada en el fitxer /etc/hosts
- interface – Representa un interfaç en un router o switch. Es tracta d'un recurs experimental encara
- k5login – Gestiona el fitxer .k5login de un usuari determinat.

9 Ruby és el llenguatge de programació amb el que s'ha fet Puppet i el que es fa servir per ampliar-lo

10 Augeas és una eina per a edició de fitxers de configuració desde la linea de comandes

- macauthorization – Gestiona la base de dades de usuaris de Mac OS X
- mailalias – Representa un alias en el nostre sistema de correu, normalment a /etc/aliases
- maillist – Crea o borra llistes de correu
- mcx – Gestiona objectes MCX en el servei de directori de Mac OS X
- mount – Gestiona muntatge i desmuntatge de particions, tant locals com remotes (per xarxa)
- nagios_command – Representa un "command" en la configuració de Nagios
- nagios_contact - Representa un "contact" en la configuració de Nagios
- nagios_contactgroup - Representa un "contactgroup" en la configuració de Nagios
- nagios_host - Representa un "host" en la configuració de Nagios
- nagios_hostdependency - Representa un "hostdependency" en la configuració de Nagios
- nagios_hostescalation - Representa un "hostescalation" en la configuració de Nagios
- nagios_hostextinfo - Representa un "hostextinfo" en la configuració de Nagios
- nagios_hostgroup - Representa un "hostgroup" en la configuració de Nagios
- nagios_service - Representa un "service" en la configuració de Nagios
- nagios_servicedependency - Representa un "servicedependency" en la configuració de Nagios
- nagios_serviceescalation - Representa un "serviceescalation" en la configuració de Nagios
- nagios_serviceextinfo - Representa un "serviceextinfo" en la configuració de Nagios
- nagios_servicegroup - Representa un "servicegroup" en la configuració de Nagios

- nagios_timeperiod - Representa un "timepoperiod" en la configuració de Nagios

- notify – Envia un missatge al client de puppet perque el mostri per pantalla.

- package – Representa un paquet que contindrà un determinat programari. En cada sistema operatiu la definició de paquet varia bastant, però en general s'intentarà sempre descarregar-lo d'un repositori central i instal·lar-lo mitjançant les eines disponibles

- resources – Mitjançant aquest recurs es poden gestionar altres recursos ja definits anteriorment

- router – Gestiona un router. Es tracta d'un recurs experimental en desenvolupament

- schedule – Defineix un horari a puppet de manera que podrem fer que determinats recursos nomes s'executin en l'interval de temps especificat.

- scheduled_task – Configura una tasca programada en un sistema Windows

- selboolean – Gestiona un boleà en un sistema amb SELinux[11]

- selmodule – Gestiona la càrrega i descàrrega de mòduls de polítiques en un sistema amb SELinux

- service – Gestiona serveis en un sistema basat en Unix.

- ssh_authorized_key – Gestiona les claus ssh al fitxer authorized_keys

- sshkey – Gestiona claus ssh de host

- stage – Defineix un "stage" en puppet, de manera que podem agrupar les execucions de recursos juntes.

- tidy – Borra fitxers del disc dur seguint un criteri determinat (per data, nom, etc).

- user – Gestiona usuaris de sistema

- vlan – Representa una vlan en un switch o router. És un recurs experimental que encara està en desenvolupament.

11 SELinux és un sistema de seguretat avançat per a sistemes Linux

- yumrepo – gestiona la definició d'un repositori de Yum de on baixar paquets amb programari.

- zfs – Gestiona instancies de zfs[12]

- zone – Gestiona una zona[13] en un sistema Solaris

- zpool – Gestiona una zpool[14] en un sistema operatiu Solaris

Classes i definicions

Tots els recursos anteriors es poden agrupar en el que anomenem "classes" o "definicions". D'aquesta manera podem crear, per exemple, una classe anomenada "webserver" que contindrà tots els recursos necessaris per configurar un servidor web amb la configuració que nosaltres desitgem.

Tant les classes com les definicions accepten paràmetres, de manera que podrem ajustar-les a les nostres necessitats.

La diferència bàsica entre una classe i una definició serà que una definició pot ser cridada varis cops en un mateix node, mentre que una classe serà única.

Exemple de classe:

```
class webserver {

      service { 'apache':
            require => Package['httpd']
      }

      package { 'httpd':
            ensure => installed,
      }

      file { '/etc/httpd/httpd.con':
            ensure  => present,
            content => 'modules/apache/httpd.conf.erb',
            notify  => Service['httpd'],
      }

}
```

12 Sistema d'arxius en el sistema operatiu Solaris
13 És el sistema de virtualització incorporat en el sistema operatiu Solaris
14 És un pool de storage en el sistema de fitxers zfs de Solaris

Aquesta mateixa classe la podríem reescriure perquè acceptés un paràmetre amb el port en el que volem que escolti el servei de apache:

```
class webserver (
      $apache_port = '80'
) {

      service { 'apache':
            require => Package['httpd']
      }

      package { 'httpd':
            ensure => installed,
      }

      file { '/etc/httpd/httpd.con':
            ensure  => present,
            content => 'modules/apache/httpd.conf.erb',
            notify  => Service['httpd'],
      }

}
```

A dins de la plantilla httpd.conf.erb hi definirem una variable apache_port que puppet substituirà amb el valor que li passem com a paràmetre. Si no li especifiquem cam paràmetre, agafarà el valor per defecte (80)

La manera com cridar aquesta classe varia substancialment si es crida amb paràmetres que si es crida sense:

Sense paràmetres:

```
include webserver
```

Amb paràmetres:

```
class { “webserver”:
      apache_port => '81',
}
```

Una definició en canvi sempre necessita un paràmetre com a mínim. Aquest paràmetre és el que s'anomena “namevar” i s'assigna automàticament a la variable $name o $title.

D'aquesta manera, i al contrari que en les classes, podem cridar varies definicions iguals però amb diferent “namevar”.

Per exemple:

```
define svn_repo($path) {
      exec { "/usr/bin/svnadmin create ${path}/${title}":
            unless => "/bin/test -d ${path}",
      }
}

svn_repo { 'puppet_repo': path => '/var/svn_puppet' }
svn_repo { 'other_repo':  path => '/var/svn_other' }
```

Les classes tenen una forma simple del concepte d'herència present en els llenguatges de programació orientats a objectes. D'aquesta manera, una classe pot heretar totes les característiques de la seva classe pare i modificar-les o afegir-ne de noves.

Exemple de herència de classes:

```
class apache {
      service { 'apache':
            require => Package['httpd']
      }
}

class apache-ssl inherits apache {
      Service['apache'] {
            require +> File['apache.pem']
      }
}
```

Nodes

En primer lloc haurem de definir tots i cada un dels sistemes que voldrem configurar. En el llenguatge de Puppet això és considerat com un "node".

Cada un dels nodes que vulguem definir, ho haurem de fer amb el nom del node. Tot i que no està recomanat, puppet suporta fer servir només el nom de la màquina sense el nom del domini, però degut a que pot portar problemes no és una bona pràctica.

D'aquesta manera en un node, podem agrupar totes les definicions, classes o recursos que siguin necessaris per a configurar un sistema en concret.

Per exemple:

```
node 'www.uoc.edu' {
      include apache
      include java
}
```

Els nodes també suporten herència de la mateixa manera que ho fan les classes.

De fet, una classe i un node no son tan diferents ja que tot i que un node no accepta cap tipus de paràmetre ni pot ser cridat en una altra part del codi, també serveix per agrupar diferents classes, recursos o definicions que s'executaran en un o varis dels nostres sistemes.

En el nom del node es poden fer servir expressions regulars que es compararan amb el nom dels nostres sistemes.

Per exemple, el node següent:

```
node /^www\d+$/ {
      include apache
      include java
}
```

Farà "match" amb tots els nodes el noms dels quals comencin per "www".

També podem definir un node per defecte que contindrà la configuració en el cas que cap altre definició de node correspongui. Aquest node tindrà el nom "default"

Mòduls

Per a facilitar la organització del nostre codi es poden (i de fet és molt recomanable) fer servir el que s'anomenen mòduls.

Un mòdul és una col·lecció portable de classes, definicions i recursos amb tots els seus fitxers necessaris, de manera que teòricament un cop escrit, si ha estat ben programat, aquest mòdul podrà ser executat en qualsevol altra instància de Puppet. De fet, els mateixos creadors de Puppet han creat una web a on es poden compartir els mòduls creats per la gent: http://forge.puppetlabs.com/

Tots els mòduls han d'estar localitzats a un directori concret (normalment /etc/puppet/modules) i segueixen una estructura ben definida de directoris:

```
MODULE_PATH/
     nom_del_modul_minuscules/
        files/
        manifests/
            init.pp
            foo.pp
        lib/
            puppet/
                parser/
                    functions/
                provider/
                type/
            facter/
        templates/
```

```
        tests
            init.pp
            foo.pp
        README
```

El principal fiter, i de fet l'únic estrictament necessari per poder executar el mòdul, és manifests/init.pp que habitualment contendrà una classe amb el nom del mòdul. És una pràctica recomanada el afegir la resta de classes o definicions en fitxers diferents de init.pp

Per exemple, en un mòdul anomenat webserver, al fitxer init.pp podríem tenir algo similar a això:

```
class webserver (
      $apache_port = '80'
) {

      service { 'apache':
            require => Package['httpd']
      }

      package { 'httpd':
            ensure => installed,
      }

      file { '/etc/httpd/httpd.con':
            ensure  => present,
            content => 'modules/apache/httpd.conf.erb',
            notify  => Service['httpd'],
      }

}
```

I també podríem tenir un altre fitxer anomenat ssl.pp amb aquest contingut:

```
class webserver::ssl inherits webserver {
      Service['apache'] {
            require +> File['apache.pem']
      }
}
```

Puppet, fent servir el que anomenen “autoloading”, automàticament anirà a buscar el fitxer localitzat a “/etc/puppet/modules/webserver/manifests/ssl.pp” quan fem un “include webserver::ssl”

D'aquesta manera podem tenir tots els nostres mòduls d'una manera organitzada i

estructurada, facilitant el seu us i organització.

En els directoris "files" i "templates" es a on hi posarem tots els fitxers necessaris per al nostre mòdul. La diferència entre els dos és que a "files" hi aniran tots els fitxers estàtics que es cridaran mitjançant la propietat "source" del recurs "file" i a "templates" hi aniran totes les plantilles escrites amb el llenguatge ERB[15] i que seran cridades mitjançant la propietat "content" i la funció "template" del recurs "file".

Mitjançant aquestes plantilles, podrem generar fitxers de configuració dinàmics depenent de les variables que especifiquem dintre del nostre mòdul.

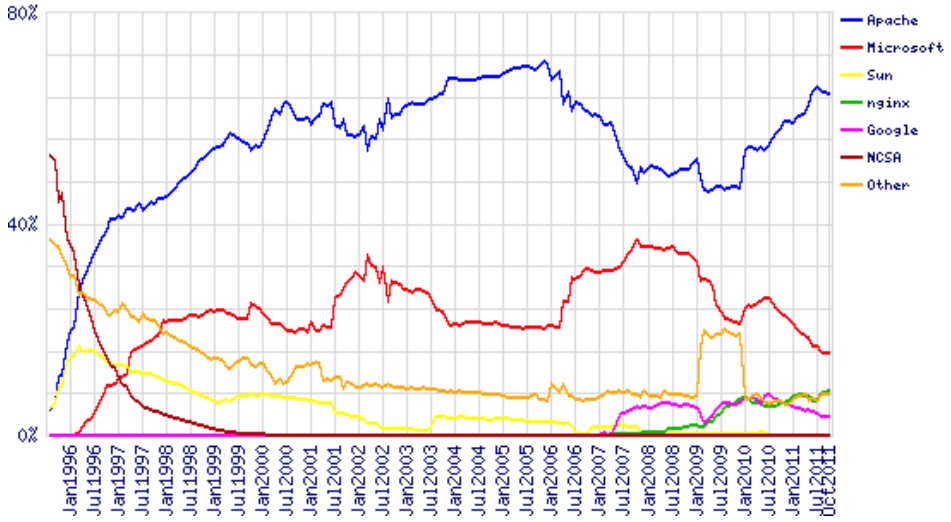
A dins el directori "lib" és on podrem escriure els nostres propis facts, les nostres pròpies funcions o els nostres "providers" en Ruby, permetent així estendre les capacitats de Puppet fins molt més enllà del que permet de sèrie.

Finalment el directori "test" és el que tindrà tots els tests necessaris per comprovar el correcte funcionament del mòdul.

Apache httpd

- http://httpd.apache.org/

El servidor web per excel·lència, s'ha convertit amb el temps amb el més utilitzat a tot Internet. I és que la seva potència i versatilitat l'han convertit en el número 1 desde ja fa temps, i tot i que actualment hi ha altres alternatives més orientades a obtenir un millor rendiment, el fet que porti tants anys i que la seva estabilitat i potència son mundialment reconegudes, ha fet que ninguna de les alternatives existents hagi pogut assolir un índex d'us tan elevat.



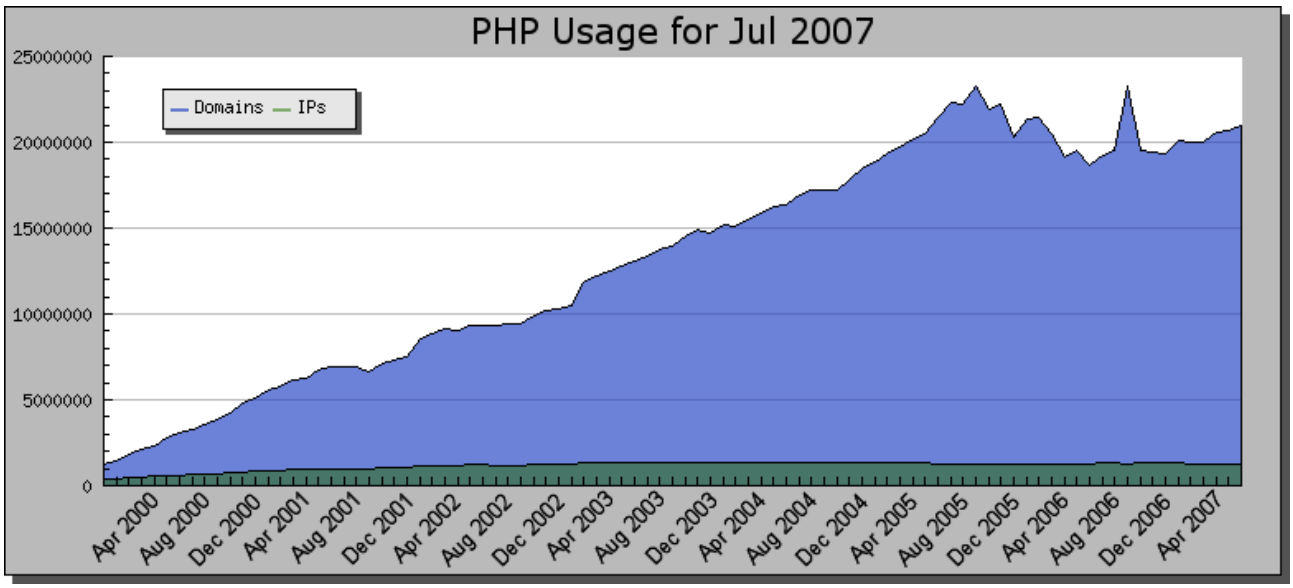
Us de servidors web per dominis

15 ERB és el llenguatge de generació de plantilles propi de ruby

PHP

- http://php.net/

PHP és un llenguatge de programació interpretat i d'us general, originalment creat per al desenvolupament de pàgines web dinàmiques. Tot i que avui en dia es fa servir per moltes altres coses, el seu principal us segueix sent la web. PHP està sent utilitzat en mes de 20 milions de pàgines web i 1 milió de servidors web[16].



Algunes de les característiques que han fet de PHP un llenguatge extremadament popular en el web son aquestes:

- Sintaxis inspirada en llenguatge C i C++ (en la programació orientada a objectes)
- Gran facilitat per incloure'l a una pàgina html fent servir els coneguts tags <?php ?>
- Gran suport integrat de les bases de dades més populars en entorns web, com poden ser MySQL, PostgreSQL, SQLite o fins i tot Oracle i MSSQL
- Suport a múltiples plataformes (tant les basades en Unix com les que no)
- És programari lliure
- La documentació és de gran qualitat i és revisada i actualitzada regularment

16 Segons http://www.php.net/usage.php (xifres del 2007)

MySQL

- http://www.mysql.com/

MySQL és un sistema de gestió de bases de dades relacionals de codi obert tot i que actualment és propietat de Oracle.

Va començar com un projecte que no estava associat a cap empresa, però la seva popularitat va fer que fos comprada per Sun Microsystems a l'any 2008. Dos anys després, al 2010, Oracle Corporation va comprar Sun Microsystems, adquirint també la propietat de MySQL. Aquest fet va fer que es temés per la continuïtat del projecte de manera lliure ja que pot entrar en conflicte amb altres productes desenvolupats per la mateixa Oracle. Per aquesta raó van sorgir altres branques que feien servir el mateix codi de MySQL però que no eren propietat d'Oracle[17].

Oracle s'ha compromès a seguir amb el sistema actual de doble llicència al menys fins l'any 2015.

A nivell tècnic, MySQL pot fer servir diferents "motors" que poden aportar diferents característiques. Els dos més coneguts son els següents:

- MyISAM: Normalment utilitzat per a webs de poca càrrega. No suporta transaccions, però per aquesta mateixa raó és molt més ràpida que d'altres mes complexes

- InnoDB: Suporta transaccions i d'altres característiques avançades. És el motor per defecte en les darreres versions de MySQL

VirtualBox

- http://www.virtualbox.org/

VirtualBox és un programa de virtualització que permet executar diferents sistemes operatius totalment independents dins del mateix "host pare".

És programari lliure actualment propietat de Oracle Corporation després de l'adquisició de Sun Microsystems.

El fet que sigui un dels pocs sistemes de virtualització multiplataforma i distribuit de manera gratuïta, ha fet que sigui un dels sistemes de virtualització més populars.

17 Per exemple: MariaDB

Altres

- LibreOffice (http://www.libreoffice.org/). Per redactar la documentació.
- Omniplan (http://www.omnigroup.com/products/omniplan/). Per fer els diagrames de Gantt
- OmniGraffle (http://www.omnigroup.com/products/omnigraffle/). Per fer la resta de diagrames

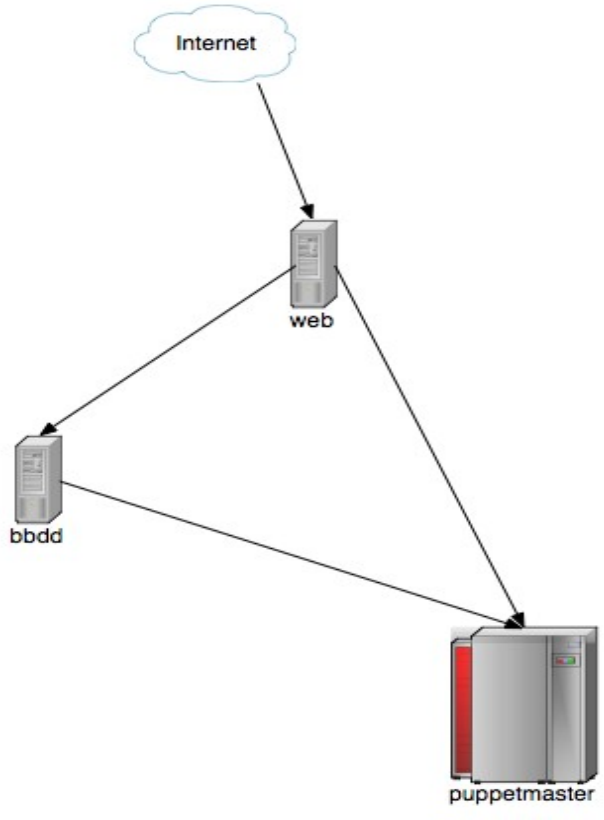
Maquinari

Per poder muntar tota l'arquitectura de la plataforma, es necessitaran com a mínim tres servidors.

1. **Servidor web**: Serà l'encarregat de estar connectat directament a Internet i servir totes les peticions que rebi
2. **Servidor de base de dades**: Encarregat de guardar tota la informació relativa a l'aplicació web
3. **Servidor Puppet**: Encarregat d'administrar totes les màquines que formen la plataforma.

Per poder dur a terme el projecte, es faran servir màquines virtuals en comptes de fer servir maquinari físic.

En un projecte real, el nombre de màquines segurament seria bastant superior, ja que s'haurien de tenir en compte coses com alta disponibilitat i balanceig de carrega, però per poder simplificar, i ja que el projecte tracta principalment sobre fer servir Puppet per crear i mantenir tota aquesta plataforma, nomes crearem tres servidors virtuals.



Resultats i anàlisi del treball

Entorn

Per realitzar el treball es van instal·lar fins a tres màquines virtuals que serien les encarregades de simular els tres servidors diferents:

- servidor de puppet (puppetmaster)
- servidor web (Apache)
- servidor bbdd (MySQL)

Com a entorn de virtualització es va escollir VirtualBox per ser un producte gratuït i de gran us dins de la comunitat Open Source.

De totes maneres, s'han tingut problemes bastant evidents de rendiment al executar tota la plataforma, ja que el host "pare" (el que executava totes les màquines virtuals) no tenia potència suficient per poder dur a terme totes les tasques que se li demanaven, fent que algunes tasques que haurien de ser gairebé immediates triguessin bastant més de l'esperat.

La solució hagués sigut executar tota la plataforma en un entorn amb un maquinari molt més potent del que actualment es disposa.

Puppet Master

La configuració i instal·lació de Puppet s'ha realitzat íntegrament fent servir els paquets RPM oficials, de manera que no s'ha hagut de compilar ningun programa ni s'ha hagut de fer una instal·lació manual, de manera que el sistema queda mes net i organitzat ja que es fan servir les eines que aporta el sistema operatiu base (CentOS) per instal·lar i mantenir el programari.

Un com instal·lat i configurat el servidor "puppetmaster" s'ha procedit a escriure el codi necessari per la configuració de la resta de servidors (Apache i MySQL)

Mòduls de Puppet

Firewall

És un mòdul genèric preparat per a configurar i gestionar el servei de tallafocs inclòs en Linux: iptables.

El mòdul consta bàsicament de dues parts:

Classe iptables

És l'encarregada de configurar el sistema perquè faci servir el tallafocs. És l'encarregada d'assegurar-se que el servei iptables està arrancat en tot moment i de configurar els fitxers necessaris per al seu correcte funcionament.

També crea el directori on es guardaran els fitxers de text amb totes les regles del tallafocs i instal·la el script que s'encarregarà d'ajuntar-les en un sol fitxer.

Funció firewall::rule

És l'encarregada de crear les regles del tallafocs. Amb aquesta funció, podem crear regles que permetin o deneguin accés a qualsevol port o ports de la màquina. Bàsicament el que fa és crear un fitxer de text a /etc/sysconfig/iptables.d/ amb la regla i tot seguit executa un script encarregat de concatenar tots els fitxers de text d'aquest directori en un de sol.

Un cop es reinicia el servei iptables, llegirà d'aquest fitxer de text per carregar totes les regles del tallafocs

Exemple d'us:

```
firewall::rule { "allow_port_80":
    comment   => "Allow webserver",
    sources   => "0.0.0.0/0",
    interface => "eth0",
    protocols => "tcp",
    ports     => "80",
    action    => "ACCEPT",
}
```

Aquest codi crearà una nova regla de firewall que obrirà el port 80/tcp (http) a qualsevol adreça ip.

Apache

El següent mòdul és l'encarregat de la configuració del servidor web Apache. S'ha intentat crear un mòdul bastant genèric que servirà per a la configuració de pràcticament qualsevol entorn.

Classe apache

Encarregada d'instal·lar el servidor web Apache i de preparar l'entorn necessari per al seu

correcte funcionament. Per exemple, aquesta classe serà l'encarregada d'assegurar-se que el servei httpd estarà funcionant constantment.

De totes maneres no realitzarà cap tipus de configuració, ja que d'això s'encarregarà una altra funció que veurem més endavant.

Classe apache::ssl

Una classe filla de "apache" que realitzarà les mateixes funcions que la primera, però que a més instal·larà i prepararà tot el necessari per poder fer servir apache en mode segur SSL.

Funció apache::configure

Aquesta funció si que serà l'encarregada de configurar Apache segons els nostres requeriments. Admet gran quantitat de paràmetres que podem fer servir per ajustar des de quants processos d'Apache volem executar fins a quins mòduls volem carregar. S'han intentat definir uns valors per defecte vàlids per a qualsevol entorn.

Exemple d'us:

```
apache::configure { $::fqdn:
     keepalive  => "On",
     maxclients => "500",
}
```

Funció apache::module

Aquesta funció és l'encarregada de carregar un mòdul d'Apache. S'ha preparat tant per poder gestionar mòduls genèrics inclosos dins de la distribució d'Apache del sistema operatiu com per instal·lar paquets externs d'alguns dels mòduls més populars com php, security o python entre d'altres.

Exemple d'us:

```
apache::module { "php":
     ensure => present,
}
```

Funció apache::vhost

Funció encarregada de configurar un "virtual host" d'Apache, per poder servir una pàgina web. Aquesta funció només s'encarregarà de configurar aquest virtual host dins de la configuració d'Apache, i en cap moment gestionarà el seu contingut, ja que això és una tasca que s'escapa de les funcions a realitzar per Puppet

Exemple d'us:

```
apache::vhost::raw { “www.uoc.edu”:
     server_alias => “uoc.edu”,
     document_root => “/var/www/html/uoc”,
}
```

Altres

També s'han creat algunes funcions auxiliars d'us intern que no estan pensades per ser fetes servir fora del propi mòdul i que per tant no es documentaran dins d'aquest document.

MySQL

Una de les grans avantatges de fer servir Puppet per a administrar sistemes, és que es pot aprofitar feina feta per altres administradors de sistemes o devops. Això és precisament el que s'ha fet amb aquest mòdul. En comptes de començar-ne un desde zero, s'ha aprofitat un mòdul existent de MySQL i s'ha modificat per adaptar-lo a les nostres necessitats.

Classe mysql::server

Instal·la i prepara l'entorn per a un correcte funcionament del servidor MySQL. No admet paràmetres però necessita que es defineixi previament la variable $mysql_rootpw amb el password del usuari “root” que volem configurar.

Exemple d'us:

```
$mysql_rootpw=”contrasenya”
include mysql::server
```

Funció mysql::server::configure

Tal i com la classe mysql::server només instal·la el servidor i prepara l'entorn, aquesta funció serà l'encarregada de configurar-lo segons les nostres necessitats.

D'aquesta manera, podem passar-li gran quantitat de paràmetres que equivalen als seus equivalents que hauràn d'anar al fitxer /etc/my.cnf

Exactament igual que en el cas de mòdul d'Apache, també s'han intentat buscar uns valors per defecte que siguin apropiats en la majoria dels casos.

Exemple d'us:

```
mysql::server::configure { $::fqdn:
```

```
        max_connections => “200”,
        wait_timeout => “10”,
    }
```

Funció mysql_database

Funció encarregada de crear una base de dades dins del servidor MySQL. Puppet la crearà i s'encarregarà de comprovar en cada execució que existeix.

De totes maneres, en cap moment modificarà ni crearà cap taula ni cap registre dins d'aquesta base de dades. Com en el cas del mòdul d'Apache, no és funció de Puppet gestionar el contingut.

Exemple d'us:

```
    mysql_database { “uoc”:
        ensure => present,
    }
```

Funció mysql_user

Funció encarregada de crear un usuari de MySQL. Amb aquesta funció únicament es crearà l'usuari necessari amb un password específic, però no es gestionarà cap dels seus permisos ni privilegis.

Exemple d'us:

```
    mysql_user { “uoc@%”:
        ensure => present,
        password_hash => mysql_password(“contrasenya”),
    }
```

Funció mysql_grant

Tal i com mysql_user crearà l'usuari necessari per fer login al servidor, amb mysql_grant podem assignar i treure permisos a un usuari determinat.

Podem assignar permisos tant de forma genèrica per a totes les bases de dades del sistema, com per a una base de dades específica.

Exemple dús:

```
    mysql_grant { “uoc@%”:
        privileges => [ "select_priv" ],
    }
    mysql_grant { “uoc@%/uoc”:
        privileges => [ "select_priv", 'insert_priv', 'update_priv',
```

```
'delete_priv' ],
     }
```

Configuració de nodes

Un cop tenim tots els nostres mòduls llestos per a poder fer-se servir, hem de definir els nodes que formaran la nostre arquitectura.

En el nostres cas definirem tan sols 3 nodes, ja que es tracta d'un entorn de proves, però un cop hem definit la configuració per a aquests 3, la podem extrapolar sense cap dificultat a un nombre indeterminat de nodes, de manera que tindrem X màquines configurades exactament de la mateixa manera i sempre en el mateix estat.

Per a dur a terme aquest projecte, s'ha optat per realitzar configuracions bastant bàsiques de tots els serveis a instal·lar, per poder demostrar la potència a l'hora de fer canvis un cop una plataforma ja està instal·lada i configurada.

Tots els nodes

S'ha definit una configuració genèrica que s'aplicarà a tots els nodes de la nostra plataforma. D'aquesta manera podrem estar segurs que certes configuracions estaran sempre de la mateixa manera sigui quina sigui la funció d'aquest node en concret.

Per exemple, es configura el tallafocs iptables en totes les màquines i s'obrirà el port ssh (22/tcp) per poder accedir de manera remota a elles.

També es gestionarà el fitxer de configuració de puppet (/etc/puppet/puppet.conf) a totes les màquines per assegurar-nos que totes tindràn exactament la mateixa configuració.

Node tfc-puppet

Aquest serà el node encarregat de tenir el servidor de Puppet (puppetmaster).

Hem optat per fer una configuració bàsica també per a aquest node tot i ser el servidor central de Puppet ja que tot i realment no ser necessària del tot aporta certe avantatges com podria ser tenir tot el codi en un repositori git o svn i poder tirar enrere canvis perillosos.

En aquest cas tan sols ens hem assegurat que el paquet “puppet-server” està instal·lat i que el port 8140/tcp (puppetmaster) és obert per a tothom.

La gestió de certificats de puppet es farà manualment mitjançant la comanda “puppet cert”

Node tfc-web

Aquest node serà l'encarregat de tenir el servidor web Apache.

Mitjançant Puppet instal·larem el servidor web Apache i farem una configuració mínima per poder demostrar la capacitat de Puppet a l'hora de gestionar els fitxers de configuració i reiniciar el servei.

També crearem un virtual host senzill i instal·larem el mòdul de php.

Per acabar farem servir el mòdul de firewall per obrir el port 80/tcp (http) a tothom perquè el servei web sigui accessible desde tot arreu.

Node tfc-bbdd

El darrer dels nostres nodes serà l'encarregat de tenir la base de dades MySQL.

S'ha configurat mitjançant puppet la contrasenya de l'usuari "root" i s'ha creat una base de dades buida anomenada "tfc".

També s'ha obert el port 3306/tcp (mysql) per a tothom per poder accedir a la base de dades sense problemes des de qualsevol lloc. En cas de voler aplicar una política de seguretat més agressiva es pot fer sense problemes, només deixant connectar a una xarxa en concret o fins i tot a un grup de ips en concret.

Annex 1: Bibliografia

Per la elaboració d'aquest document s'han consultat les següents fonts:

- Wikipedia (http://www.wikipedia.org) en diversos idiomes: anglès, català i castellà
- Documentació oficial de Puppet (http://docs.puppetlabs.com/)