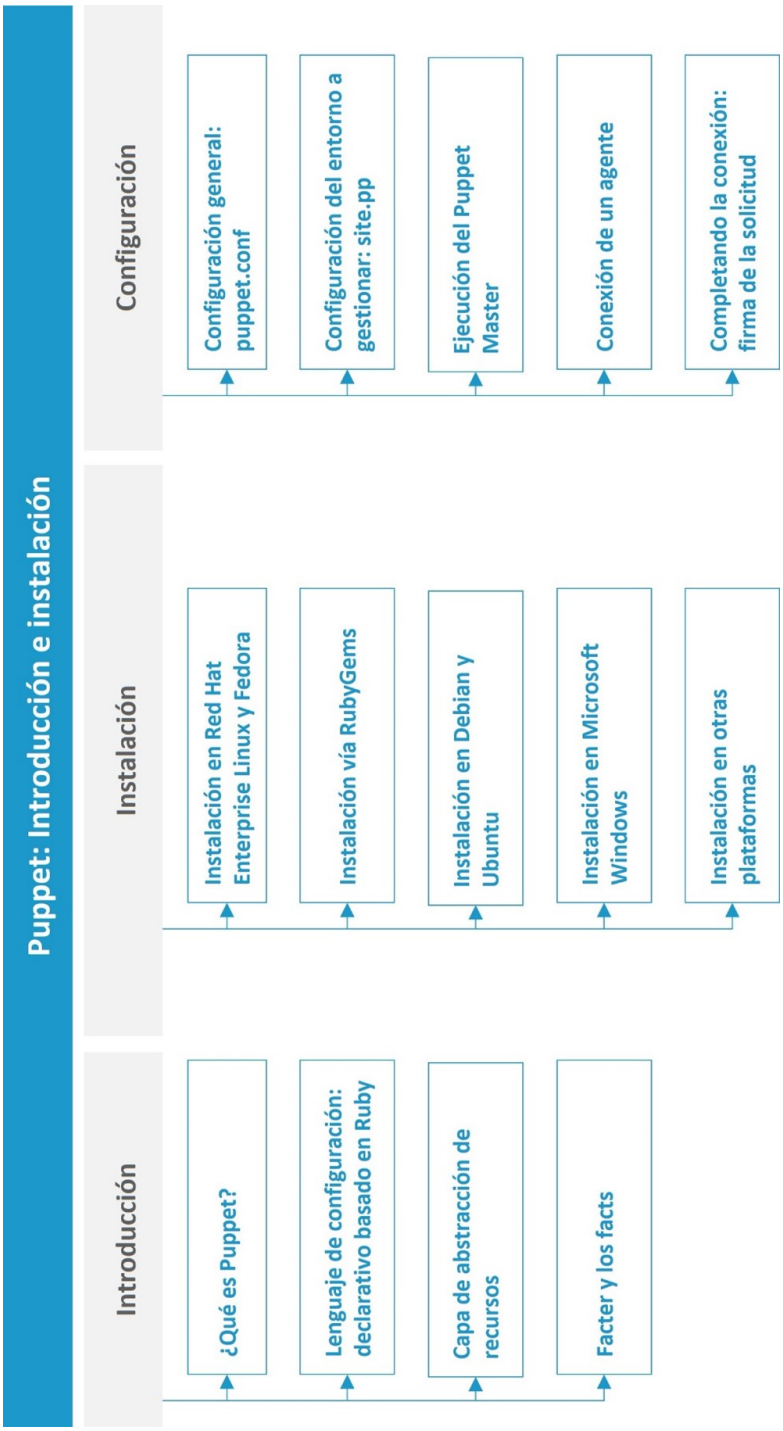
Herramientas de Automatización de Despliegues

Tema 1. Puppet. Introducción e instalación

Índice



1.1. Introducción y objetivos

Puppet es un *framework* de código abierto que incluye a su vez un conjunto de herramientas para la gestión de la configuración de los sistemas informáticos. Puppet proporciona una entrega y funcionamiento estándar de *software* sin importar dónde esté corriendo.

Con el enfoque de Puppet, el usuario define cómo desea que funcionen y se organicen sus aplicaciones e infraestructura, utilizando un lenguaje común y de fácil lectura. Se puede compartir, probar y aplicar los cambios necesarios a través del *datacenter*, con absoluta visibilidad en cada uno de los pasos, así como también obtener informes detallados para tomar decisiones acertadas y documentadas. En este tema vamos a ver cómo utilizar Puppet para administrar la configuración de servidores.

Los objetivos que se pretenden conseguir en este tema son los siguientes:

- Conocer qué es y cómo funciona Puppet.
- Conocer el lenguaje de configuración y modelo de abstracción.
- Interactuar con la herramienta de inventario Facter.
- Instalar Puppet.
- Configurar Puppet y el entorno.

1.2. ¿Qué es Puppet?

Puppet es una herramienta de gestión de la configuración escrita en lenguaje Ruby con licencia de código abierto GPLv2, aunque también dispone de una versión comercial denominada Enterprise. Se ejecuta habitualmente con un modelo cliente-servidor, aunque también se puede ejecutar en modo *stand-alone* (independiente).

Fue desarrollado por el ingeniero Luke Kanies en la compañía Puppet Labs (anteriormente llamada Reductive Labs). Kanies ha estado trabajando con sistemas Unix y con la administración de sistemas desde el año 1997; y ha desarrollado Puppet por la insatisfacción que le provocaban las herramientas de gestión de la configuración existentes en aquel entonces. En el año 2005 Kanies fundó la empresa Puppet Labs, para el desarrollo de herramientas de automatización de código abierto. Al cabo de poco tiempo, esta empresa lanzó Puppet, su producto principal.

Puppet es capaz de gestionar la configuración de servidores basados en las plataformas UNIX (incluyendo OSX) y Linux, así como servidores basados en Microsoft Windows.

Asimismo, es utilizado habitualmente para la gestión a lo largo de todo el ciclo de vida de las máquinas: comenzando desde la creación y primera instalación, continuando por las actualizaciones y el mantenimiento, y finalizando, al acabar su vida útil, migrando los servicios que proporcionaba a otros sistemas. El diseño de Puppet está pensado para estar en continua interacción con las máquinas que gestiona, al contrario que otras herramientas de aprovisionamiento que únicamente se encargan de la etapa de construcción de las máquinas.

Puppet tiene un modelo de funcionamiento sencillo, fácil de entender y de aplicar, que se compone de tres componentes básicos:

- Despliegue.

- Lenguaje de configuración y capa de abstracción de recursos.
- Capa transaccional.

Despliegue

El despliegue de Puppet sigue habitualmente un **modelo cliente-servidor.** El servidor recibe el nombre de **Puppet Master,** mientras que el cliente de Puppet que se ejecuta en las máquinas (*hosts*) que se van a gestionar se llama **agente** y el propio *host* se define como un **nodo.**

El proceso de Puppet Master se ejecuta como un *daemon* en el servidor, donde se almacena la configuración necesaria de todo el entorno gestionado. Los agentes se identifican con el servidor Puppet Master al conectarse y utilizan el estándar SSL para establecer un canal de comunicación cifrado, a través del cual se obtiene la configuración que se va a aplicar.

Cabe destacar que, si el agente no tiene disponible una configuración de Puppet, o ya tiene la configuración requerida, Puppet no hará ningún cambio. Solo realizará cambios en la máquina si, para alcanzar la configuración requerida, es necesario hacerlos, proceso que se conoce como **ejecución de la configuración.**

Cada uno de los agentes Puppet puede estar ejecutándose como un proceso *daemon* (demonio), mediante mecanismos tales como cron, o incluso el agente puede ejecutarse manualmente cuando así se requiera. Lo más habitual es configurar el agente para que se ejecute como un *daemon*, conectándose periódicamente con el servidor para determinar si su configuración está actualizada o, en caso contrario, para descargar y aplicar cualquier cambio requerido en la configuración.

Sin embargo, muchas personas creen que ejecutar el agente Puppet a través de un planificador como cron o manualmente es más adecuado para sus necesidades. El

agente de Puppet consultará periódicamente el Puppet Master por si hubiera cambios en su configuración o algo nuevo que aplicar. La periodicidad por defecto de esta consulta es de 30 minutos, pero este valor puede cambiarse según sean los requisitos del entorno.

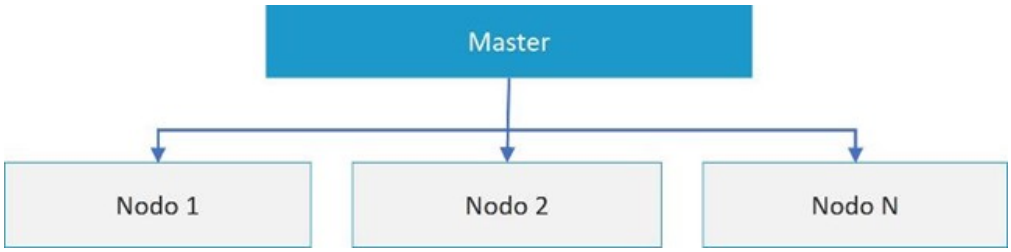


Figura 1. Master y nodos.

También existen otros modelos de despliegue. Se puede ejecutar Puppet en modo independiente, sin contar con una conexión a un Master. La configuración debe encontrarse instalada localmente en la máquina y se ejecuta el binario de Puppet que comprobará y aplicará los cambios necesarios para alcanzar esa configuración.

1.3. Lenguaje de configuración y capa de abstracción de recursos

Puppet cuenta con un lenguaje declarativo que sirve para definir los distintos elementos de la configuración, que se denominan **recursos.** Este aspecto declarativo es una característica importante de Puppet con respecto a otras herramientas de configuración. Mediante el lenguaje se declara el estado deseado de la configuración, tal como indicar que un determinado paquete debería estar instalado, o un determinado servicio no debería estar ejecutándose.

En cambio, otras herramientas utilizadas para la gestión de la configuración, tales como Shell o Perl, son de naturaleza imperativa o procedimental: indican las operaciones concretas a realizar, en lugar de declarar el estado final deseado. La mayoría de los scripts personalizados que se implementan para automatizar la configuración de un sistema son un ejemplo del modelo imperativo.

Esto significa que los usuarios de Puppet simplemente declaran el estado deseado de sus *hosts*: qué paquetes deben ser instalados, qué servicios deberían correr, etc. En otras palabras, Puppet se encargará de lograr el estado deseado y los administradores de sistemas abstraerán la configuración de los *hosts* a base de definir los recursos.

Lenguaje de configuración

Vamos a ver un ejemplo práctico de lo que significa realmente que Puppet utiliza un lenguaje declarativo. Para ello, vamos a considerar lo que implicaría la instalación de la aplicación «vim» en diferentes entornos que cuentan con diferentes sistemas operativos, tales como Red Hat Enterprise Linux (RHEL), Ubuntu y Solaris.

Los pasos a continuación indican lo que haría falta implementar en un script para realizar la instalación:

- Conectar a las máquinas necesarias, con sus usuarios y contraseñas o claves correspondientes.
- Comprobar si la aplicación «`vim`» está ya instalada.
- En caso de no estarlo, usar el comando adecuado en cada sistema operativo para instalar «vim», esto es, utilizar yum en Red Hat o apt-get en caso de Ubuntu.
- Informar del resultado al usuario para que sepa si la operación se ha completado con éxito.

Por su parte, Puppet se aproxima a este proceso de manera muy diferente. Lo primero que haremos será definir la configuración de recursos para el paquete «`vim`». Un recurso está compuesto por un **tipo,** que indica la clase de recurso que se está gestionando, como, por ejemplo, paquetes, servicios o ficheros; un **título,** que especifica el nombre del recurso, y una serie de **atributos,** que son los valores que definen el estado del recurso, como, por ejemplo, si el fichero debe estar presente o se debe borrar.

```
package {"vim":
 ensure => present,
}
```

En nuestro ejemplo, el recurso define que el paquete denominado «`vim`» debería estar instalado.

Se construye así:

```
type {title:
 attribute => value,
}
```

Como se puede ver en la primera secuencia de código, el tipo de recurso es de tipo paquete (`package`). Puppet incorpora una serie de tipos de recursos de manera predeterminada incluyendo tipos para administrar archivos, servicios, paquetes y trabajos de cron, entre otros.

A continuación, se especifica el título del recurso, que en nuestro caso se trata del nombre del paquete que queremos instalar: «`vim`». Para utilizar el tipo y el título del recurso como referencia, Puppet nos permite unir el tipo de recurso con la primera letra en mayúscula seguido del nombre entre conchetes, como, por ejemplo: `Package["vim"]`.

Finalmente, hemos incluido un único atributo `ensure`, con el valor: `present`. Los atributos que especifiquemos le dicen a Puppet el estado deseado del recurso de configuración. Cada tipo dispone de una serie de atributos para configurarse. En nuestro caso, el atributo `ensure`indica el estado deseado del paquete: instalado, desinstalado, etc. El valor `present` especifica que queremos que el paquete esté instalado, por lo que se procederá a instalarlo si no lo está ya. Para indicar que no queremos que el paquete esté instalado, tendríamos que cambiar el valor de este atributo a `absent`, lo que lo desinstalaría en caso de estar instalado previamente.

Capa de abstracción de recursos

Cuando se hayan definido todos los recursos, Puppet será el que se encargue de gestionar los detalles de la configuración del recurso una vez que se conecten los agentes. Se encargará del «cómo», ya que conoce cómo cada tipo de recurso es

gestionado en las distintas plataformas y sistemas operativos. Los tipos de recursos constan de distintos proveedores en función de cada plataforma. En el caso de «vim», el proveedor conoce el cómo de la gestión de paquetes en cada plataforma, utilizando la herramienta de gestión de paquetes correspondiente en cada caso. Para el tipo de recurso `package`, Puppet tiene más de veinte proveedores distintos, que utilizan distintos gestores de paquetes, tales como:

Cuando un agente se conecta, Puppet utiliza una herramienta denominada Facter (que veremos en el apartado siguiente) para devolver información sobre ese agente, incluyendo el sistema operativo que está ejecutando. Puppet se encarga de elegir el proveedor del tipo `package`que se corresponde con ese sistema operativo y será el proveedor el que verifique si el paquete «vim» se encuentra ya instalado. En Red Hat ejecutaría `yum`, por ejemplo, mientras que en Ubuntu se ejecutaría `apt`, y en Solaris, el comando `pkg`. Si el paquete no está ya instalado, Puppet se encargará de instalarlo a través del proveedor, pero si ya se encuentra instalado, no realizará ninguna acción. Puppet reportará al Puppet Master el resultado de la aplicación del recurso o, en caso de haberse producido un error, la información de este.

Capa transaccional

La capa transaccional es el motor de Puppet. Se conoce como transacción en Puppet al proceso de configuración de cada máquina, esto es:

- Interpretación y compilación de la configuración.
- Transmisión de la configuración compilada a cada agente.
- Aplicación de la configuración en la máquina del agente.
- Informe del resultado de la ejecución al Puppet Master.

Lo primero que hace Puppet es el análisis de la configuración para calcular cómo debería aplicarse en el agente, creando para ello un grafo que incluye los distintos

recursos y las relaciones entre ellos, así como con cada agente. Esto permite a Puppet establecer en qué orden aplicar cada recurso al *host*, basándose en las relaciones que se crean.

Acto seguido, con los recursos obtenidos previamente, Puppet compila un catálogo de estos para cada agente. Este catálogo es lo que se envía a cada máquina para que el agente de Puppet lo aplique sobre su máquina. Con resultado de esta ejecución, se genera un informe, que es lo que se envía posteriormente de vuelta al Puppet Master.

La capa transaccional permite crear configuraciones y aplicarlas repetidamente en el *host*. A esto lo llama Puppet **idempotencia,** que significa que múltiples aplicaciones del mismo conjunto de operaciones darán lugar a los mismos resultados. La configuración de Puppet se puede aplicar varias veces con seguridad, ya que vamos a obtener el mismo resultado en la máquina, lo que asegura que la configuración va a ser consistente.

Sin embargo, Puppet no es totalmente transaccional: las transacciones no se registran (más allá de los registros informativos) y, por lo tanto, no se pueden deshacer como se hace con algunas bases de datos. Lo que sí podemos hacer es configurar estas transacciones en modo NOOP (del inglés *no operation mode*), lo que nos posibilita el probar la ejecución de la configuración sin realmente realizar ningún cambio.

1.4. Facter y los facts

Facter es la herramienta que incluye Puppet para obtener la información del inventario del sistema. Esta herramienta devuelve los denominados *facts* (hechos), que es información relativa a cada máquina, como su nombre de *host*, dirección o direcciones IP, sistema operativo y su versión; y otra gran cantidad de datos de configuración. Estos datos se recopilan cuando se ejecuta el agente, y son posteriormente enviados al Puppet Master, creándose automáticamente variables para representarlos, lo que los hace disponibles para su uso en Puppet.

Para ver los *facts* que están disponibles en cada máquina, podemos ejecutar directamente en ella el comando `facter` desde la línea de comandos. Cada uno de los facts se muestra en forma de par «`clave => valor`» (`key => value pair`), como el ejemplo siguiente:

```
operatingsystem => Ubuntu
```

```
ipaddress => 10.0.0.10
```

Por consiguiente, se pueden utilizar las variables correspondientes para configurar individualmente cada máquina, como por ejemplo la dirección IP, que podemos utilizar para configurar una red en la máquina. Al combinar estas variables de *facts* con la configuración de Puppet, podemos personalizar la configuración en cada una de las máquinas, permitiendo escribir recursos genéricos, como la configuración de red, y personalizarlos con datos de los agentes.

Facter también sirve de ayuda a Puppet para entender cómo debe manejar los recursos de un agente individual, como hemos visto en el apartado anterior. Por ejemplo, cuando Facter detecta que el sistema operativo de la máquina es Ubuntu, se lo dice a Puppet, que entonces sabe que debe usar `apt` para instalar los paquetes en la máquina del agente. Facter permite extensiones, con lo que se puede añadir

información personalizada y específica sobre las máquinas.

1.5. Instalación de Puppet

Puppet está disponible para instalar en multitud de plataformas, entre las cuales destacamos:

- Red Hat Enterprise Linux, CentOS, Fedora y Oracle Enterprise Linux.
- Debian y Ubuntu.
- Mandrake y Mandriva.
- Solaris y OpenSolaris.
- MacOS X y MacOS X Server.
- BSD.
- AIX.
- HP-UX.
- *Hosts* de Microsoft Windows (únicamente el agente y con recursos de archivo con soporte limitado).

En estas plataformas, Puppet gestiona una variedad de elementos de configuración, incluyendo:

- Archivos: mediante el recurso `file`
- Servicios: `service`
- Paquetes: `package`
- Usuarios: `user`
- Grupos: `group`

- Trabajos Cron (Cron jobs): `cron`
- Comandos shell: `exec`
- SSH Keys: `sshkey, ssh_authorized_key`

Tanto las instalaciones del agente Puppet como las del servidor Puppet Master son muy semejantes, pero la mayor parte de los sistemas operativos y sus correspondientes gestores de paquetes los distribuyen en paquetes separados. En algunos sistemas operativos y de distribución será necesario instalar Ruby y sus bibliotecas e incluso algunos paquetes adicionales, aunque la mayor parte de los gestores de paquetes incorporan los paquetes necesarios para instalar Puppet o Facter como prerrequisito, tales como Ruby, por lo que se instalarán automáticamente junto con el propio Puppet o Facter.

Instalación en Red Hat Enterprise Linux y Fedora

En Red Hat Enterprise Linux (RHEL) y derivados basados en Red Hat, como CentOS, es necesario instalar algunos requisitos previos: el lenguaje de programación Ruby, bibliotecas de Ruby y la biblioteca Shadow de Ruby, para permitir a Puppet gestionar usuarios y grupos, lo cual se puede hacer muy fácilmente a través del propio gestor de paquetes de Red Hat: `yum`.

```
# yum install ruby ruby-libs ruby-shadow
```

Acto seguido, para poder instalar la versión más actualizada de Puppet, tendrás que añadir el repositorio a la máquina, para luego instalar desde ese repositorio los paquetes propios de Puppet. Puedes añadir el repositorio EPEL incluyendo el siguiente comando RPM (.rpm Administrador de paquetes).

```
# rpm -Uvh https://yum.puppet.com/puppet6/puppet6-release-el-
7.noarch.rpm
```

En el Puppet Master es necesario instalar el paquete `puppetserver` del repositorio

anteriormente configurado:

```
# yum install puppetser<span style="font-size: 12.8px;">r</span>
<span style="font-size: 12.8px;">ve</span>
```

Para instalar un agente en una máquina RHEL, solo se necesitan instalar los prerrequisitos y el paquete del agente:

```
# yum install puppet-agent
```

Instalación vía RubyGems

Tal como la mayor parte de las aplicaciones hechas con Ruby, también es posible instalar las herramientas Puppet y Facter mediante RubyGems. De cara a poder hacerlo de esta manera, necesitas primero instalar el propio Ruby y el paquete `RubyGems` adecuado a tu sistema operativo, que, para los sistemas basados en Fedora (RHEL, CentOS, Fedora), SUSE (SUSE/SLES) y Debian (Debian y Ubuntu), el paquete se llama `rubygems`. Una vez instalado este paquete, el comando para la gestión de gemas (*gem*) estará accesible para su uso, por lo que puedes utilizarlo para instalar Puppet y Facter:

```
# gem install puppet facter
```

Instalación en Debian y Ubuntu

En Debian y Ubuntu también tenemos que configurar el repositorio de origen para la instalación de Puppet:

```
# wget https://apt.puppetlabs.com/puppet6-release-bionic.deb
```

```
# dpkg -i puppet6-release-bionic.deb
```

A continuación, puedes instalar los paquetes necesarios para Puppet. En el Puppet Master instalamos la parte servidora:

```
# apt-get install puppetserver
```

En las máquinas donde necesitemos el agente:

```
# apt-get install puppet-agent
```

Nota: Tanto la instalación de los paquetes de `puppet-agent` como los de `puppetmaster` instalarán también los prerrequisitos necesarios, como Ruby, si no se encuentra ya instalado.

Instalación en Microsoft Windows

Puedes instalar el agente de Puppet en máquinas Windows, para poder gestionarlo desde tu Puppet Master.

Se puede realizar una instalación del paquete MSI desde la interfaz gráfica, pero aquí vamos a incluir los comandos necesarios para instalarlo mediante la interfaz de comandos de Windows, lo que te permite personalizar `puppet.conf`, los atributos CSR y otras propiedades del agente. Una vez descargado el paquete MSI, ejecutamos:

```
msiexec /qn /norestart /i <PACKAGE_NAME>.msi
```

Puedes incluir `/l*v <LOG_FILE>.txt` para que se genere el log de la instalación en el fichero indicado.

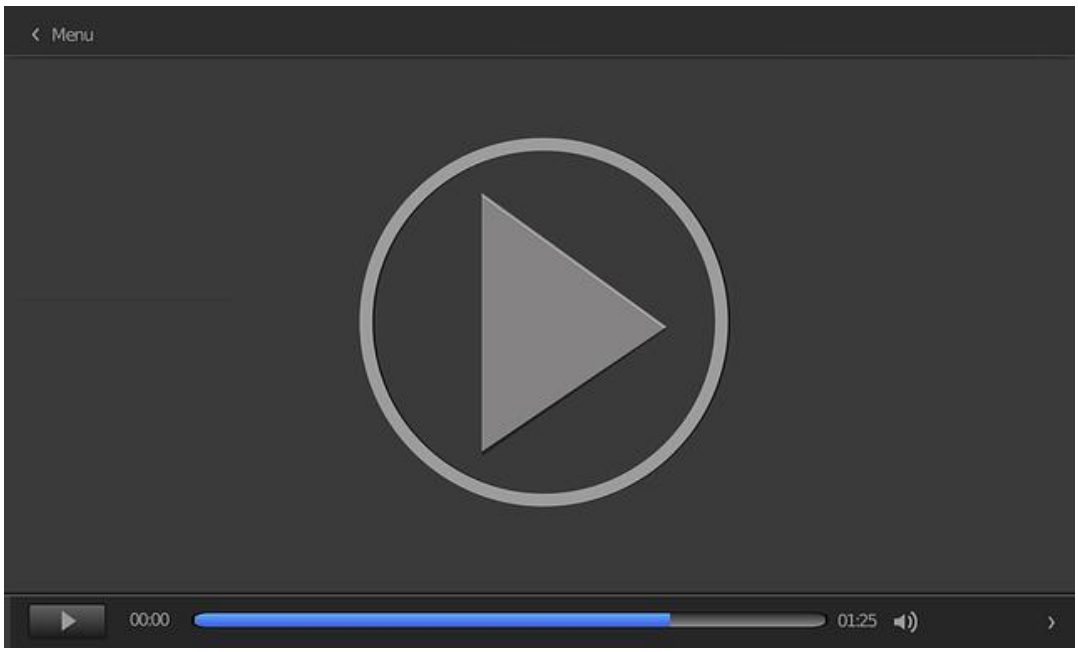
Instalación en otras plataformas

Hasta aquí hemos visto cómo instalar Puppet en algunas de las plataformas más populares. Pero, además de las que hemos visto, se puede instalar en muchas otras, tales como:

- MacOS X vía http://downloads.puppetlabs.com/mac/
- Solaris vía Blastwave.
- SLES/OpenSuSE vía http://software.opensuse.org/
- Mandrake y Mandriva vía el repositorio Mandriva contrib.
- FreeBSD vía ports tree.

Los paquetes de código fuente de Puppet también contienen algunos elementos en el directorio conf, como puede ser el archivo de especificaciones RPM y scripts de construcción de OS X, que pueden permitir al usuario crear paquetes propios que sean compatibles con el sistema operativo. Ahora que hemos instalado Puppet en alguna de estas plataformas, ya podemos empezar a configurarlo.

En el siguiente vídeo se explica con detalle la instalación de Puppet:



Accede al vídeo:https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=1b720133-63cb-48ac-8efe-abb600b32d56

1.6. Configuración de Puppet

Vamos a describir la configuración del Puppet Master, que será nuestro servidor de configuración, por lo que empezaremos haciendo un repaso a los ficheros de configuración, la configuración de red y el acceso mediante *firewall*, y veremos por último cómo iniciar el Puppet Master. Utilizaremos Puppet en el modo habitual de cliente-servidor, por lo que el Puppet Master contendrá toda la información de la configuración, mientras que los agentes de Puppet se conectarán a través de SSL y descargarán la configuración que les corresponda.

El directorio `/etc/puppet` es donde se guarda la configuración del Puppet Master, en la mayor parte de las plataformas.

El fichero principal de configuración de Puppet se puede encontrar en la siguiente ruta: `/etc/puppet/puppet.conf`. Este fichero se crea habitualmente durante la instalación de Puppet, pero, si no es el caso, el siguiente comando nos permite crear el fichero:

```
# puppetmasterd --genconfig > puppet.conf
```

Nota: Aquí asumimos que el directorio `/etc/`es el que está usando el sistema operativo para almacenar los archivos de configuración, como ocurre en la mayoría de las distribuciones Unix/Linux. Si estás en otra plataforma que no usa esta ruta, como Microsoft Windows, utiliza la ubicación que corresponda para el fichero `puppet.conf`.

El fichero `puppet.conf` tiene una estructura muy similar a los ficheros de configuración de formato INI, ya que está dividido en diferentes secciones. Cada una de las secciones se centra en configurar un aspecto concreto de Puppet. Por ejemplo, tenemos la sección del agente `[agent]` para establecer la configuración del agente y la sección de Puppet Master `[master]` para configura el servidor Master.

También se encuentra una sección llamada `[main]` para configuración general. En esta sección se configuran opciones generales para todos los componentes de Puppet.

Por ahora, no añadiremos en el fichero de `puppet.conf` nada más que una entrada, `certname`, para especificar el nombre del Puppet Master. Para ello, tenemos que añadir la entrada `certname` a la sección `[master]`, que habrá que crearla en caso de que no exista todavía:

```
[master]
```

```
certname=puppet.ejemplo.edu
```

Debes reemplazar puppet.ejemplo.edu con el nombre correspondiente al dominio completo de tu máquina.

Al añadir esta opción certname y especificar el nombre de dominio completo, estamos facilitando la solución posterior de posibles problemas relacionados con los certificados. Asimismo, es recomendable que se cree una entrada CNAME DNS para el servidor de Puppet, en nuestro caso, `puppet.ejemplo.edu` y lo incluyas en tu configuración DNS, así como a tu archivo `/etc/hosts`, que quedaría similar a:

```
127.0.0.1 localhost
```

```
192.168.0.1 puppet.ejemplo.edu puppet
```

Ahora que hemos realizado la configuración de DNS adecuada para Puppet, vamos a seguir con el archivo `site.pp` que debemos añadir, en el que se incluyen los elementos de configuración básicos que vamos a administrar.

El archivo site.pp

El fichero `site.pp` es el que contiene las máquinas que se van a gestionar y la

configuración que debe aplicar a estas. Este archivo suele almacenarse en el subdirectorio manifests dentro del directorio `/etc/puppet/`.

Nota: los ficheros que Puppet maneja para la definición de la configuración de los recursos se denominan ficheros `manifests` (manifiestos). Un fichero manifest de Puppet tiene la extensión .pp.

Puppet habitualmente crea este directorio y el archivo `site.pp` cuando se instala, pero, si no existen aún, créalos manualmente, ya que, si no encuentra este fichero, Puppet no arrancará:

```
mkdir /etc/puppet/manifests
```

```
touch /etc/puppet/manifests/site.pp
```

Otra consideración que hay que tener en cuenta es que se puede sobrescribir la ubicación y el nombre del fichero `manifest` mediante el archivo `site.pp` usando la opción de configuración y `manifest`, respectivamente. Ambas opciones se pueden establecer en el fichero `puppet.conf`, bajo la sección `[master]`.

El servidor Puppet Master se ejecuta por defecto escuchando el puerto TCP 8140, por lo que el *firewall* que esté configurado en la máquina del Puppet Master debe tener este puerto abierto, así como cualquier otro *firewall* o dispositivo de red que intervenga entre los clientes y el Master, ya que los clientes deben ser capaces de acceder y conectar a través de ese puerto. Si por configuración se cambiase el puerto de escucha por defecto del Master, habría que hacer lo propio con dicho puerto. El siguiente ejemplo muestra cómo establecer la regla de *firewall* necesaria en el cortafuegos Netfilter para permitir conexiones a través del puerto por defecto:

```
-A INPUT -p tcp -m state --state NEW --dport 8140 -j ACCEPT
```

Esta línea establece la regla que permite el acceso a través del puerto TCP 8140

desde cualquier origen, aunque sería conveniente en la medida de lo posible solo permitir el tráfico entrante desde las redes donde residan las máquinas con los agentes que deben conectar a nuestro Puppet Master, tal como:

```
-A INPUT -p tcp -m state --state NEW -s 192.168.0.0/24 --dport 8140
-j ACCEPT
```

Con esta regla restringimos los orígenes que pueden acceder a través del puerto 8140 a las máquinas cuya IP está dentro de la subred 192.168.0.0/24.

Inicio de Puppet Master

El Puppet Master puede iniciarse en la mayoría de las distribuciones de Linux mediante un *script* init de inicio de servicio. En Red Hat, ejecutaríamos el script de inicio con el comando service de esta forma:

```
# service puppetmaster start
```

En Debian o Ubuntu, lo ejecutamos con el comando `invoke-rc.d`:

```
# invoke-rc.d puppetmaster start
```

Si inicias el *Daemon,* se iniciará el entorno Puppet, se creará una autoridad de certificados locales *(local Certificate Authority)*, certificados y claves para el master, y se abrirá el socket de red adecuado para recibir las conexiones del cliente. Se pueden ver de los certificados e información SSL de Puppet en el directorio `/etc/puppet/ssl`.

```
# ls –l /etc/puppet/ssl
```

```
drwxrwx--- 5 puppet puppet 4096 2009-11-16 22:36 ca
```

```
drwxr-xr-x 2 puppet root 4096 2009-11-16 22:36
certificate_requests
```

```
drwxr-xr-x 2 puppet root 4096 2009-11-16 22:36 certs
```

```
-rw-r--r-- 1 puppet root 361 2009-11-16 22:36 crl.pem
```

```
drwxr-x--- 2 puppet root 4096 2009-11-16 22:36 private
```

```
drwxr-x--- 2 puppet root 4096 2009-11-16 22:36 private_keys
```

```
drwxr-xr-x 2 puppet root 4096 2009-11-16 22:36 public_keys
```

El directorio en el Master contiene la autoridad de certificados *(Certificate Authority),* las solicitudes de certificado de los clientes, un certificado para el Master y certificados para todos sus clientes.

Nota: se puede sobrescribir la ruta donde se encuentran los archivos SSL mediante la opción `ssldir`.

También puedes ejecutar Puppet Master desde la línea de comandos para ayudar a probar y depurar errores *(debug).* Es muy recomendable que lo hagas si pruebas Puppet por primera vez. Para ello, inicia el *daemon* de Puppet Master:

```
# puppet master --verbose --no-daemonize
```

La opción `--verbose` muestra la salida de log detallada y la opción `--no-daemonize` mantiene el demonio en el primer plano y redirige el log a la salida estándar. También es posible utilizar el parámetro `--debug` para generar una salida más detallada de depuración del proceso *daemon.*

Un único binario

Toda la funcionalidad de Puppet está disponible a partir de un único fichero binario, como ocurre en otras herramientas como GIT, en lugar de los binarios individuales para cada función utilizados en las versiones iniciales de Puppet. Por ello, es posible arrancar el Puppet Master de esta forma:

```
# puppet master
```

Y para iniciar la funcionalidad del agente:

```
# puppet agent
```

Puedes ver la lista completa de las funciones disponibles en el comando de Puppet ejecutándolo con el parámetro de ayuda:

```
$ puppet --help
```

Conexión del primer agente

Ahora que tenemos al servidor Puppet Master configurado y funcionando, es el momento de conectar el primer agente. En la máquina donde vamos a instalar el agente, es necesario instalar los prerrequisitos y el paquete adecuado, como ya vimos previamente, mediante el gestor de paquetes correspondiente. A modo de ejemplo, en esta sección instalaremos un agente en la máquina `nodo1.ejemplo.edu` y la conectaremos con nuestro Master.

Para conectar el primer agente, vamos a ejecutarlo desde la línea de comandos, en vez de ejecutarlo como servicio, para ver más fácilmente lo que va pasando cuando conectamos a través de la salida estándar por consola. El daemon del agente de Puppet es ejecutado utilizando el comando puppet agent y se puede ver la conexión iniciada con el Master en el siguiente listado (también se puede ejecutar el cliente Puppet en el propio Puppet Master, pero vamos a comenzar con la forma más

tradicional cliente-servidor):

```
node1# puppet agent --server=puppet.ejemplo.edu --no-daemonize --
verbose
```

```
info: Creating a new certificate request for nodo1.ejemplo.edu
```

```
info: Creating a new SSL key at
/var/lib/puppet/ssl/private_keys/nodo1.ejemplo.edu .pem
```

```
warning: peer certificate won't be verified in this SSL session
notice: Did not receive certificate
```

Nota: Si no se especifica el servidor, Puppet buscará por defecto un *host* llamado puppet. Por lo general es recomendable definir un CNAME para el Puppet Master, tal como: `puppet.ejemplo.edu`

Podemos definir esto mismo en el fichero `/etc/puppet/puppet.conf` de configuración del agente, en la sección principal:

```
[main]
```

```
server=puppet.ejemplo.edu
```

El agente necesita poderse conectar con el Master, por lo que debe poder resolver su nombre de *host* y tener acceso al puerto de conexión, por eso es necesario, aparte de haber incluido la regla para permitir la conexión al puerto en el *firewall*, tener definido un CNAME para el Puppet Master y especificarlo en el fichero `/etc/hosts` del cliente.

El parámetro `--no-daemonize` nos permite ejecutar el agente Puppet en primer plano, sin crear el proceso *daemon*, y muestra la salida del comando por consola. Si no especificamos este parámetro, por defecto el agente Puppet se ejecutará en modo *daemon*.

Tal como ya hemos mencionado, para autenticar las conexiones entre el Master y los agentes, Puppet hace uso de certificados SSL. En las últimas versiones de Puppet se soporta tanto una arquitectura de CA (autoridad certificadora) simple, con un certificado raíz autofirmado que también se utiliza para firmar, como una arquitectura de CA intermedia, con un certificado raíz autofirmado que emite un certificado CA intermedio que se utiliza para firmar las peticiones entrantes de certificado. La arquitectura con CA intermedia es la recomendada, ya que es más segura y facilita la regeneración de certificados.

Para generar una CA intermedia para el servidor Puppet se debe ejecutar el siguiente comando antes de iniciar el servidor por primera vez:

```
puppetserver ca setup
```

Cuando un agente Puppet se conecta por primera vez al servidor, envía la solicitud de certificado al Master y espera a que el Master lo firme y devuelva el certificado. Cada dos horas comprobará si recibe el certificado firmado, hasta que se reciba o se cancele su ejecución (usando Ctrl-C, por ejemplo).

Nota: el tiempo que espera el agente para recibir el certificado se puede configurar mediante la opción `–waitforcert`, mediante un valor en segundos, o 0 para no esperar y parar de ejecutarse de inmediato.

Completando la conexión

El paso necesario que nos queda para completar la conexión y autenticar al nuevo agente es firmar el certificado que el agente envió al Master. Esto se hace mediante `puppet cert` en el Master:

```
puppet# puppetserver ca list
```

```
nodo1.ejemplo.edu
```

La acción `list` muestra todos los certificados que están a la espera de ser firmados, tras lo que procederemos a firmarlo mediante la acción `sign`.

```
puppet# puppetserver ca sign --certname nodo1.ejemplo.edu
```

```
Signed nodo1.ejemplo.edu
```

Puppet también permite activar el modo `autosign`, en el que, en lugar de firmar cada certificado individualmente, todos los certificados provenientes de direcciones IP o rangos de direcciones especificadas se firmarán automáticamente. Esto tiene, obviamente, algunas implicaciones y riesgos de seguridad, por lo que debes asegurarte de que lo utilizas adecuadamente.

En el agente debería mostrarse la siguiente salida un par de minutos después de haber firmado el certificado (o podríamos parar y arrancar de nuevo el agente de Puppet para no tener que esperar):

```
notice: Got signed certificate
```

```
notice: Starting Puppet client version 6.4
```

Ya tenemos a nuestro agente autenticado con el Master, aunque ahora recibimos otro mensaje de error:

```
err: Could not retrieve catalog: Could not find default node or by
name with 'nodo1.ejemplo.edu , nodo1' on node nodo1.ejemplo.edu
```

El agente está conectado y se ha autenticado correctamente con el Master, pero el mensaje de error nos está indicando que no existe todavía ninguna configuración disponible para la máquina `nodo1.ejemplo.edu`. El siguiente paso será definir en el Master la configuración que debe utilizar este agente.

Ideas clave

Nota: Las conexiones SSL confían en la exactitud de los relojes de los *hosts*, por lo que, si esta es incorrecta entre el Master y el agente, la conexión puede fallar con un error que indica que los certificados no son de confianza. Para asegurarnos de que la hora de los relojes de las máquinas está sincronizada podemos utilizar NTP (Network Time Protocol).

1.7. Referencias bibliográficas

Puppet. (2020). *Open-source Puppet documentation*. *Welcome to Puppet 7.8.0.* https://puppet.com/docs/puppet/latest/puppet_index.html

Rhett, J. (2015). *Learning Puppet 4*. O'Reilly.

Uphill, T. (2014). *Mastering Puppet*. Packt Publishing.

Puppet Overview and Setup

Dubey, A. (2018). *Puppet Overview and Setup.* Medium. https://medium.com/@abhishekbhardwaj510/puppet-overview-and-setup-1a6bad24d61c

Este tutorial nos guía en la instalación y configuración de Puppet en un escenario con un Puppet Master y dos agentes.

Documentación de referencia de Puppet

Puppet. (2020). https://puppet.com/docs/puppet/latest/puppet_index.html

En el sitio oficial de documentación de la herramienta Puppet encontrarás la documentación de referencia más completa y actualizada. También puedes acceder a la versión de la documentación correspondiente a la versión de la herramienta que estés utilizando. Es el primer recurso recomendado al que debes acceder en busca que cualquier información relacionada con Puppet, para asegurarte de su veracidad y actualidad.

Puppet – Overview and Architecture

TutorialsPoint. (2018). Puppet Overview. https://www.tutorialspoint.com/puppet/puppet_overview.htm
TutorialsPoint. (2018). Puppet Architecture. https://www.tutorialspoint.com/puppet/puppet_architecture.htm

Los dos enlaces de esta referencia son parte de un tutorial básico de Puppet más extenso. Estos enlaces corresponden a los dos primeros capítulos, donde se ofrece una visión general de la herramienta y de su arquitectura.

Test

1. ¿Cómo se denomina en Puppet al host principal que gestiona toda la configuración del entorno?

A. Puppet Server.

B. Puppet Master.

C. Puppet Agent.

D. Nodo.

2. ¿Cómo se denomina en Puppet a los hosts gestionados por un Puppet Master?

A. Puppet Server.

B. Puppet Master.

C. Puppet Agent.

D. Nodo.

3. ¿Cómo se conectan los agentes de Puppet con su servidor maestro?

A. A través de una conexión cifrada utilizando el protocolo SSH.

B. A través de una conexión cifrada y autentificada utilizando el protocolo SSH.

C. A través de una conexión cifrada y autentificada utilizando el estándar SSL.

D. A través de una conexión cifrada y autentificada por HTTP.

4. ¿Qué característica tiene el lenguaje que utiliza Puppet?

A. Es un lenguaje declarativo, que declara los pasos que hay que ejecutar de configuración.

B. Es un lenguaje declarativo, que declara el estado final de la configuración.

C. Es un lenguaje imperativo o procedimental, que describe los pasos que hay que ejecutar de configuración.

D. Es un lenguaje imperativo o procedimental, que define el estado final de la configuración.

5. ¿A qué se denomina proveedor en Puppet?

A. Es el que proporciona los recursos en la nube, como AWS o Azure.

B. Es el que proporciona el "cómo" de cada tipo, según la plataforma.

C. Es el que proporciona el "qué" de cada tipo, según la plataforma.

D. Es el que proporciona la conectividad entre el nodo y el maestro.

6. ¿Cómo se gestiona la idempotencia en Puppet?

A. Puppet no soporta idempotencia.

B. Mediante los proveedores, que implementan los tipos de recursos.

C. Mediante los tipos, que permiten gestionar los distintos elementos de configuración.

D. Mediante la capa transaccional, que permite crear configuraciones y aplicarlas repetidamente en el *h*ost.

7. ¿Qué herramienta nos proporciona los *facts* en Puppet?

A. Facter.

B. Puppetfacts.

C. Factia.

D. Puppetfacter.

8. ¿Qué prerrequisitos puede tener la instalación de Puppet?

A. Instalar Python y alguna de sus bibliotecas.

B. Instalar Git.

C. Instalar Ruby y alguna de sus bibliotecas.

D. Instalar Facter.

9. ¿Cómo le indicamos a Puppet la configuración que hay que cargar y dónde aplicarla?

A. A través de variables de entorno.

B. Mediante parámetros de configuración.

C. A través del archivo `site.pp`

D. A través del archivo `manifest.pp`

10. ¿Qué pasa cuando un agente se conecta por primera vez al Puppet Master?

A. Espera la respuesta del servidor con la configuración a aplicar.

B. Envía una solicitud de certificado. El Master deberá generarlo y devolverlo firmado.

C. El Master lee el archivo site.pp para ver si el agente está ahí referenciado.

D. El agente informa de su nombre para que el Master lo añada al archivo `site.pp`