Gestión de la Configuración con - Puppet -

@madrid_devops @jmoratilla @ricbartm

Indice

Introducción
- > Qué es puppet?
- > Cómo lo hace?

Componentes
- > Lenguaje
- > Puppet client
- > Puppet Master

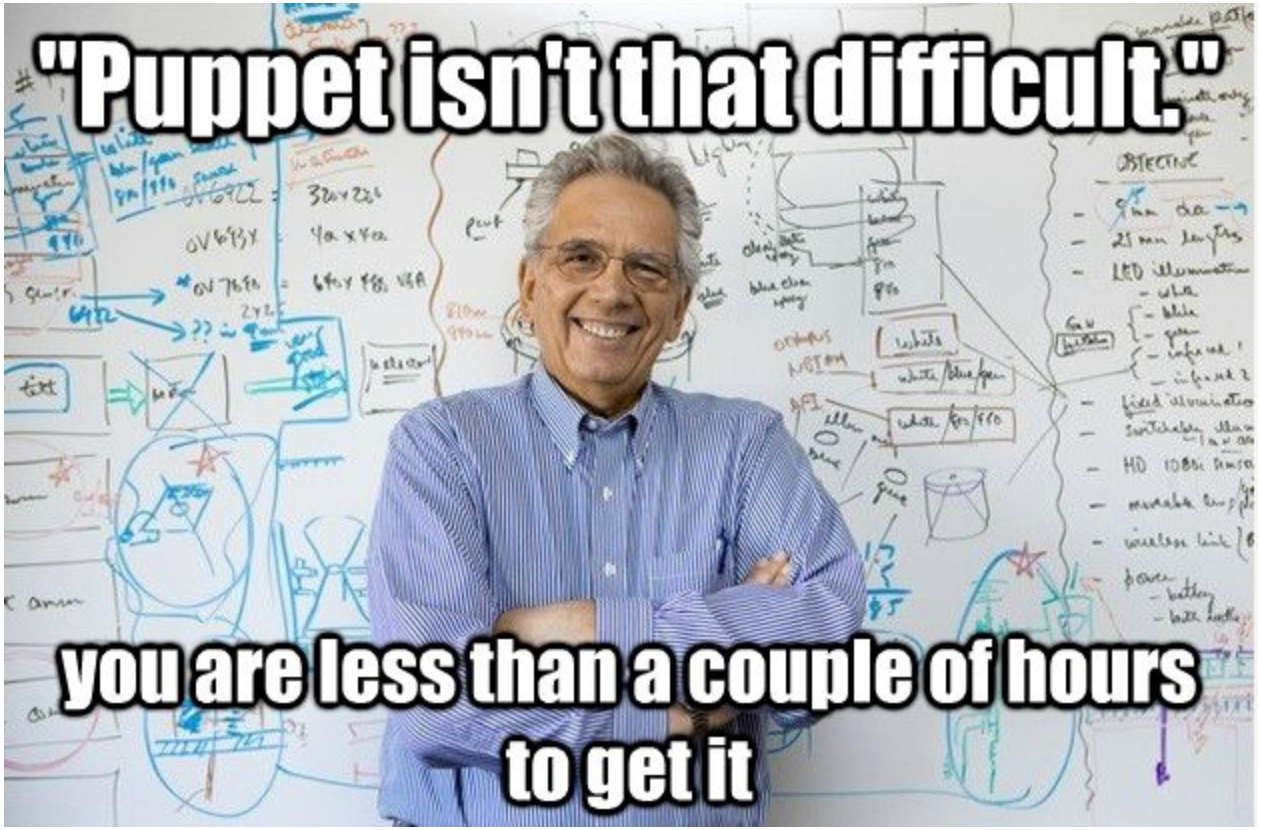
Puppet @ abstra

Puppet @ tuenti

Ponentes

@jmoratilla

@ricbartm


"Puppet isn't that difficult."
you are less than a couple of hours
to get it

puppet

Qué es puppet?

-- Sistema de gestión de la configuración

++ claaaroo... ¿Y eso para qué sirve?

-- Sirve para configurar y mantener un grupo de máquinas, todas iguales, de forma fácil y sencilla.

++ Ya! y algo más?

-- Permite actualizar automáticamente la configuración de las máquinas, y auditarlas.

++ Ok.

Por qué puppet?

Consistencia y flexibilidad

- La configuración manual no es recomendable. Errores humanos.
- Las "Golden Images" son un quebradero de cabeza.

Por qué puppet?

Consistencia y flexibilidad

- La configuración y el mantenimiento se realiza desde un control central.
- Permite reproducir configuraciones completas en caso de desastre (10th floor test).

Por qué puppet?

Consistencia y flexibilidad

- Roles, clases y nodos: misma definición, misma configuración, o no ;)
- Un agente corre cada 30 minutos: cada 30 minutos se comprueba que todo está en su sitio, y si no, se vuelve a poner en su sitio.

Por qué puppet?

Operativa

- La curva de aprendizaje y el <grin> "Time To Market" </grin> son muy cortos.
- Toda la lógica de configuración bajo Puppet: permite que cualquiera lo lea y entienda sin tener que indagar.
- No más páginas horribles explicando cómo configurar algo. Céntrate en el servicio en sí.

Por qué puppet?

Portabilidad

- La configuración, al ser reproducible, permite llevar el servicio a una nube privada, pública, o a un entorno de pruebas aislado con facilidad.
- Si los módulos están bien escritos, no debería importar mucho si se trata de Debian, Ubuntu, CentOS o RHEL.

Cómo lo hace?

Tecnología

- Basado en lenguaje ruby.
- Utiliza un lenguaje específico de dominio (DSL) orientado a "recursos"

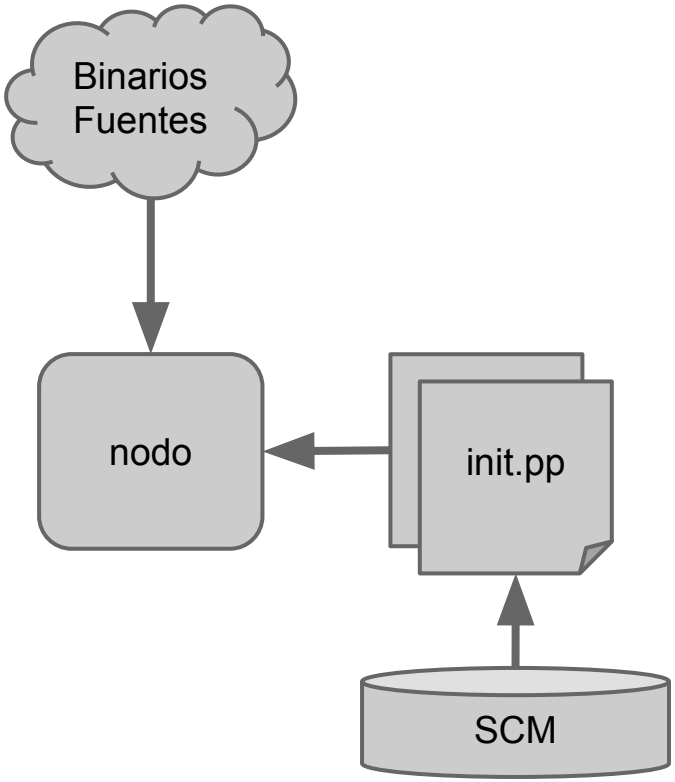
  Recursos? Comandos, Usuarios, Ficheros, Paquetes de software, Servicios, Configuraciones específicas, etc.

Cómo lo hace?

La práctica:

Stand Alone (puppet apply)

- El programa (agente) accede a unos ficheros de definición (.pp)
- Verifica si el estado de los recursos indicados en el fichero coinciden con la situación actual.
- Si todo es correcto, aplica los cambios hasta coincidir con lo indicado en el fichero de definición.



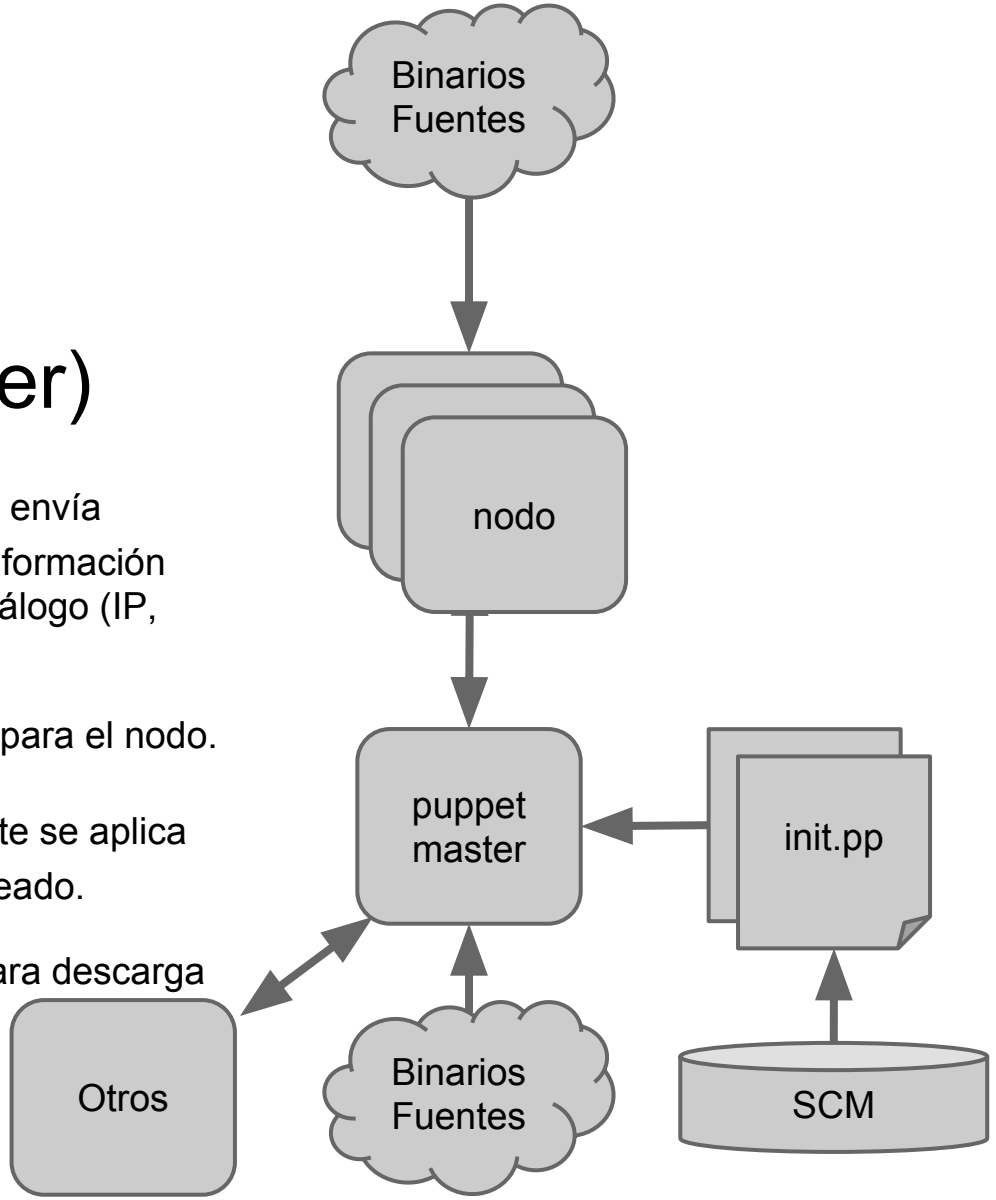
puppet

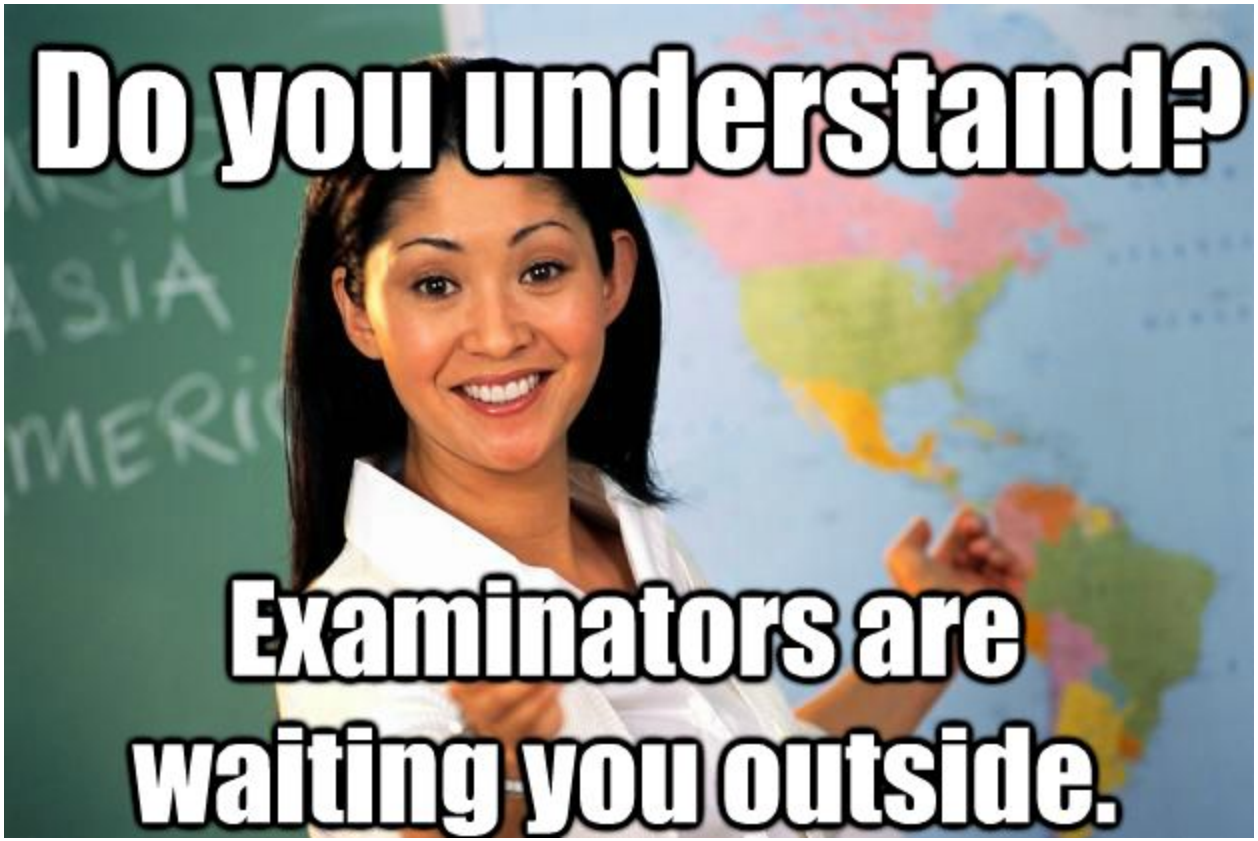
Cómo lo hace?

La práctica:

Servidor (puppetmaster)

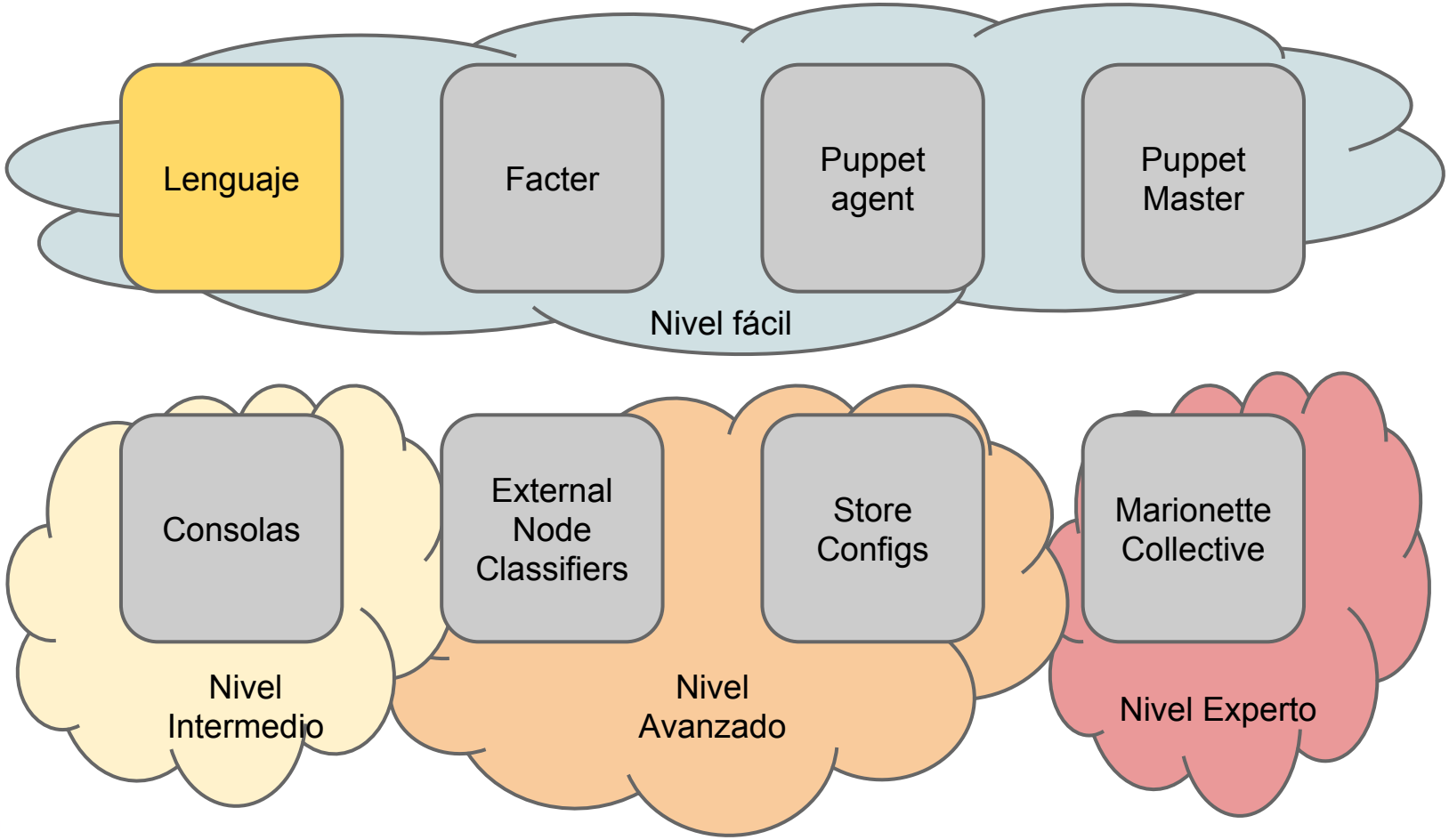
- El cliente (agente) conecta al master y le envía información de sí mismo. Parte de esa información podría ser reutilizada para generar el catálogo (IP, hostname, etc).
- El master compila un catálogo a medida para el nodo.
- Una vez el nodo dispone del catálogo éste se aplica localmente llegando hasta el estado deseado.
- Puppet Master tiene su propia interfaz para descarga de ficheros.




Do you understand?
Examinators are waiting you outside.

Componentes de la infraestructura de Puppet

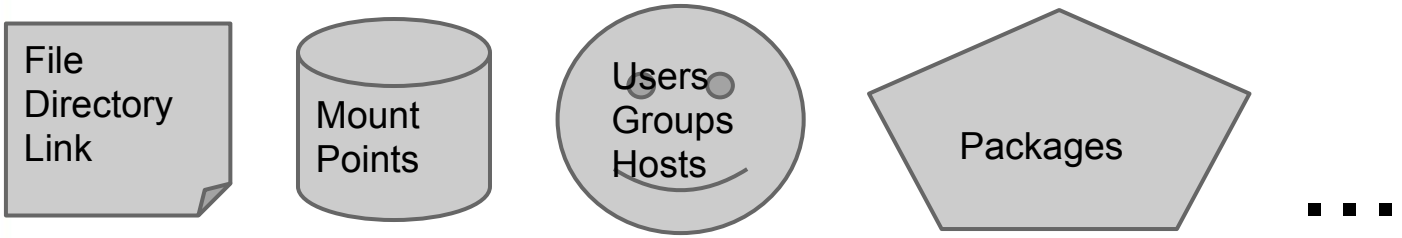
Componentes de puppet



El DSL de Puppet

Lenguaje: Definición de Recursos

Recursos: elementos de un sistema operativo



Tipos Nombres Parámetros

Recursos

```
file { "/etc/hosts":
  ensure  => present,
  owner   => root,
  group   => root,
  mode  => 0644
}
```

```
user {
"scarter":
  ensure  => present,
  uid  => 1000,
  gid  => 1000,
  managehome  => true,
"operator":
  ensure  => absent,
}
```

```
package {
  "openjdk-6-jdk":
  ensure  => present,
  provider  => apt,
}
```

```
cron { "logrotate":
  command  =>
"/usr/sbin/logrotate",
  user   => root,
  hour   => ['2-4'],
  minute  => '*/10'
}
```

Recursos: Paquetes y versiones

```
package {
  "openjdk-6-jdk":
  ensure   => "present",
  provider  => apt,
}
```

Instalar ....

```
package {
  "openjdk-6-jdk":
  ensure   => "6b22-1.10.6-0ubuntu1",
  provider  => apt,
}
```

.... esta versión ...

```
package {
  "openjdk-6-jdk":
  ensure   => "latest",
  provider  => apt,
}
```

.... o ir a la última!

Recursos: Tipos y Proveedores

```
service {
  "mongodb":
  ensure   => running,
  provider   => debian,
}
```

```
type: service
parameters:{
    name, path,
    ensure, enable,
    hasrestart,hasstatus,
    pattern, ...
    }
providers: {
     base, bsd, daemontools
     debian, freebsd, gentoo
     init, launchd, redhat
     runit, smf, src, ...
     }
```

Recursos: Genéricos

Sirven tanto para un roto...

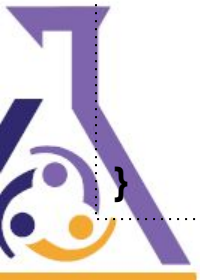
```
exec { "install glassfish":
      require  => [Package["unzip"],User["glassfish"],File["upload
glassfish"]],
      command => "unzip /var/tmp/glassfish-3.1.1.zip",
      cwd      => "/opt",
      user     => "glassfish",
      creates  => "/opt/glassfish3",
      path     => ['/bin','/usr/bin','/usr/sbin']
}
```

Recursos: Genéricos

... como para un descosido

```
exec { "add jdbc-connection-pool":
        require     => File["/opt/glassfish3/glassfish/lib"],
        command => "asadmin create-jdbc-connection-pool \
                --datasourceclassname com.mysql.jdbc.jdbc2.optional.MysqlDataSource \
                --restype javax.sql.DataSource
                --property User=glassfish:Password=$mysql_password:\
                        Url=\"jdbc:mysql://localhost:3306/test\":\
                        ZeroDateTimeBehavior=convertToNull DataChannelPool",
        path        => "/opt/glassfish3/bin",
}

exec { "add jdbc-resource":
        require     => Exec["add jdbc-connection-pool"],
        command => "asadmin create-jdbc-resource \
                --connectionpoolid JotadebecePool jdbc/data",
        path        => "/opt/glassfish3/bin",
}
```



Variables, Facts, Condiciones y Parámetros

Variables

Variables personalizadas definidas en los manifests

Relativa

$version = "3.1.1"

Global (concepto: 'namespaces' y 'namespace raíz' $::)

$::version = "3.1.1"

De clase (namespace = clase::subclase::variable)

$glassfish::params::version = "3.1.1"

External lookups

Son siempre personalizados. Permiten definir los valores de una variable personalizada consultando datos externos (un CSV, MySQL, etc).

Variables de sistema (obtenidas mediante facter, siempre globales)

$hostname, $architecture, $memorysize, $memoryfree,...

Variables

Personalizadas

```
$openfire = "openfire_3_7_1.tar.gz"

file { "/var/tmp/$openfire":
        ensure  => present,
        owner   => "root",
        group   => "root",
        mode    => 644,
        source  => "puppet:///modules/openfire/$openfire"
}
```

Facts

De sistema

Afirmación: "kernelversion es un fact"

facter: programa auxiliar que recolecta y muestra información (facts) relativa al sistema.

Ej: facter kernelversion

2.6.32

Puppet accede a estas variables (facts) para usarlas en las definiciones, como valores para asignar o para usarlas en las condiciones de puppet. Los facts pueden extenderse creando tus propios ... "*custom facts"*.

Condiciones

Sentencia if-elsif-else:

if condición { bloque }
elsif condición { bloque }
else { bloque }

```
if ($hostname == "calypso") {
        file{ "/home/user/.m2/settings.xml":
                require    => File["/home/user/.m2"],
                ensure     => "present",
                mode       => "644",
                source     => "puppet:///modules/maven/settings.xml";
        }
}
```

Condiciones

Sentencia case:

```
case variable {
        valor1: { bloque }
        valor2: { bloque }
}

case $architecture {
        x86_64,amd64: {
                $arch = 'x64'
        }
        i386,i486,i586,i686: {
                $arch = 'i586'
        }
}
```

Condiciones

Sentencia selector:

```
$variable = $fact ? {
        valor1 => 'y',
        valor2 => 'z',
        default => 'x',
}

$arch = $architecture ? {
        x86_64,amd64            => 'x64',
        i386,i486,i586,i686     => 'i586',
        default                 => 'noarch',
        }
}
```

Parámetros

Te permiten diseñar módulos genericos que sirven para diferentes tipos de servidor sin harcodear hostnames dentro de los manifests. Un ejemplo de cómo NO hacerlo:

```
class sysctl {
        case $hostname ? {
                'database1.localdomain': {
                        file { "/etc/sysctl.conf":
                                source => "puppet:///sysctl/etc/sysctl.conf.database",
                        }
                }
        }
}
node database1.localdomain { include sysctl }
```

Parámetros

Una forma más limpia de hacerlo (no me atrevo a decir que la correcta):

```
class sysctl ( $server_role = 'default' ) {
        file { "/etc/sysctl.conf":
                source => "puppet:///sysctl/etc/sysctl.conf.$server_role",
        }
}


node database1.localdomain {
        class { 'sysctl': server_role => 'database' }
}
node frontend1.localdomain {
        class { 'sysctl': server_role => 'frontend' }
}
```

Nodos, Clases y Módulos

Nodos

Puppet describe la configuración de un nodo, definiendo los recursos y sus parámetros, en ficheros llamados manifiestos (manifests o ficheros *.pp).

Un nodo se referencia por ...

- un nombre estático 'bm.ovh.net'
- una expresión regular /^bm\..*$/

```
node '*.ovh.net' {

        user { 'paquito':
                ensure          => present,
                uid     => 2000,
                gid     => 2000,
                home  => '/home/paquito',
                managehome          => true,
                shell   => '/bin/zsh',
                require => [Package['zsh'],Group['fulanito']],
        }

        group { 'fulanito':
                ensure          => present,
                gid     => 2000,
        }

        package { 'zsh':
                ensure          => present,
        }
}
```

Nodos

En un nodo puede definirse toda la configuración que necesite esa máquina.

No es lo más elegante, pero funciona.

Si quieres hacerlo elegante y mantenible, necesitarás definir roles y trabajar con diferentes clases.

Clases

```
node '*.ovh.net' {

        user { 'paquito':
                ensure          => present,
                uid     => 2000,
                gid     => 2000,
                home  => '/home/paquito',
                managehome          => true,
                shell   => '/bin/zsh',
                require => [Package['zsh'],Group['fulanito']],
        }

        group { 'fulanito':
                ensure          => present,
                gid     => 2000,
        }

        package { 'zsh':
                ensure          => present,
        }
}
```

```
class usuarios {
        user { 'paquito':
                ensure          => present,
                uid     => 2000,
                gid     => 2000,
                home  => '/home/paquito',
                managehome          => true,
                shell   => '/bin/zsh',
                require => [Package['zsh'],Group['fulanito']],
        }
        group { 'fulanito':
                ensure          => present,
                gid     => 2000,
        }
}
class zsh {
        package { 'zsh':
                ensure          => present,
        }
}
node '*.ovh.net' {
        include usuarios, zsh
}
```


Clases

Las clases sólo definen lo que debe llevar el sistema, para complementar estas definiciones, se necesitan ficheros, binarios, scripts, etc.

Para estructurar una clase y sus herramientas auxiliares se definen los módulos.

Módulos

Puppet define módulos como:

"re-usable bundles of code and data"

Contienen clases, binarios, imágenes, scripts, tests, incluso código adicional para extender las funcionalidades de puppet.

Módulos

```
/etc/puppet/
    manifests/
        site.pp          <== Definiciones Generales
        nodes.pp         <== Definiciones de Nodos
    modules/
        foo/
            manifests/
                init.pp <== Definición de la Clase
                foo.pp  <== SubClase (foo::foo)
             files/
                   foo.jar <== Ficheros
             lib/
                 facter/
                     foo.rb <== "facts" personalizados

    lib/
         puppet/
             foo.rb <== Extensión de Puppet
```

Relaciones de Orden

Relaciones entre recursos

Puppet es declarativo, lo que importa es el final. El orden es cuestión de relaciones entre recursos, no de posición en un fichero de manifest.

Es responsabilidad nuestra establecer las relaciones entre recursos y controlar el orden de ejecución.

Relaciones entre recursos

Referencias a Recursos

Recurso	Referencia y Genéricos
user	User
file	File
package	Package
exec	Exec
...	...

Relaciones entre recursos

before (antes) y require (después)

notify (publisher) y subscribe (subscriber)

File before Exec

Exec require File

File notify Exec

Exec subscribe File

Relaciones entre recursos

Ejemplo del fichero modules/sysctl/manifests/init.pp donde desplegamos un fichero y solo si este cambia se ejecuta un comando.

```
class sysctl ( $server_role = 'default' ) {
        file { "/etc/sysctl.conf":
                source => "puppet:///sysctl/etc/sysctl.conf.$server_role",
        }
        exec { "reload_sysctl":
                command => "/sbin/sysctl -p",
                refreshonly => true,
                returns => [ 0,255 ],
                subscribe => File["/etc/sysctl.conf"],
                require => File["/etc/sysctl.conf"],
        }
}
```

Relaciones entre recursos

Ejemplo del fichero modules/sysctl/manifests/init.pp donde desplegamos un fichero y solo si este cambia se ejecuta un comando.

```
class sysctl ( $server_role = 'default' ) {
        file { "/etc/sysctl.conf":
                source => "puppet:///sysctl/etc/sysctl.conf.$server_role",
                notify => File["/etc/sysctl.conf"],
        }
        exec { "reload_sysctl":
                command => "/sbin/sysctl -p",
                refreshonly => true,
                returns => [ 0,255 ],
                require => File["/etc/sysctl.conf"],
        }
}
```

Relaciones entre recursos

Ejemplo del fichero modules/rsync/manifests/server.pp donde algunos ficheros solo se despliegan en sistemas Debian, y han de desplegarse antes de levantar el servicio.

```
class rsync ( $server_role = 'default' ) {
    case $operatingsystem {
        Debian: {
            file { "$rsync_default_config":
                content =>
            template("rsync/etc/default/rsync.erb.$server_role"),
                notify => Service["$rsync"],
                before => Service["$rsync"],
            }
        }
    }
```

Relaciones entre recursos

Podemos crear stages que funcionan como containers para tener un arbol de dependencias más limpio.

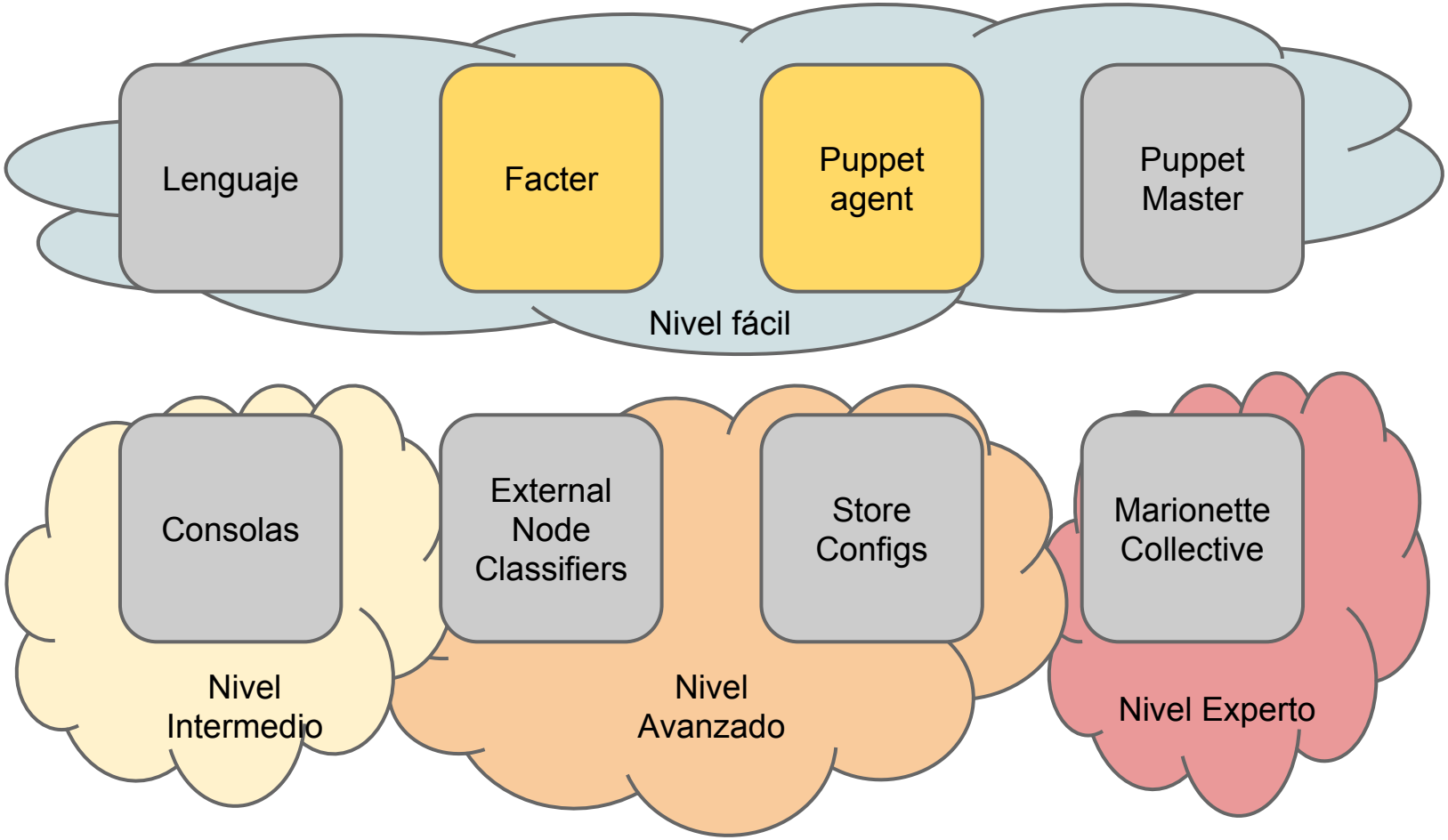
```
stage { "boot": before => Stage["os"] }
stage { "service": }
stage { "os": before => Stage["service"] }
stage { "online": require => Stage["service"] }
```

```
class { 'apt': stage => 'boot' }
class { 'rsyslog': stage => 'os' }
class { 'zabbix': stage => 'os' }
class { 'nginx': conf_role => 'default', vhost_role => 'f00', stage => 'service'}
```

Enough!

Vamos a por la chicha
(y sí, nos saltamos los templates)

Componentes de puppet



Primeros pasos con Puppet

Instalación y práctica

Instalación

Por paquetería

apt-get install puppet

yum install puppet

Verificar las versiones de puppet (2.7.x recomendado)

También se puede descargar el código fuente. Mejor leer la documentación para ello.

Uso: Primeros pasos

```
# facter
```

Uso: Primeros pasos

```
# puppet help
```

Uso: Primeros pasos

puppet describe [-h] [-l] [-p]

Uso: Primeros pasos

```
# puppet resource user root

# pupet resource user

# puppet resource package

# puppet resource service
```

Uso: Primeros pasos

```
# puppet resource package zsh ensure=latest
```

```
# puppet resource user paquito ensure=present
managehome=true shell=/bin/zsh
```

Uso: Primeros pasos

puppet apply init.pp

Con esto ya se puede trabajar.

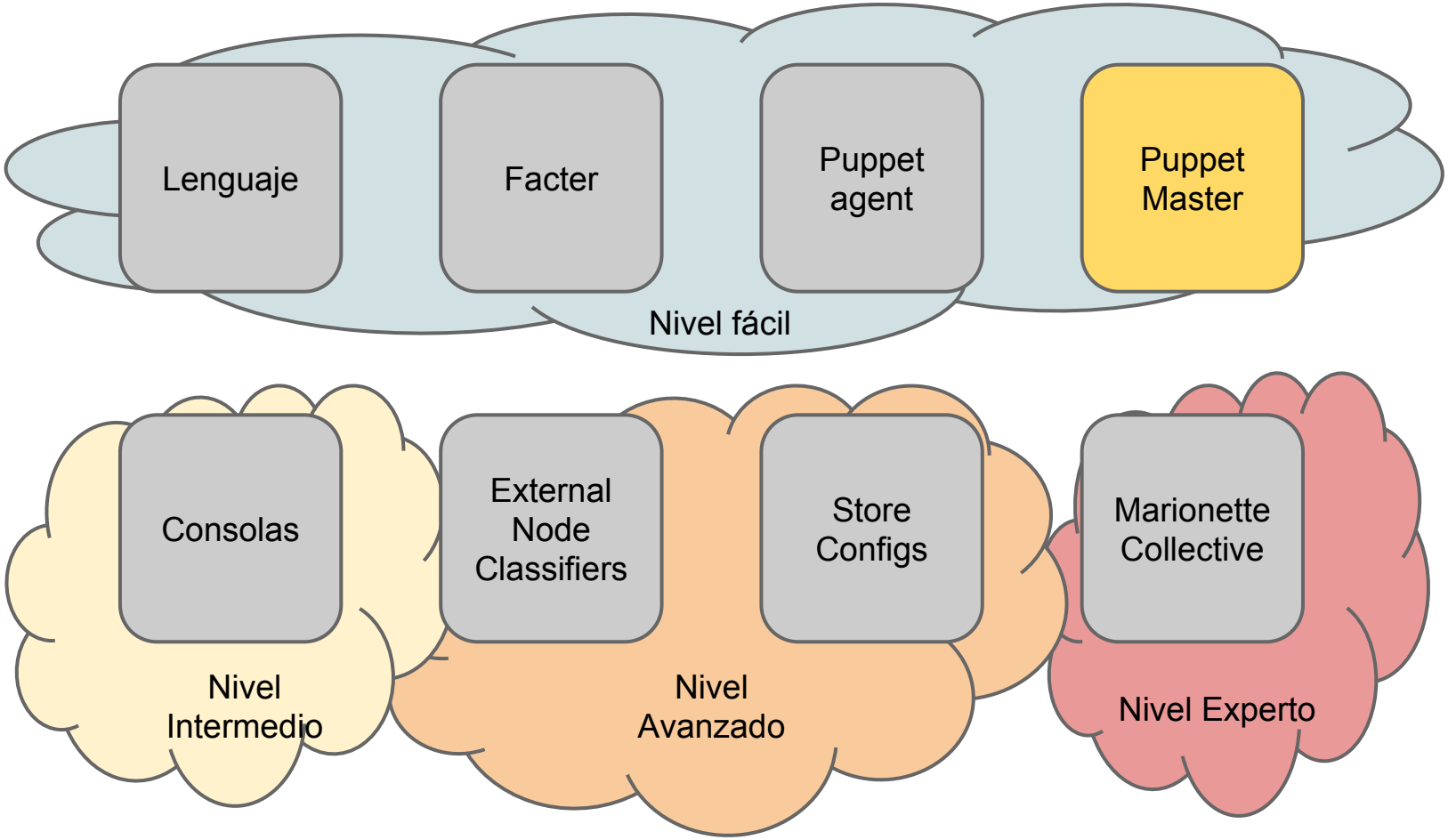
Agrégale un repositorio de código o un servidor de ficheros (SMB o NFS) y una entrada en el crontab, y ya tienes una gestión de la configuración mínima.

```
# vi init.pp
node 'bm.ovh.net' {
  user { 'paquito':
    ensure   => present,
    uid      => 2000,
    gid      => 2000,
    home     => '/home/paquito',
    managehome => true,
    shell    => '/bin/zsh',
    require => [Package['zsh'],Group['fulanito']],
  }
  group { 'fulanito':
    ensure   => present,
    gid      => 2000,
  }
  package { 'zsh':
    ensure   => present,
  }
}
```

Y ahora... Master of Puppets!


Your boss is seeking you to deploy a new version of the website
Nice
It's time ...
puppetize!!
YEEAAAAH!

Componentes de puppet



Puppet Master

- Compila el catálogo y parsea las templates para los nodos.
- Ofrece una PKI completa para emitir certificados SSL a los nodos. Gestiona su autenticación.
- Habilita la transferencia y backup de ficheros (puppet:/// y filebucket)

Puppet Master

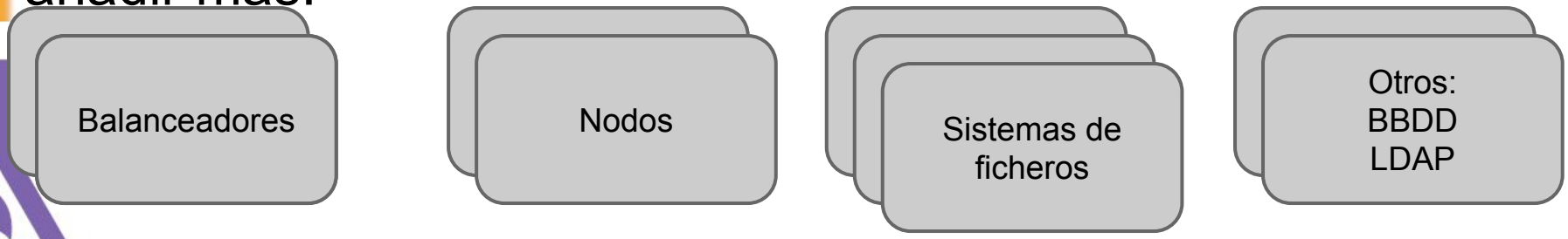
- Suele ser el encargado de servir los ficheros a los agentes.
- Suele ser el encargado de recibir y procesar informes de los nodos.
- Se recomienda utilizar Passenger con NGiNX o Apache y a ser posible Ruby Enterprise para un mejor rendimiento.

Cómo funciona el puppet master?

A nivel plataforma es una aplicación web (puerto HTTP 8140) con autenticación SSL de cliente.

Puppetmaster corre sobre Webrick (un servidor web usado en rails), aunque también puede correr sobre {apache|nginx} + (passenger|mongrel).

Al ser web, es ampliamente escalable: sólo tienes que añadir más:



Instalación del servidor

Por paquetería

```
# apt-get install puppetmaster
# yum install puppet-server
```

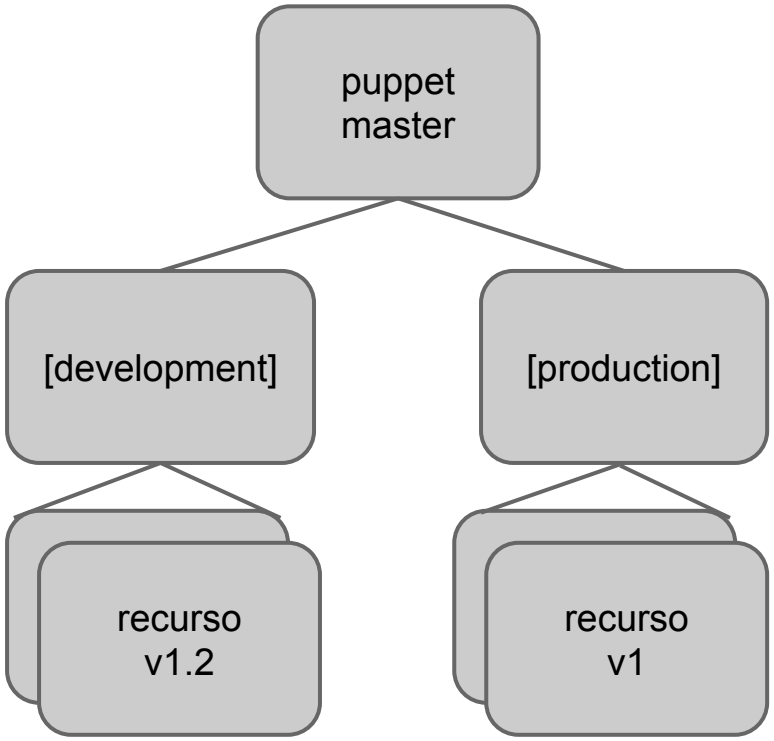
También se puede descargar el código fuente. Mejor leer la documentación para ello.

Puppet Master: Entornos

Concepto:

Entorno es una agrupación de recursos a desplegar en un conjunto de nodos de forma estructural.

- Elemento de la configuración del master
- Permite mantener distintas versiones de recursos en directorios separados
- Permite mantener distintos proyectos separados en el mismo servidor puppet
- Por defecto: production.
- Se aplican con la opción --environment



Puppet Master: CA

Puppet incorpora una CA para reconocer a todos los nodos que gestiona.

=> el nombre del nodo se extrae del certificado

=> el nodo genera el CSR y puppetmaster firma el certificado

=> para que funcione, el cliente y el servidor deben de estar sincronizados (fallo de SSL)

=> un nodo no puede cambiar de nombre una vez configurado. (necesita borrado de .pem)

=> la CA está basada en openssl y es sencilla

Puppet Master: CA

[nodo]#: puppet agent --test --onetime
[puppet]#: puppet cert list [--all]
[puppet]#: puppet cert sign nodo
[nodo]#: puppet agent --test --onetime

Puppet Master: CA

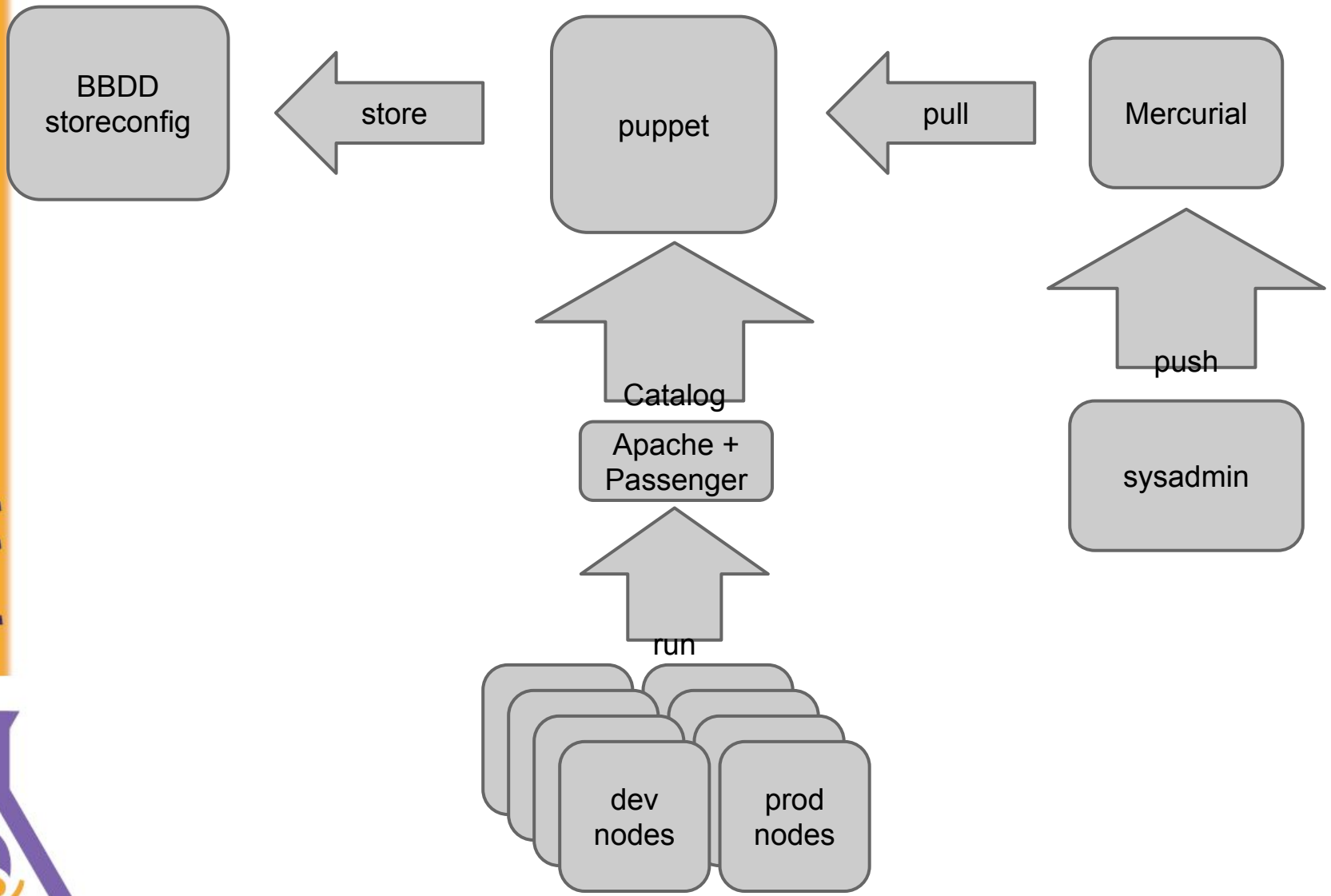
Gestión de la CA

A través de comandos:

A) puppet cert

B) puppetca (es lo mismo, aunque se está deprecando)

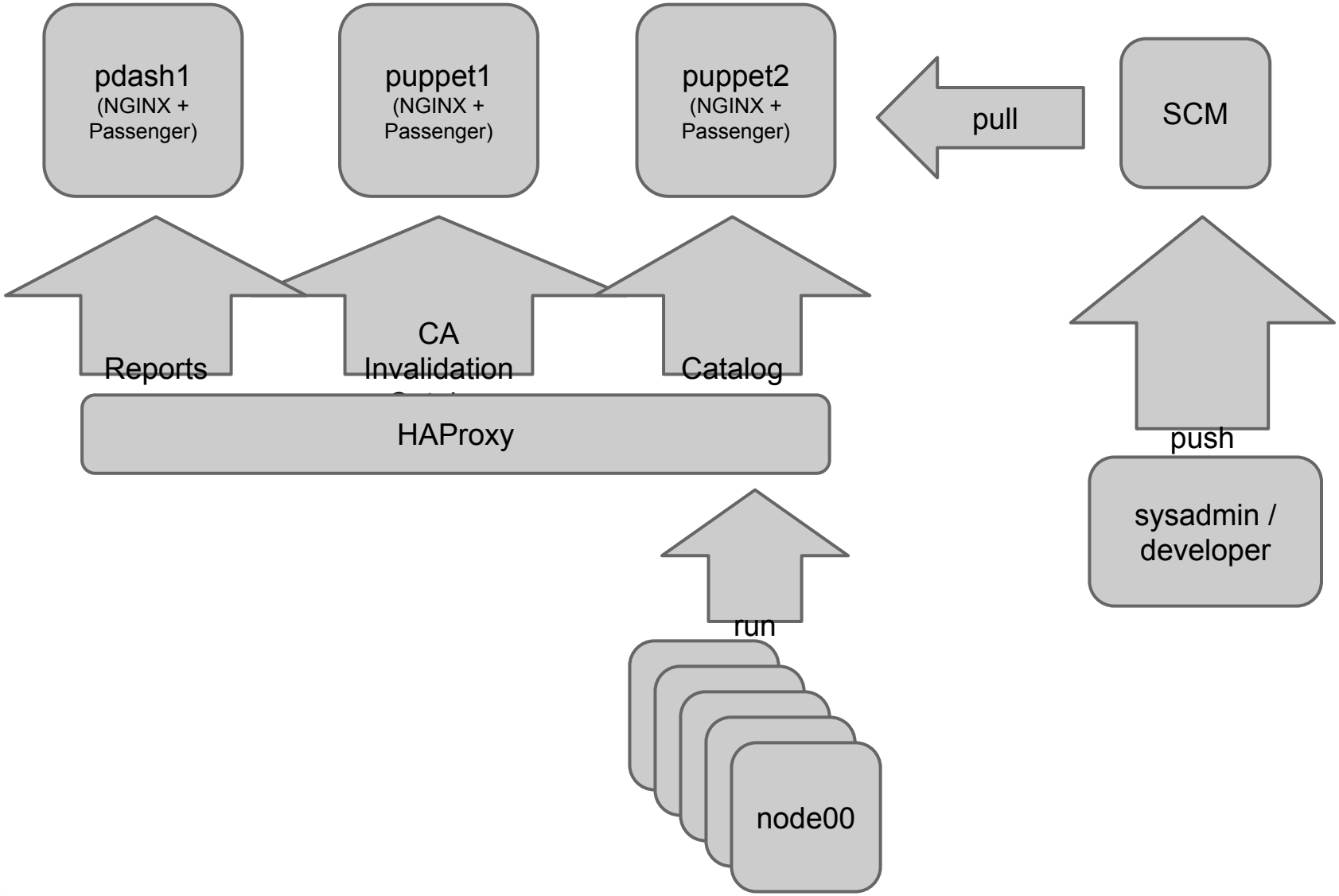
Puppet @ abstra·cc



Puppet Master @ abstra·cc

- SCM: mercurial
- puppet: pull data from SCM
- puppet:
  - CA master
  - RDBMS: store config en MySQL

Puppet @ Tuenti



Puppet Master @ Tuenti

- SCM: svn, git, mercurial
- puppet1/puppet2: hacen pull del SCM. Ambos atienden peticiones de catálogo. Podemos utilizar cuantos queramos.
- puppet1: CA master. los certificados son preferiblemente emitidos por ella y replicados a otros masters.
- puppet1: servicio de invalidación de CA, igual que el anterior.
- pdash1: reports (interfaz web y trabajos batch)

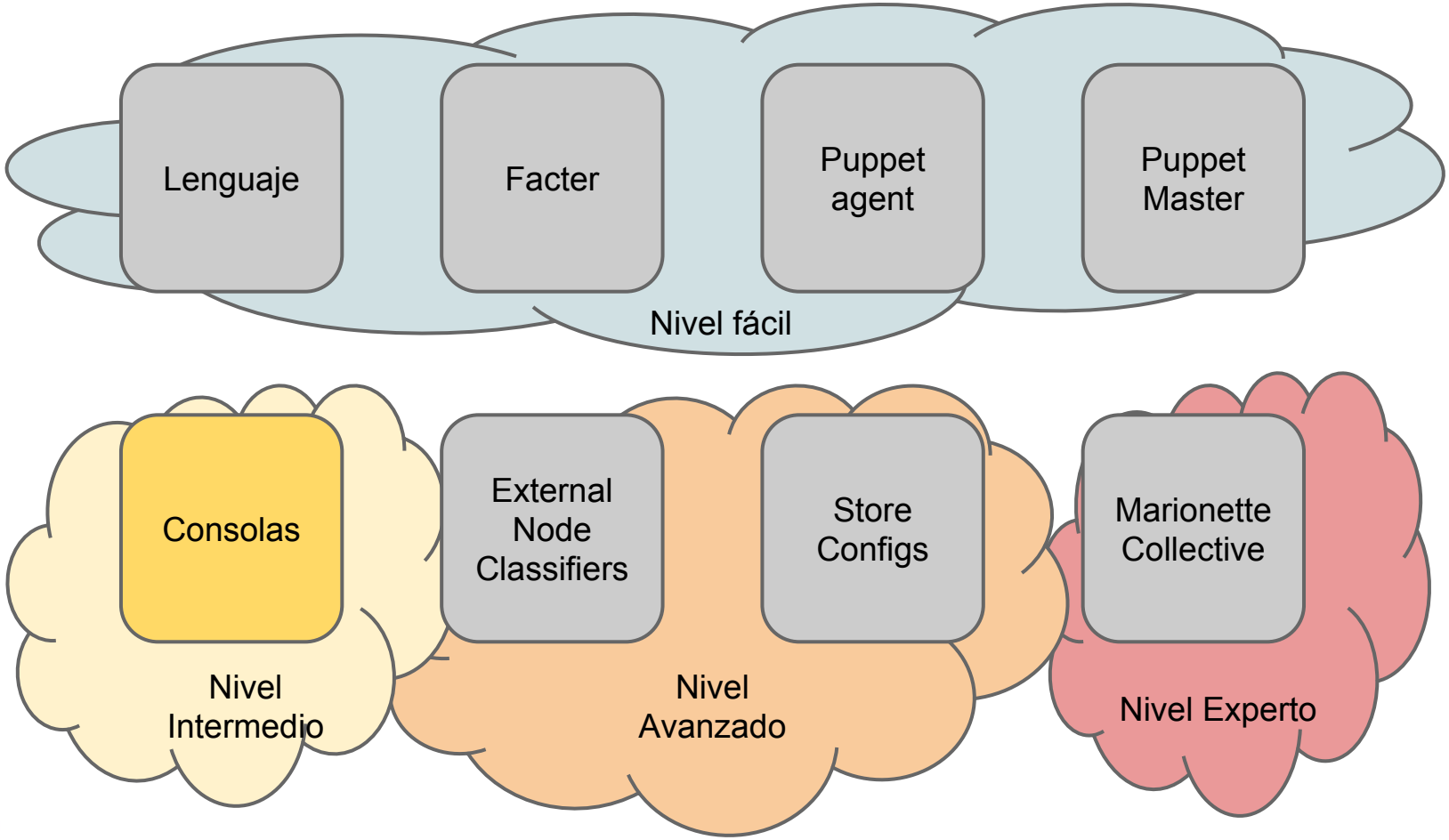
Y hasta aquí por hoy...

Nos quedan todavía muchas cosas que contar, pero vendrán más adelante.

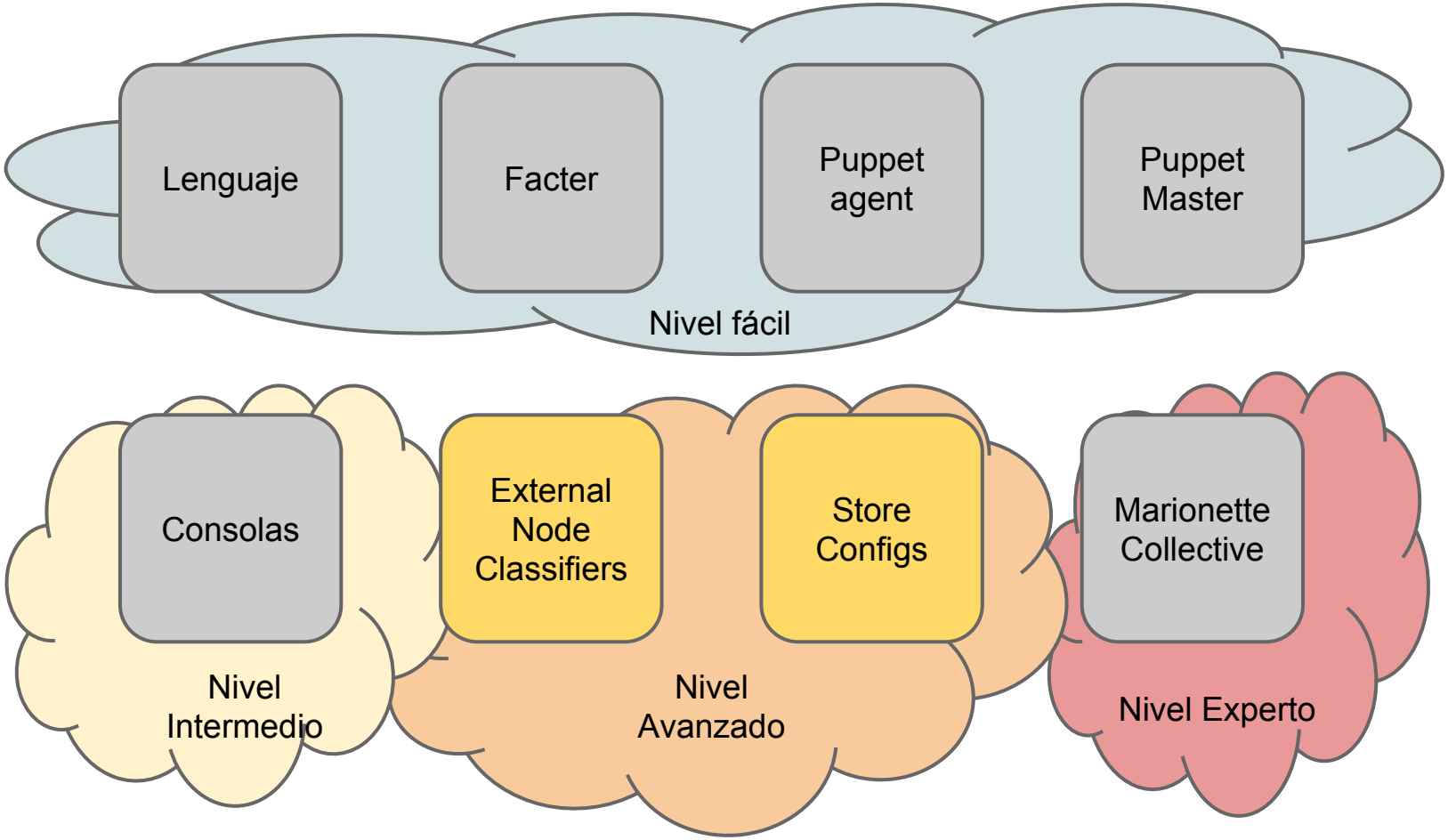
Para la próxima charla/taller

Cosas que nos gustaría ver.

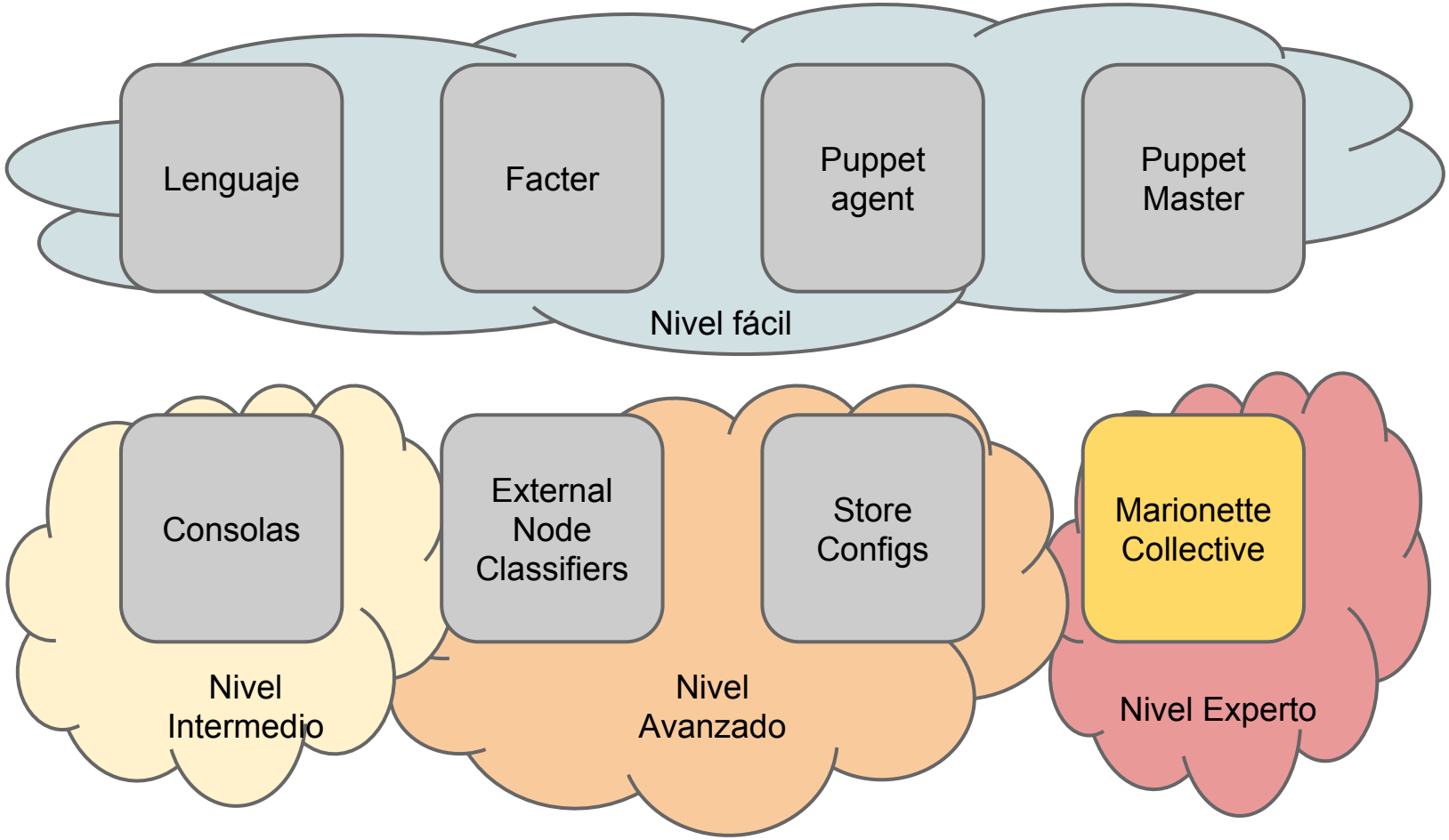
Componentes de puppet



Componentes de puppet



Componentes de puppet



Referencias

Libros
Enlaces
Multimedia

Referencias

- [URL] Puppet Labs: http://docs.puppetlabs.com/
  - puppet.pdf y learning puppet vm
- [Libro] Pro Puppet: http://goo.gl/j2yat
- [Grupos] Puppet users: http://groups.google.com/group/puppet-users
- [Presentación] PyCon2011: http://rcrowley.org/talks/pycon-2011/
- [Presentación] Extending Puppet: http://rcrowley.org/talks/sv-puppet-2011-01-11
- [Utilidad] Gepetto: https://github.com/cloudsmith/geppetto