Ingeniería de servidores - Creación de entornos virtuales con Vagrant

Doble Grado de Ingeniería Informática y Matemáticas
Universidad de Granada - UGR
18001 Granada, Spain

May 1, 2016

Resumen

Vagrant es un software que simplifica la creación de entornos virtuales, automatizando dicho proceso a unos pocos comandos usando la terminal. Para conseguir esto, se sirve de máquinas virtuales predeterminadas, descargables desde su página web. En este trabajo explicaremos cómo vagrant puede ayudarnos a crear entornos virtuales a partir de dichas máquinas, configurarlos a nuestro gusto, personalizarlos de forma automática con el software que necesitemos, y finalmente crear nuestras propias máquinas predeterminadas.

1 Introducción

Este trabajo va a versar sobre Vagrant, un software de virtualización que nos permite levantar, utilizar y compartir de forma sencilla entornos virtualizados, normalmente orientados al desarrollo y testeo de software. Este software está desarrollado por la empresa Hashicorp [4], la cual se dedica a proporcionar soluciones para la gestión de grandes infraestructuras y la coordinación de grupos de trabajo numerosos. Fue fundada en 2012 por dos ingenieros informáticos de la Universidad de Washington, Mitchell Hashimoto y Armon Dadgar. Dicha empresa trabaja dentro de lo que se conoce como la filosofía de trabajo "DevOps". Dicha filosofía enfatiza especialmente en la colaboración entre los distintos desarrolladores de software dentro de una misma empresa, así como la comunicación con otros profesionales de la tecnología que en ella trabajen.

En los últimos años, Hashicorp se ha convertido en un referente dentro de este sector, y su software es utilizado por grandes empresas, como la red social Pinterest, Cisco, o Digital Ocean, empresa proveedora de servidores virtuales.

Dentro de las soluciones que nos aporta la empresa, además de Vagrant, tenemos Packer, que nos permite automatizar la creación de máquinas virtuales con una configuración concreta, Serf, que nos permite gestionar clústers de equipos, Consul, que nos permite crear y manejar "datacenters", Terraform, que permite gestionar redes de equipos que están conectados a través de la nube, Otto, que es una herramienta para el desarrollo colaborativo de software en redes, o Vault, que es un gestor de seguridad para redes de equipos que permite el manejo de certificados, contraseñas...

La empresa tiene dos modalidades de trabajo bien diferenciadas. Por un lado, todas las herramientas mencionadas antes son software libre y de código abierto [10]. Por otro, tienen una suite de software orientada a empresas, conocida como Atlas, que engloba el software mencionado anteriormente, así como algunas funcionalidades extra, que dependen del plan de precios que se seleccione. Además, ofrecen soporte técnico tanto por correo electrónico como telefónico. No podemos citar precios del servicio dado que la empresa adapta el precio de la suite al tamaño de la empresa que la solicita. Dentro de las empresas usuarias de Atlas, podemos encontrar Mozilla, o OpenAI, que es una empresa fundada por Elon Musk en Diciembre de 2015, que se dedica a fomentar el desarrollo de inteligencia artificial con software libre.

2 Primeros pasos - Instalación de programas necesarios

Es intuitivo darse cuenta de la necesidad de instalar Vagrant para poder profundizar en el estudio del mismo. Vagrant puede instalarse tanto en Linux, como Mac OS X y Windows. Otro requisito para poder usar Vagrant es la instalación de software de virtualización como VirtualBox. Estas dos herramientas necesarias son gratuitas. Por lo tanto, partimos de los siguientes componentes fundamentales para construir el entorno:

1. Software Vagrant.
2. Máquina virtual (En nuestro caso VirtualBox).

Podemos encontrar el software de Vagrant en la página oficial (https://www.vagrantup.com/downloads.html). Allí elegimos la distribución necesaria para el sistema operativo desde el que se parta. En nuestro caso haremos una instalación para Ubuntu. Nos dirigimos a la página oficial y descargamos el paquete correspondiente (Debian 64 bits). Se nos descargará un archivo del tipo .deb. Con el comando `sudo dpkg -i <ruta_al_paquete>` queda Vagrant instalado en el sistema.

En cuanto al software de virtualización, utilizaremos VirtualBox, ya que es el primer software de virtualización con el que trabajaba la empresa, y la guía de inicio viene redactada con ejemplos usando dicho software [6]. No obstante, en sus últimas versiones ha ido incorporando nuevos programas que pueden realizar la virtualización del sistema, como VMware o AWS. En cuanto a la instalación de VirtualBox, podemos conseguir la distribución que más nos convenga desde su página oficinal [14].

3 Iniciando Vagrant - Manejo de máquinas

Una vez instalados Vagrant y VirtualBox en el equipo, vamos a ver cómo instalar y correr entornos virtuales. Para ello, se usan las "Boxes". Las Boxes son versiones de entornos virtuales ya configurados y con caraterísticas predeterminadas. Dichas boxes se pueden clonar y ser utilizadas en nuestros proyectos.

Es importante comprender lo que significa usar una imagen base para clonar. Esto es, la imagen es global y puede ser usada por varias máquinas sin que la imagen original sea modificada. Si dos proyectos utilizan la misma base box, dichas boxes son independientes e independientes al mismo tiempo de la box original. Lo que se modifique en uno u otro de los proyectos, o los archivos que se añadan no afectan en ningún caso a la otra máquina ni a la Box original.

Las Boxes pueden provenir del catalogo oficial de Vagrant [13] o pueden ser creadas (customizadas) por el propio usuario. Al final del trabajo veremos cómo puede crearse una Box propia. Esta posibilidad nos ofrece una gran flexibilidad a la hora de crear nuevos entonrnos virtuales, ya que podremos tener creada una Box con una configuración estándar que nos interese, y usar dicha Box siempre que nos sea necesario, para tantos proyectos como queramos.

Por lo tanto, cada máquina se inicia con una Box base. Se trata de un paquete que contiene una version del sistema operativo (el que nosotros deseemos para nuestro proyecto) preconfigurado. Por lo general, la configuración es mínima y su objetivo prncipal es contener los mecanismos necesarios para poder establecer la comunicación con Vagrant, dado que esta herramienta la que administrará las máquinas. El usuario es realmente el responsable de construir el entorno de trabajo añadiendo cualquier otro dispositivo (ej. un servidor de bases de datos).

3.1 Manejo de boxes

Veamos qué posibilidades nos ofrece Vagrant en cuanto al manejo de boxes en nuestro equipo. En particular comentaremos las órdenes que nos permiten añadir nuevas boxes (add), listar las boxes que tenemos instaladas en nuestro equipo (list), actualizar boxes que hayan sido modificadas (update) y eliminar boxes del equipo (remove).

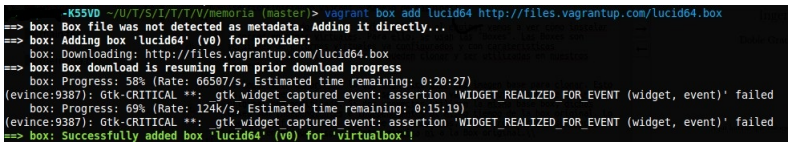

Figura 1: vagrant box add

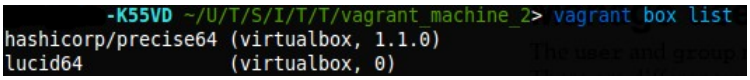

Figura 2: vagrant box list

3.1.1 Añadir una Box

Una box puede ser fácilmante añadida en el proyecto con el siguiente comando:

`vagrant box add <nombre-box>`

El nombre de las boxes tiene dos campos fundamentales. Aparecen tanto el nombre del usuario como el nombre de la box separados mediante una barra. Si se desconoce el nombre de la caja también es posible referirse a ella mediante una url o mediante una ruta local.

Como ya hemos dicho existen boxes oficiales y boxes ofrecidas por usuarios. Los nombres de las boxes no están sujetos a copyright. Es decir, el nombre "hashicorp" en una box no valida el producto como uno propio de la empresa, ya que cualquier usuario podría haberse dado de alta con ese nombre y utilizarlo dentro del sistema. Aunque no es habitual que una box presente algún problema, es responsabilidad del usuario confiar en la procedencia de las boxes, ya que la empresa no se hace responsable del software que las mismas puedan traer preinstalado.

Vamos a añadir una box de ejemplo en nuestro sistema, para poder crear un entorno virtual usando la misma más adelante. La que añadiremos en este caso será una imagen de Ubuntu 10.04 (Lucid Lynx), desarrollada directamente por VirtualBox. En la figura 1 se muestra el proceso de adición mencionado. Se recomienda utilizar siempre boxes de empresas confiables, para evitar problemas en nuestra propia máquina.

Como podemos observar en la imagen, lo que se hace es descargar en el equipo la base box y añadirla a la lista de boxs disponibles, usando el nombre que hemos especificado (lucid64). Las boxes que tenemos instaladas en nuestro equipo se guardan en el directorio `/.vagrant.d/boxes`. Una vez hecho esto, podremos utilizar dicha box para iniciar cualquiera de nuestros entornos virtuales, indicando simplemente el nombre de la box en nuestro equipo.

3.1.2 Listar las boxes disponibles

Como ya hemos dicho antes, las boxes que podemos utilizar en nuestros proyectos han de estar guardadas de forma local. Para consultar las boxes que tenemos instaladas, así como el nombre de las mismas, disponemos el comando `vagrant box list`, cuyo funcionamiento se muestra en la figura 2.

3.1.3 Actualizar una box

Puede ocurrir que se modifique la box original después de haber sido importada a nuestro equipo. Si queremos que dichas actualizaciones se vean reflejadas en nuestras boxes para posteriores usos, debemos actualizarlas. Para esto, tenemos el comando `vagrant box update <nombre_box>`. Dicho comando comprueba si la box que tenemos en nuestro equipo está anticuada respecto a la que hay en la red, y en tal caso, actualiza nuestra box.

Si sólo queremos comprobar si existen actualizaciones para una determinada box de las que tenemos instaladas, podemos utilizar el comando `vagrant box outdated <nombre_box>`

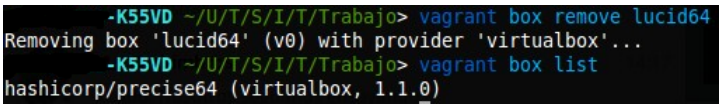

Figura 3: vagrant box remove

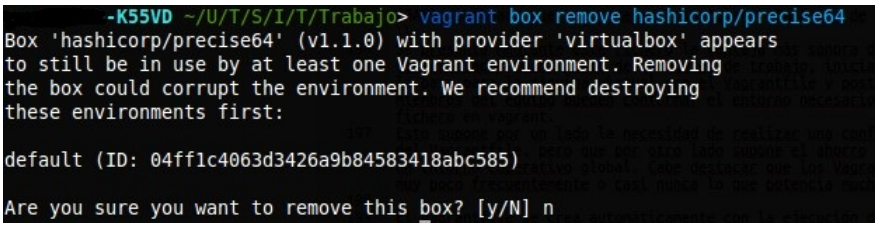

Figura 4: Advertencia en vagrant box remove

3.1.4 Eliminar una box del equipo

Si tenemos alguna box instalada que ya no es necesaria, podemos eliminarla de nuestro sistema. Para ello se utiliza el comando `vagrant box remove <nombre_box> <proveedor>`. En proveedor podemos especificar el proveedor de máquina virtual que estemos usando para levantar esa máquina. En nuestro caso, será virtualbox. El funcionamiento de este comando se muestra en la figura 3.

Como podemos observar en la imagen, hemos eliminado la box que habíamos importado anteriormente. Ahora, cuando listamos las boxes que nos quedan disponibles en el equipo, no nos aparece la antigua.

Por seguridad, Vagrant comprueba que la box que se está intentando eliminar no está siendo utilizada por ningún entorno virtual, ya que si estuviera siendo usada y se elimina, los entornos virtuales que la utilizaran quedarían inservibles. El error se muestra en la figura 4.

Esto se debe a que, aunque cada entorno es independiente de los demás con los que comparte box, y que las modificaciones hechas en cada entorno no modifican a la box original, el kernel que utilizan todos los entornos que usan la misma box se almacena en la propia box. De esta forma, la creación de un nuevo entorno en el sistema apenas necesita recursos, ya que la mayoría de la información queda almacenada en la box que dicho proyecto utiliza.

4 Proyectos

4.1 Crear un proyecto

Cada uno de los poryectos de vagrant se ejecutan en un entorno aislado. Un proyecto queda identificado por su Vagrantfile, el cuál es único en cada proyecto.

4.1.1 Vagrantfile

El Vagrantfile consiste en un fichero, escrito en el lenguaje de programación Ruby, que contiene la información necesaria para que Vagrant construya el entorno de trabajo [7]. Incluyen datos tan necesarios como el sistema operativo en el que se trabaja, características físicas deseadas para la máquina virtual (RAM), software que sea necesario instalar y la forma de acceder a la máquina. Se presenta mediante este fichero la ventaja más sonora de Vagrant. Cuando el desarrollador forma parte de un equipo de trabajo, inicialmente puede construir la base para la máquina virtual con el Vagrantfile y posteriormente, el resto de miembros del equipo pueden conformar el entorno necesario con sólo ejecutar el fichero en vagrant. Esto supone por un lado la necesidad de realizar una configuración a conciencia del Vagrantfile, pero que por otro lado supone el ahorro de tiempo y energía en un entorno coperativo global. Cabe destacar que los Vagranfile son modificados muy poco frecuentemente o casi nunca lo que potencia mucho más su utilidad.

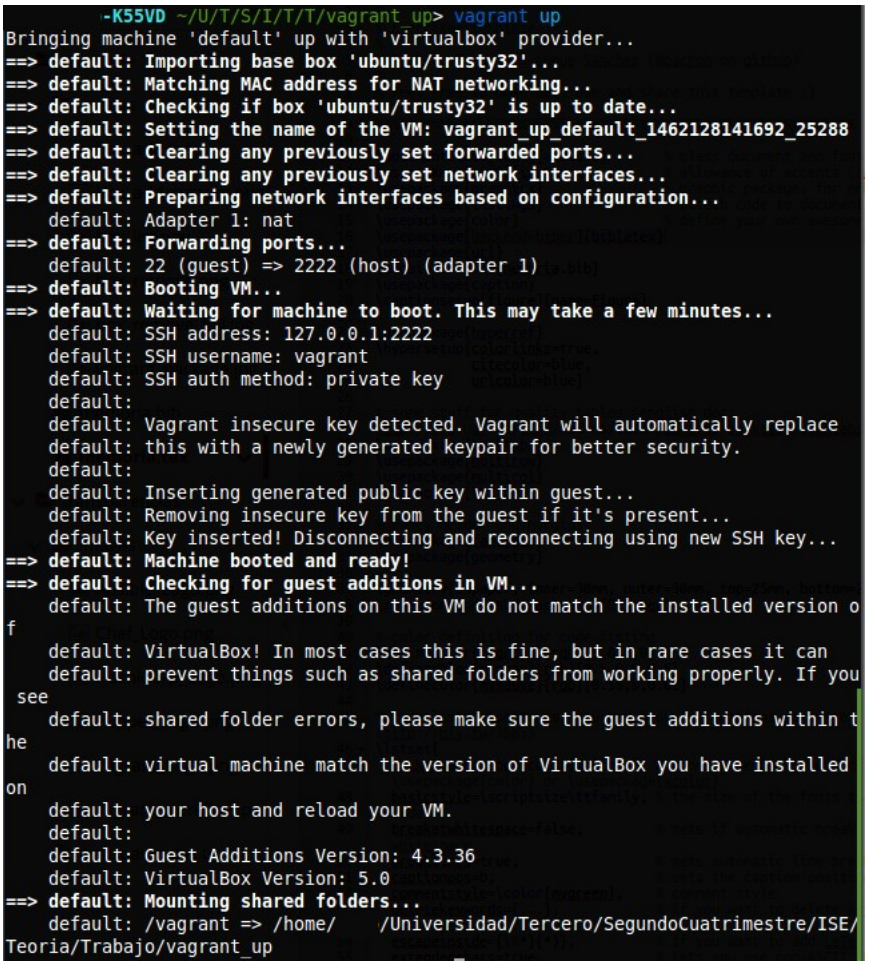

Figura 5: vagrant up

El Vagrantfile se crea automáticamente con la ejecución del comando `vagrant init <nombre-box> <url-de-box>`. Esta orden utiliza el directorio actual como entorno de Vagrant y crea el Vagrantfile si no ha sido creado anteriormente.

4.1.2 Up

El último paso en la creación de un proyecto consiste en la creación de la máquina Vagrant. Con la instrucción `vagrant up` se crea y configura la máquina virtual con la información que se presenta en el Vagrantfile. Podemos especificar el tipo de proveedor de virtualización que utilizamos en la creación de la máquina con la opción `--provider <nombre-proveedor>`. Por defecto utilizará VirtualBox. Tenemos un ejemplo del comando vagrant up en la figura 5

4.1.3 Sistema de archivos compartidos

Vagrant proporciona la configuración necesaria para compartir y sincronizar archivos desde el host a la máquina virtual y viceversa. Nos topamos así con la segunda gran ventaja de Vangrant. El desarrollador puede editar los archivos en el editor que más le convenga en la máquina host y Vagrant se encarga de la sincronización necesaria para que los cambios se lleven a cabo tambien en el guest. El resultado más tangible para esta ventaja radica en que los usuarios no deben aprender cómo manejar un nuevo editor, sino que el trabajo se facilita al estar familiarizados con su entorno de trabajo.

Además, el sistema de archivos compartidos proporciona un espacio donde almacenar los archivos para evitar que sean destruidos si se elimina una máquina virtual. Siguen estando disponibles para la siguiente máquina que se cree siempre que la configuración del Vagrantfile no provoque errores.

5 Provisioners

Ya tenemos levantado nuestro primer entorno virtual con Vagrant. El problema es que dicho entorno viene con un software muy básico, el mínimo para que la máquina funcione. En este punto es donde aparecen los proveedores (Provisioners en inglés). Estos programas nos permiten configurar nuestros entornos virtuales con el software que nos interese de forma rápida y simple. Simplemente, teniendo una serie de archivos de configuración creados previamente, podemos usar estos programas para que se encarguen de instalar todo el software que necesitemos, así como añadir y modificar los archivos de configuración pertinentes.

Durante el trabajo comentaremos los cuatro proveedores más importantes. Por un lado, tenemos Puppet [9] y Chef [2], que son los más utilizados, y son los más desarrollados en la documentación oficial de vagrant. Por otro, tenemos ssh, que nos permite conectarnos directamente al entorno virtual desde nuestro equipo y configurar los archivos directamente en el entorno virtual. Por último tenemos Ansible [8], que es una herramienta comercial de provisionamiento. Dicha herramienta, al estar enfocada a empresas, tiene un coste bastante alto, pero desde la propia página te permiten descargar una box de prueba para ver su funcionamiento.

Desde la documentación oficial de Vagrant recomiendan que se utilice ssh al principio, hasta familiarizarse con el "provisioning", y pasar posteriormente a utilizar Puppet o Chef. Pasamos entonces a estudiar todas estas herramientas:

5.1 SSH

Vagrant incorpora la herramienta necesaria para establecer una conexión ssh y poder conectarnos directamente con la máquina a través del comando `vagran ssh`.
Sin embargo, cabe destacar que la versión de Vagrant para Windows no trae incorporado el client SSH. El usuario debe hacerse cargo de instalar un cliente (Por ejemplo con OpenSSH).
Por lo tanto, podremos asegurar que SSH es una manera fundamental de comunicación entre el host y la máquina virtual y por ello es usado con Puppet o Chef.
Realmente existen dos formas de provisionamiento con SSH. La primera y más costosa consiste en hacer una conexión SSH con la máquina virtual y manualmente encargarse de realizar las instalaciones de software o añadir las características que queremos que nuestro entorno posea. Salta a la vista que este método es tedioso si el número de máquinas es elevado. La segunda forma y la más recomendable en este estilo de situaciones consiste en automatizar los procesos con scripts. El programador simplemente debe crearlos, transportarlos a su entorno Vagrant y ejecutarlos. Los scripts son una forma eficaz para agilizar el trabajo de cualquier programador, además de liberarlo de tareas repetitivas.

A la hora de configurar la conexión, podemos ajustar los parámetros necesarios en el archivo `config.ssh`. Por ejemplo, el programador podría querer cambiar el nombre del usuario que utiliza SSH ya que por defecto es "vagrant".

Anteriormente se ha mencionado que Vagrant permite compartir el entorno de trabajo de forma colaborativa. Este objetivo se consigue mediante `vagrant share`. Vagrant ofrece tres modos de colaboración no excluyentes.

1. **Compartir por HTTP**: proporciona una url que conecta directamente con el entorno Vagrant. Esta dirección puede ser compartida con cualquiera que que el usuario invitado no necesita tener instalado Vagrant.

2. **Compartir por SSH**: proporciona acceso SSH al entorno Vagrant. El usuario que intenta acceder si necesita Tener instalado Vagrant ya que la conexión se hace a través de `vagrant connect --ssh`.

3. **Compartir todo**: proporciona acceso a cualquier persona en cualquiera de los puertos disponibles. También se necesita disponer de Vagrant y la instrucción necesaria es `vagrant connect`.

5.1.1 SSH Sharing

Una vez habilitado, Vagrant genera dos claves nuevas. La clave pública se inserta en la máquina Vagrant, mientras que la privada se guarda en el servidor Vagrant que maneja los shares. La clave privada se encripta pidiendole una contraseña al usuario. Esta contraseña no se transmite y proporciona una capa extra de seguridad sobre el entorno compartido. Además, le otorga nombre al entorno para poder rederirse a él desde el exterior.

Por otro lado, el usuario invitado que desea entrar en la máquina necesita utilizar el comando `vagrant connect`, conocer el nombre de la máquina y la contraseña. Ejecutará el comando `vagrant connect -ssh <nombre-entorno>` y, puesto que se ha encriptado la clave privada, debe introducir la contraseña.

5.2 Puppet

Puppet es una herramienta de administración de configuración y provisionamiento de la empresa PuppetLabs[9]. Vagrant ofrece Puppet con filosofía Cliente-Servidor o en modo standalone. Este último es el más simple y en el que nos centraremos. Para poder usar Puppet de este modo necesitamos especificar en el Vagrantfile el tipo de provisionamiento que haremos. Esto se consigue introduciendo la siguiente línea `config.vm.provision :puppet`.

Los elementos básicos de Puppet son: el archivo manifiesto y los módulos. Toda la información sobre el servidor que administraremos o el fin con el qué usaremos Puppet (software, archivos, usuarios, etc) se concreta en su manifiesto. Por defecto, Puppet espera que el archivo manifiesto esté en la carpeta manifiesto. El segundo elemento de Puppet son los módulos. Los módulos utilizan herramientas internas de Puppet para configurar la máquina, customizarla, y pueden ser manejador para funcionar en etapas. De nuevo, necesitamos modificar el Vagrantfile para indicar la localización de los módulos en el sistema. Aunque se está indicando la ruta en el sistema host, Vagrant se encarga de copiar y trasladar los modulos a la máquina virtual para que Puppet pueda trabajar en esta máquina también. Existen muchos modulos ya creados y dispuestos a ser usados, que facilitan mucho el primer contacto con Puppet, como los que encontrados en Puppet Forge http://forge.puppetlabs.com/. Los módulos además contienen clases y es ahí donde el programador puede especificar lo que quiere que Puppet maneje. Por defecto Vagrant asume que tanto el manifiesto como los módulos se encuentran en la carpeta manifiesto y utiliza como manifiesto inicial el archivo `default.pp`.
Cabe destacar que Puppet no necesita que su estructura esté ordenada. Permite indicar dependecias, es decir, en un mismo bloque de instalación, el programador puede solicitar la existencia de una determinada funcionalidad o software como requisito a otra instalación. Puppet resuelve las dependencias comprobando que el sistema cumpla el requisito y en caso de no hacerlo, instala lo necesario. A continuación vamos a ver cómo se instala un servidor LAMP usando Puppet[1].

El Vagranfile tendría la siguiente apariencia.

```
Vagrant.configure(2) do |config|
  config.vm.box = "hashicorp/precise64"

  config.vm.provision :puppet do |puppet|
    puppet.manifests_path = "manifests"
    puppet.manifest_file = "default.pp"
    puppet.module_path = "modules"
  end

end
```

Lo siguiente constituye el cuerpo del manifiesto. En él se puede apreciar claramente estas dependencias de las que hablamos utilizando las instrucciones `require`, requisito previo, y `ensure`, que indica el estado del requisito previo.

```
# execute 'apt-get update'
exec { 'apt-update':
  command => '/usr/bin/apt-get update'
}

```

```
# install apache2 package
package { 'apache2':
  require => Exec['apt-update'],
  ensure => installed,
}

# ensure apache2 service is running
service { 'apache2':
  ensure => running,
}

# install mysql-server package
package { 'mysql-server':
  require => Exec['apt-update'],
  ensure => installed,
}

# ensure mysql service is running
service { 'mysql':
  ensure => running,
}

# install php5 package
package { 'php5':
  require => Exec['apt-update'],
  ensure => installed,
}

# ensure info.php file exists
file { '/var/www/html/info.php':
  ensure => file,
  content => '<?php  phpinfo(); ?>',
  require => Package['apache2'],
}
```

Entrará en funcionamiento ejecutanto `puppet apply <ruta-manifiesto>`[11], cuando levantemos la máquina con `vagrant up` ó ejecutando el comando de provisionamiento `vagrant provision`, el cuál ejecuta cualquier provisionamiento configurado en una máquina[5]

5.3 Chef

Chef es una herramienta de provisionamiento pura, enfocada especialmente a esta tarea, así que como el resto de funcionalidades son secundarias, nos centraremos sólo en el aprovisionamiento desde el punto de vista de Vagrant. Cuando se monta un sistema que va a ser provisto usando Chef, normalmente se instala un servidor (chef-server) en una máquina anfitriona, y un cliente (chef-client) en las máquinas virtuales. La instalación de dichos clientes está automatizada, pueden ser instalados haciendo correr un script que se crea en la máquina virtual tras la instalación, en el directorio /home de la máquina (llamado `postinstall.sh`). Una vez hecha la instalación cliente-servidor, podremos gestionar a todos los clientes desde el servidor.

Dado que la configuración del servidor puede ser engorrosa y no merece la pena cuando se trata de configurar una sola máquina, existe otra herramienta que permite el provisionamiento de la máquina virtual sin necesidad de la estructura cliente-servidor. Dicha herramienta se llama `chef-solo`, y es con la que haremos la prueba en nuestra máquina. Esta herramienta se instala también con el script de postinstalación.

Comentemos un poco el funcionamiento de Chef antes de mostrar la prueba. Los archivos que especifican las distintas instrucciones que se ejecutan en Chef son conocidos como recetas (recipes). Dichas recetas se almacenan en directorios, llamados libros de cocina (cookbooks). Estos cookbooks son los que se usan para provisionar a la máquina virtual. Dichos archivos se configuran en la máquina anfitriona y `chef-solo` se encarga de instalar y mantener el software de la máquina virtual, así como de realizar las tareas especificadas de forma periódica. En nuestra prueba lo que haremos será crear un cookbook para que se instale y actualice un servidor LAMP en nuestra máquina virtual.

Para manejar los cookbooks, vamos a utilizar otra herramienta, que nos permitirá descargarlos de lo que se conoce como "Chef supermarket" [3]. En dicho dominio se pueden encontrar cookbooks ya preparados para los servicios más comunes. La herramienta con la que gestionaremos la búsqueda

de cookbooks en este dominio se llama Berkshelf. Se puede instalar un plugin para vagrant que utiliza esta herramienta con el comando `vagrant plugin install vagrant-berkshelf`. Una vez instalado dicho plugin, vamos a ver cómo provisionar la máquina virtual con chef. Lo primero que haremos será configurar la carpeta de cookbooks local, así como el árbol de directorios que necesita berks para funcionar. Creamos la carpeta cookbooks en el directorio en el que vamos a crear la máquina virtual. Dentro de dicho directorio, ejecutaremos el comando `berks cookbook dev`, que nos configurará Berkshelf en el directorio. después, en el archivo dev/Berksfile, introduciremos los cookbooks que nos interese instalar (en nuestro caso, apache2, mysql y php). Una vez añadidos los cookbooks, modificaremos el Vagrantfile para que trabaje con chef y permita el uso de Berkshelf:

```
Vagrant.configure(2) do |config|
  config.vm.box = "hashicorp/precise64"

  config.berkshelf.enabled = true
  config.berkshelf.berksfile_path = "./cookbooks/dev/Berksfile"

  config.vm.provision :chef_solo do |chef|
  chef.run_list = [
    'recipe[apache2]',
    'recipe[mysql]',
    'recipe[php]',
  ]

  end
end
```

Ya sólo queda levantar la máquina virtual con `vagrant up`, y chef se encargará de instalar los paquetes especificados.

5.4 Ansible

Vamos a repetir el proceso de la sección anterior, instalar un servidor LAMP, pero ahora utilizando Ansible. La principal diferencia que tiene esta herramienta respecto de las anteriores es que trabaja usando SSH, sin necesidad de dependencias instaladas en la máquina virtual. Simplemente, teniendo la herramienta instalada en la máquina anfitriona, podemos configurar las máquinas virtuales de forma sencilla. Además, otra gran ventaja es la simplicidad de los archivos de configuración de Ansible. Son listas de tareas escritas en YAML, conocidos como "playbooks". Veamos, por ejemplo, el código que nos instalaría un servidor LAMP en la máquina virtual:

```
---
- hosts: all
  sudo: true
  tasks:
    - name: update apt cache
      apt: update_cache=yes
    - name: install apache
      apt: name=apache2 state=present
    - name: install mysql
      apt: name=mysql-server state=present
    - name: install php
      apt: name=php5-cli state=present
```

Como podemos observar, el archivo tiene una estructura muy simple. Especificamos que el trabajo ha de ser realizado por un superusuario, y en las tareas introducimos la actualización de la caché de apt, así como la instalación de los paquetes de apache, mysql y php. Una vez comentado el funcionamiento de Ansible, vamos a instalar una nueva máquina virtual con este software como provisioner.

Lo primero que hacemos es, en el directorio en el que vamos a montar la máquina, hacer `vagrant init hashicorp/precise64`, para que se cree el Vagrantfile con la box que nos interesa. Una vez hecho esto, vamos a modificar dicho archivo para especificar que nuestro software de provisionamiento va a ser Ansible:

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

  config.vm.box = "hashicorp/precise64"

  config.vm.network "forwarded_port", guest: 80, host: 8080

  config.vm.provision :ansible do |ansible|
    ansible.playbook = "playbook.yml"
  end

end
```

De paso, hemos cambiado también los puertos, para poder hacer una conexión desde la máquina local a la remota por http (hemos mapeado el puerto 80 de la máquina virtual al 8080 de la anfitriona). Una vez cambiado el Vagrantfile, vamos a escribir en el archivo `playbook.yml` (el que hemos especificado en el Vagrantfile como archivo playbook para Ansible) el código que listamos arriba. Una vez hecho esto, con el comando `vagrant up` iniciamos la máquina, y el sistema se encarga de dejarlo todo configurado.

5.5 Comparativa

Los tres provisioners presentados realizan en esencia la misma tarea: especificar la información necesaria para configurar nuestras máquinas. Otra similitud es que tanto Puppet como Chef ofrecen una versión simple y una con estructura Cliente-Servidor y mantienen vigilancia y status sobre las máquinas.Sin embargo existe una serie de sutiles diferencias.
La mayor diferencia entre Puppet y Chef se fundamenta en cómo se estructura el código. Puppet es declarativo en cuanto a que describe el estado deseado utilizando su manifiesto e indicando los pasos a seguir. Podríamos decir que interpreta su manifiesto con dependencias. Esto es, el programador puede especificar instalar un software antes que otro. Ahorra problemas como la instalación del software necesario antes de continuar con el resto de código. Chef es imperativo. El programador debe estructurar el código del cookbook de forma que se indique qué se debe hacer y en qué orden[15]. Por otro lado podemos separar Puppet y Chef de Ansible. Mientras que los dos primeros hacen uso de agentes, Ansible trabaja directamente con conexión SSH lo que convierte a esta herramienta en una más cercana a la forma de trabajar de un administrador de sistema. Las dos primeras utilizan Ruby en su código lo que ata al usuario a aprenderlo. También es cierto que ofrecen soporte en la mayoría de sistemas operativos debido a llevar más años en el mercado. Ansible en este aspecto queda por debajo dado que es relativamente nuevo y todavía necesita ser plenamente testeado.

6 Creación de una box propia

La última parte de nuestro trabajo la vamos a dedicar a la creación de una box propia. En muchos casos puede ser interesante crear una box personalizada que sea fácilmente clonable usando vagrant, en vez de recurrir a las predeterminadas que ya existen. Esto nos permite tener una box previamente configurada a nuestro gusto, y no tener que recurrir al provisionamiento cada vez que se crea una máquina nueva. Daremos una descripción general del proceso, dado que hacerla bien detallada podría ocupar mucho espacio. Se puede consultar una guía más detallada en [12]

Lo primero que tendremos que hacer es crear una máquina virtual usando el software de máquina virtual que nos interese (en nuestro caso VirtualBox). No nos extenderemos en este proceso porque no tiene complejidad ninguna. Lo único que varía con la creación de una máquina virtual estándar es al crear el disco duro, que lo seleccionaremos con extensión .vmdk en lugar de .vdi, para permitir la compatibilidad con otros softwares.

Una vez creada, tendremos que hacer algunos cambios en la configuración. Nos vamos a la ventana de configuración de nuestra máquina virtual, y una vez dentro deshabilitamos el sonido, ya que no es necesario, creamos un adaptador de red y lo configuramos como una red NAT, y deshabilitamos el control por USB, ya que no va a ser necesario tampoco.

Figura 6: vagrant package

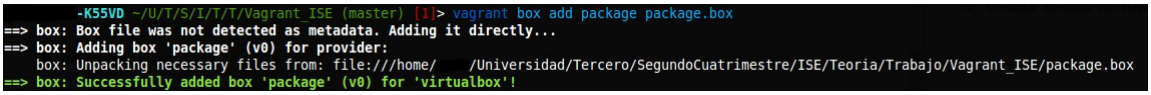

Figura 7: vagrant box add

Llegados a este punto, vamos a arrancar la máquina y a instalarle un SO. En nuestro caso, instalaremos Ubuntu 14.04. Durante el proceso de instalación hay que introducir una serie de campos de texto, para los que Vagrant espera una serie de valores predeterminados. Dichos campos son los siguientes:

- Nombre del sistema: vagrant-nombre-del-sistema (usaremos vagrant-ubuntu-trusty)
- Dominio: vagrantup.com
- Contraseña del root: vagrant
- Nombre de usuario principal: vagrant
- Contraseña de usuario principal: vagrant

Una vez instalado el sistema, vamos a instalar el software necesario para comunicar las máquinas. Esto lo podemos hacer instalando el software "VirtualBox Guest Additions" que nos proporciona Virtualbox, así como el paquete openssh-server.

Nuestro siguiente paso será crear un grupo de administradores en el que incluiremos el usuario "vagrant" que creamos durante la instalación. Esto se hace para que al conectarse, se tengan los permisos necesarios para instalar software, de cara al buen funcionamiento de los provisioners.

Ahora pasamos a permitir la conexión por ssh directa, usando clave pública. Vagrant nos aporta un par de claves pública y privada que podemos descargar desde https://github.com/mitchellh/vagrant/tree/master/keys/. Instalando dichas claves en el fichero de la máquina virtual `/ssh/authorized_hosts` tendremos solucionado el acceso por ssh usando pares de claves.

Finalmente, queda instalar el software de los provisioners que vayamos a usar (Chef, Puppet y Ansible, antes comentados).

Una vez terminado el proceso, podemos apagar la máquina virtual, y con el comando `vagrant package --base vagrant-ubuntu-raring`, del que se muestra el funcionamiento en la figura 6.

Una vez tenemos la box creada, podemos añadirla a las boxes del sistema con `vagrant box add` como se muestra en la figura 7

7 Bibliografía

References

[1] Mitchell Anicas. *Getting Started With Puppet Code: Manifests and Modules*. URL: https://www.digitalocean.com/community/tutorials/getting-started-with-puppet-code-manifests-and-modules.

[2] Chef Co. *Chef*. URL: https://www.chef.io/.

[3] Chef Co. *Chef Supermarket*. URL: https://supermarket.chef.io/.

[4] Hashicorp Co. *Hashicorp website*. URL: https://www.hashicorp.com/.

[5] Hashicorp Co. *Provision*. URL: https://www.vagrantup.com/docs/cli/provision.html.

[6] Hashicorp Co. *Vagrant - Getting Started*. URL: https://www.vagrantup.com/docs/getting-started/.

[7] Hashicorp Co. *Vagrantfile*. URL: https://www.vagrantup.com/docs/vagrantfile/.

[8] Red Hat Co. *Ansible*. URL: https://www.ansible.com/.

[9] The Puppet Company. *Puppet*. URL: https://puppet.com/.

[10] Mitchellh's Github. *Vagrant*. URL: http://github.com/mitchellh/vagrant.

[11] Luke Kanies. *Man Page: puppet apply*. URL: https://docs.puppet.com/puppet/4.4/reference/man/apply.html.

[12] Michael Peacock. *Creating Development Environments with Vagrant*. URL: https://books.google.es/books?id=v_MwBwAAQBAJ&lpg=PP2&dq=creating%20development%20environments%20with%20vagrant&hl=es&pg=PP2#v=onepage&q&f=false.

[13] Gareth Rushgrove. *Vagrant Box Catalog*. URL: http://www.vagrantbox.es/.

[14] VirtualBox. *Downloads*. URL: https://www.virtualbox.org/wiki/Downloads.

[15] Peter Wayner. *Puppet or Chef: The configuration management dilemma*. URL: http://www.infoworld.com/article/2614204/data-center/puppet-or-chef--the-configuration-management-dilemma.html.