>_ developerlover.com

CREAR UN ENTORNO DE DESARROLLO CON VAGRANT Y PUPPET

12 FEBRERO, 2015 / 0 COMENTARIOS

En este post vamos a explicar cómo **crear un entorno de desarrollo con Vagrant y Puppet** sobre una máquina virtual **Ubuntu** utilizando **VirtualBox** como software de virtualización.

Sólo como ejemplo para escribir esta entrada y para entender el funcionamiento de Vagrant y Puppet, vamos a crear un **entorno de desarrollo LAMP** (Linux, Apache, MySQL y PHP). Una vez comprendamos su funcionamiento podremos crear cualquier tipo de entorno sin problema.

Pero antes de nada, vamos a explicar qué es Vagrant, qué es Puppet y para qué los necesitamos.

¿Qué es Vagrant?

Como dicen en su página web, Vagrant es una herramienta de código abierto cuyo objetivo principal es la **creación y configuración de ambientes virtuales de desarrollo de manera muy ligera, reproducible y portátil**.

Dicho de una manera más sencilla, Vagrant se va a encargar, con un simple comando, de levantar una máquina virtual utilizando el software de virtualización que queramos (VirtualBox, VMware, KVM…), y configurarla automáticamente siguiendo las reglas que le hayamos indicado en el fichero de configuración, llamado Vagrantfile.

En el fichero Vagrantfile podremos indicarle, entre otras muchas cosas, qué número de procesadores usar, memoria, sistema operativo, IP privada, carpetas compartidas, aprovisionadores, etc… Prácticamente todo lo relacionado con el hardware y la configuración de la máquina virtual.

Como hemos dicho en el párrafo anterior, y como veremos en este post, podremos indicar a Vagrant como aprovisionar la máquina, es decir, realizar cambios.

Un **aprovisionador** es una herramienta que nos va a permitir, en la máquina virtual creada por Vagrant, instalar paquetes, editar ficheros, arrancar servicios y todo lo que se nos ocurra que podríamos hacer de forma manual pero automatizado, nosotros solo le tendremos que indicar qué y cómo instalar todo aquello que necesitemos.

Algunos de los aprovisionadores más conocidos son Puppet (el que veremos en este post), Chef y Ansible entre otros. También podríamos aprovisionar una máquina mediante **shell script**, pero no es la mejor práctica.

¿Qué es Puppet?

Puppet es una herramienta de gestión de la configuración de código abierto. Es lo que antes hemos llamado **aprovisionador**. Está escrito en **Ruby**, y se va a encargar de configurar el sistema de nuestra máquina virtual.

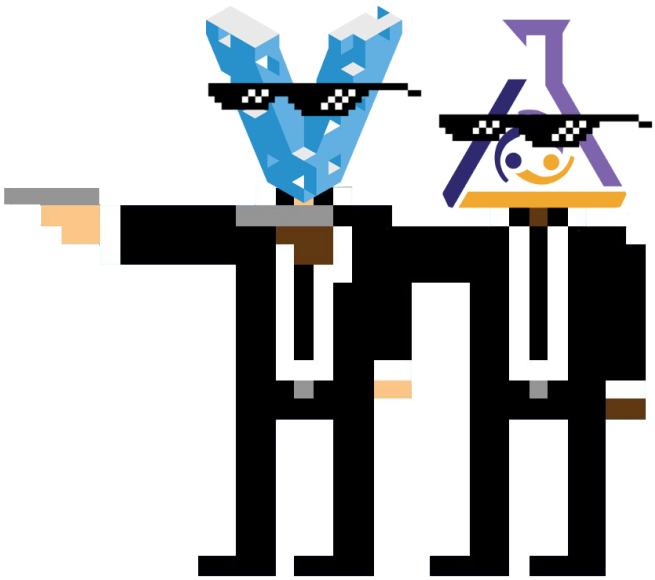
Si antes hemos dicho que la función de Vagrant es configurar el hardware de la máquina virtual, Puppet lo va a hacer del sistema operativo instalado en dicha máquina y de su software.

Para los que conozcan Composer, el concepto de Puppet podría entenderse como algo parecido, pero orientado al software y su configuración.

Al igual que con Vagrant y su Vagrantfile, aquí también tendremos ficheros de configuración llamados **manifiestos**. Digo ficheros porque, aunque podríamos escribir toda la configuración en un único fichero, va a ser mejor separarla y agruparla en **módulos**. Por ejemplo, el módulo de PHP, que se va a encargar de realizar todas las operaciones relacionadas con PHP, como instalar el paquete o editar el fichero php.ini.

Los manifiestos incluirán **recursos**, y con ellos podremos **indicar qué paquetes instalar en el sistema operativo** (PHP, Apache, Curl, htop...), **configurar los paquetes** (php.ini, apache.conf...), **clonar repositorios**, y todo lo que nos podamos imaginar. En resumen, con Puppet podremos hacer todo lo que podríamos hacer a mano, pero escrito en ficheros y ejecutado posteriormente de forma automática a través de Vagrant.

¿Para qué necesito Vagrant y Puppet?



Entre otras muchas cosas, para **crear o desechar entornos de desarrollo con diferentes configuraciones** o **configurarnos un entorno local con el que trabajar en nuestro proyecto**. Es prácticamente lo mismo que crearnos nuestra máquina virtual a mano pero no.

Con Vagrant y Puppet podremos tener todos los ficheros de configuración (**Vagrantfile** y **manifiestos**) sincronizados con nuestro sistema de control de versiones y disponibles para el resto de desarrolladores. De esta forma, todos podrán tener y trabajar con el mismo entorno con solo ejecutar un comando y en cuestión de minutos.

Imaginemos también que se incorpora un compañero nuevo al equipo. ¿Debería crearse a mano su entorno de trabajo para todos los proyectos con los que trabajemos? NO. Bastaría con que se instalase Vagrant y el software de virtualización elegido, clonarse el proyecto que contenga los ficheros de configuración de Vagrant y Puppet, y ejecutar el comando:

```
$ vagrant up
```

Y de esta forma tan sencilla tendría un entorno de trabajo funcionando, configurado e idéntico al del resto del equipo.

Descarga e instalación de Vagrant y Puppet

Para configurarnos nuestra máquina virtual vamos a necesitar instalar Vagrant y VirtualBox. Como he dicho antes, también es compatible con otros proveedores de virtualización como VMware o KVM.

Los enlaces a la página de descarga de Vagrant y VirtualBox son los siguientes:

- Descargar Vagrant
- Descargar VirtualBox

Configuración de Vagrant

Para este paso vamos a crear un directorio y dentro de él ejecutar el siguiente comando:

```
$ vagrant init
```

Este comando va a crear el fichero `Vagrantfile` en nuestro directorio. No es obligatorio usar el comando para crearlo, también podemos crear un fichero vacío, el único requisito es que su nombre sea "**Vagrantfile**".

Cuando implementemos Vagrant en nuestro proyecto, una buena localización del fichero `Vagrantfile` sería la raíz del proyecto.

Para este tutorial vamos a utilizar el siguiente `Vagrantfile` :

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure('2') do |config|

  # Sistema operativo
  config.vm.box = 'ubuntu/trusty64'

  # IP privada
  config.vm.network 'private_network', ip: '10.0.0.100'

  # Configuración de VirtualBox
  config.vm.provider :virtualbox do |vb|

    # Nombre de la máquina virtual
    vb.name = 'developerlover'

    # Memoria
    vb.memory = 2048

    # Número de procesadores
    vb.cpus = 2

  end

  # Habilitar aprovisionamiento por Puppet
  config.vm.provision :puppet do |puppet|

    # Localización de los ficheros de configuración manifiestos
    puppet.manifests_path = 'puppet/manifests'

    # Nombre del manifiesto que se va a ejecutar inicialmente
    puppet.manifest_file = 'default.pp'
```

```
34.
35.     # Opciones de Puppet. Se activa el modo debug y verbose
36.     puppet.options = [
37.       '--verbose',
38.       '--debug',
39.     ]
40.
41.   end
42.
43. end
```

Solo comentar que Vagrant utiliza boxes para instalar en la máquina virtual. Una box es un sistema operativo, pero puede traer más software instalado. Podéis encontrar más en vagrantcloud o vagrantbox.

Cuando ejecutamos `vagrant up`, Vagrant detecta si ya tenemos esa box descargada. Si no la tenemos, se la descargará automáticamente y la instalará en nuestra máquina virtual.

Podemos ver la lista de boxes descargas con el comando:

```
$ vagrant box list
```

Configuración de Puppet

Para la configuración de Puppet vamos a crear el siguiente árbol (que previamente hemos configurado en el archivo `Vagrantfile`):

```
.
| - - puppet
|     ` - - manifests
|           ` - - default.pp
` - - Vagrantfile
```

IMPORTANTE: Como hemos dicho al inicio, y sólo como ejemplo para este post, vamos a crear un **entorno de desarrollo LAMP** (Linux, Apache, MySQL y PHP). Pero lo importante es entender como funciona Puppet con Vagrant, qué son los manifiestos y los recursos y cómo utilizarlos. Una vez entendido esto podremos crear cualquier tipo de entorno según nuestras preferencias.

Y dicho esto, como contenido del fichero `default.pp` utilizaremos el siguiente:

```
1. # Actualizar los repositorios de paquetes
2. exec { "apt-get update":
```

```
    command => "/usr/bin/apt-get update"
}

# Instalación de Apache
package { "apache2":
    ensure => present,
    require => Exec["apt-get update"]
}

# Arrancar el servicio de Apache
service { "apache2":
    ensure => running,
    require => Package["apache2"]
}

# Lista de paquetes de PHP para instalar
$packages = [
    "php5",
    "php5-cli",
    "php5-mysql",
    "php5-dev",
    "php5-curl",
    "php-apc",
    "libapache2-mod-php5"
]

# Instalación de los paquetes de PHP
package { $packages:
    ensure => present,
    require => Exec["apt-get update"],
    notify => Service["apache2"]
}

# Instalación del servidor de MySQL
package { "mysql-server":
    ensure => present,
    require => Exec["apt-get update"]
}

# Arrancar el servicio de MySQL
service { "mysql":
    ensure => running,
    require => Package["mysql-server"]
}
```

IMPORTANTE: Aunque he dicho antes que es recomendable agrupar los recursos por módulos, en este tutorial vamos a utilizar solo un manifiesto ya que no son muchos los recursos que necesitamos. En el siguiente post explicaremos cómo crear y utilizar los módulos de Puppet con Vagrant y veremos como nos serán de utilidad a medida que vayamos creando mas recursos.

Cada bloque que vemos en este fichero es lo que antes hemos llamado recurso. De forma resumida, hay que entender un recurso como una acción. En este caso, la acción de actualizar los repositorios, la acción de instalar los paquetes de PHP, la acción de arrancar el servicio de Apache... Los recursos que utilizamos aquí, junto con una explicación muy resumida, son los siguientes:

- Exec: utilizado para la ejecución de comandos.
- Package: utilizado para la instalación de paquetes. Este comando internamente utilizará el gestor de paquetes del sistema operativo que tengamos instalado (apt-get, yum, pacman...).
- Service: el recurso utilizado con todo lo relacionado con los servicios.

Cada recurso va a tener un **identificador** obligatorio, que es el texto que hay anterior a los dos puntos. Este identificador debe ser único, y se utilizará entre otras cosas, para indicar una dependencia, notificar a un servicio que se reinicie, etc...

MUY IMPORTANTE: En Puppet las dependencias son indispensables, ya que **Puppet no hace una ejecución secuencial de los recursos**. Esto quiere decir que no se ejecuta de arriba a abajo. Le da igual el orden en el que escribamos los recursos en el fichero. Puppet leerá todos los recursos y creará su propia lista de ejecución basándose en las dependencias.

Una **dependencia** le indica a Puppet, mediante el atributo `require`, qué recurso o recursos son necesarios para ejecutar otro. Por ejemplo, en nuestro fichero estamos indicando a Puppet que para arrancar el servicio de Apache es necesario que ejecute el recurso de su instalación. Como es lógico, antes de arrancar el servicio de Apache lo tenemos que tener instalado.

Y para finalizar la (resumida) explicación de Puppet, indicar que cada recurso tiene **atributos**, que son los valores que se ven posterior a los dos puntos y que van a decir al recurso como ejecutarse. Como por ejemplo, en qué path crear un directorio o qué owner, grupo y permisos tiene que tener. Cada recurso tiene sus propios atributos, aunque hay algunos comunes, como el atributo `require`.

Crear/Encender la máquina virtual

Con todos los ficheros y directorios creados, solo nos quedaría levantar la máquina. Y como hemos dicho antes, esto lo vamos a hacer ejecutando el siguiente comando en el directorio donde se encuentra el fichero `Vagrantfile`:

```
$ vagrant up
```

Y si todo ha ido bien, al acceder a **http://10.0.0.100** deberíamos ver la página por defecto de Apache.

IMPORTANTE: Como dice el título, este comando crea o arranca la máquina virtual dependiendo si es la primera ejecución o no. Si Vagrant detecta que es la primera ejecución, también realizará todo el proceso de aprovisionamiento. En cambio, si no es la primera ejecución, simplemente encenderá la máquina virtual.

Acceder a la máquina virtual por SSH

Para acceder a la máquina por SSH basta con ejecutar, dentro del directorio donde tenemos el fichero `Vagrantfile`, el siguiente comando:

```
$ vagrant ssh
```

Añadir un cambio a Puppet

Imaginemos que queremos añadir un cambio a la configuración de Puppet. Por ejemplo, queremos **editar la contraseña del usuario root de MySQL**.

Solo tendríamos que añadir el siguiente recurso a nuestro manifiesto `default.pp`:

```
1. # Editar password del usuario root de mysql
2. exec { "set-mysql-root-password":
3.     unless => "mysqladmin -u root -p1234 status",
4.     command => "mysqladmin -u root password 1234",
5.     require => Service["mysql"],
6.     path => "/bin:/usr/bin",
7. }
```

IMPORTANTE: El atributo `unless` se utiliza como filtro antes de ejecutar un recurso. Significa que sólo va a ejecutar el comando indicado en el atributo `command` cuando el comando indicado en el atributo `unless` falle. Aunque lo explicaremos un poco más abajo, este tipo de atributos se utilizan para no repetir la ejecución de un recurso en cada aprovisionamiento.

Y una vez añadido el recurso, desde nuestro directorio donde se encuentra el fichero `Vagrantfile` y con la máquina encendida, ejecutamos en la terminal el siguiente comando:

```
$ vagrant provision
```

Este comando va a ejecutar, como su propio nombre indica, el aprovisionamiento de la máquina.

MUY IMPORTANTE: **Aprovisionar la máquina significa que Puppet va a ejecutar TODOS los recursos, incluyendo tanto los nuevos como los viejos**. Esto quiere decir que si tenemos un recurso que descarga un archivo, en cada aprovisionamiento que hagamos Puppet va a ejecutar ese recurso y volverá a descargar ese archivo. Esto lógicamente tiene solución.

Aquí entra en juego un principio fundamental en Puppet, el **principio de idempotencia**. ¡Que no asuste el nombre! Esto significa que no importa el número de veces que se ejecute un recurso, el efecto debe ser como si sólo se hubiera ejecutado una sola vez.

Dicho de otra manera, nosotros tenemos que conseguir que Puppet no ejecute un recurso si el objetivo final ya se ha conseguido. Y claro, para conseguir esto Puppet nos ofrece soluciones.

Volviendo al ejemplo de la descarga del archivo, supongamos que queremos descargarnos el adminer (para su posterior instalación) y como es lógico sólo queremos descargarlo una sola vez, no queremos hacerlo en todos los aprovisionamientos.

Un **recurso erróneo** sería el siguiente:

```
1. # Descargar adminer
2. exec { "adminer-download":
3.     command => "sudo wget 'http://www.adminer.org/latest.php' -O
   /home/vagrant/adminer.php",
4.     path => "/bin:/usr/bin"
5. }
```

Este recurso se ejecutaría y descargaría el adminer en todos los aprovisionamientos, ya que no hay ninguna regla que le indique cuando no descargarlo.

Esto no debería ser así, ya que aunque no dé error, sería tiempo extra que añadiríamos a cada aprovisionamiento. Imaginaros si clonásemos repositorios, descargásemos bases de datos, ejecutásemos composers... El tiempo de cada aprovisionamiento sería excesivo para por ejemplo, instalar un paquete nuevo de PHP. Como solución, Puppet nos ofrece algunos atributos en los recursos que nos ayudan en estos casos.

Para este recurso en concreto (que es el encargado de ejecutar comandos), existe un atributo llamado `creates`, que le va a indicar a Puppet que la ejecución de ese comando va a crear un archivo o directorio (en este caso el fichero `/home/vagrant/adminer.php`), y al siguiente aprovisionamiento, si detecta que ya existe ese directorio o archivo, no ejecutar el recurso.

Añadiendo el atributo `creates` el recurso debería quedar así:

```
1. # Descargar adminer
2. exec { "adminer-download":
```

```
3.      command => "sudo wget 'http://www.adminer.org/latest.php' -O
   /home/vagrant/adminer.php",
4.      creates => "/home/vagrant/adminer.php",
5.      path => "/bin:/usr/bin"
6.  }
```

De esta forma solo se ejecutará cuando Puppet detecte que ese fichero no existe.

Y esto ha sido todo. Por último decir, que en este post cuento de forma muy resumida qué es Vagrant y Puppet, y que de aquí en adelante intentaré escribir, de forma más detallada y específica, otras características y aplicaciones de estas herramientas.

Por cierto, os dejo unos links que a mi me sirven de bastante ayuda:

- Recursos de Puppet
- Puppet Cookbook
- Puppet Cheatsheet

Entradas Relacionadas:

1. **Cómo crear y utilizar los módulos de Puppet con Vagrant**
2. **Crear una máquina virtual CentOS con VirtualBox (1 de 2)**
3. **Crear una máquina virtual CentOS con VirtualBox (2 de 2)**

Categorías: Puppet, Vagrant, VirtualBox

Etiquetas: DevOps, Entorno de desarrollo, LAMP, Manifiestos Puppet, Máquina virtual, Recursos Puppet, Vagrantfile

Cómo crear y utilizar los módulos de Puppet con Vagrant »

« Habilitar enlaces simbólicos en las carpetas compartidas de VirtualBox

Deja un comentario

Your email address will not be published.

Name

Email

Website

Publicar comentario

¿Quieres recibir una notificación por correo electrónico cada vez que se añada un comentario en esta entrada?

Sólo si alguien responde a mi comentario(s) ▼

PÁGINAS

Entradas

VirtualBox

Vagrant

Puppet

Contacto

RSS

CATEGORÍAS

CentOS

ELK

MySQL

Node.js

OS X

Otros

Puppet

Sistemas operativos

Sphinx

Vagrant

VI

VirtualBox

ETIQUETAS

Archivo hosts Backup Base de datos Big data Buscador

Carpetas compartidas VirtualBox Cheatsheet DevOps Elasticsearch Enlaces simbólicos

Entorno de desarrollo Full text HTTP HTTPS IP Kibana LAMP Logs

Logstash Línea de comandos Manifiestos Puppet Monitorización mysqldump

Máquina virtual Módulos Puppet OS X Recursos Puppet SELinux SphinxQL SQL

SSH Vagrantfile VI VirtualBox Guest Additions VirtualHost XML xmlpipe2

Up ↑