Vagrant Documentation

Versión 0.1.0

Rafael Rodríguez Gayoso

09 de noviembre de 2017

Índice general

CAPÍTULO 1

Presentación

Vagrant es una herramienta que simplifica el trabajo para ejecutar y gestionar máquinas virtuales (MVs). Ofrece las siguientes características:

- Dispone de una interfaz de línea de comandos muy simple para gestionar las MVs.
- Soporta las principales soluciones de virtualización: VirtualBox, VMWare y Hyper-V.
- Tiene soporte para las principales herramientas de configuración, incluyendo a Ansible, Chef, Puppet y Salt.
- Facilita la distribución y migración de los entornos virtuales.

Aunque destaca en su uso bajo entornos de desarrollo de aplicaciones web, puede utilizarse para tareas diferentes. Podemos considerarla como una herramienta de propósito general. Si trabajamos de un modo manual en nuestro entorno de virtualización, tendremos que ejecutar las siguientes operaciones:

- Descargar la imagen de la MV.
- Arrancar la MV.
- Configurar los directorios compartidos y las interfaces de red.
- Posiblemente, instalar software adicional dentro de la MV.

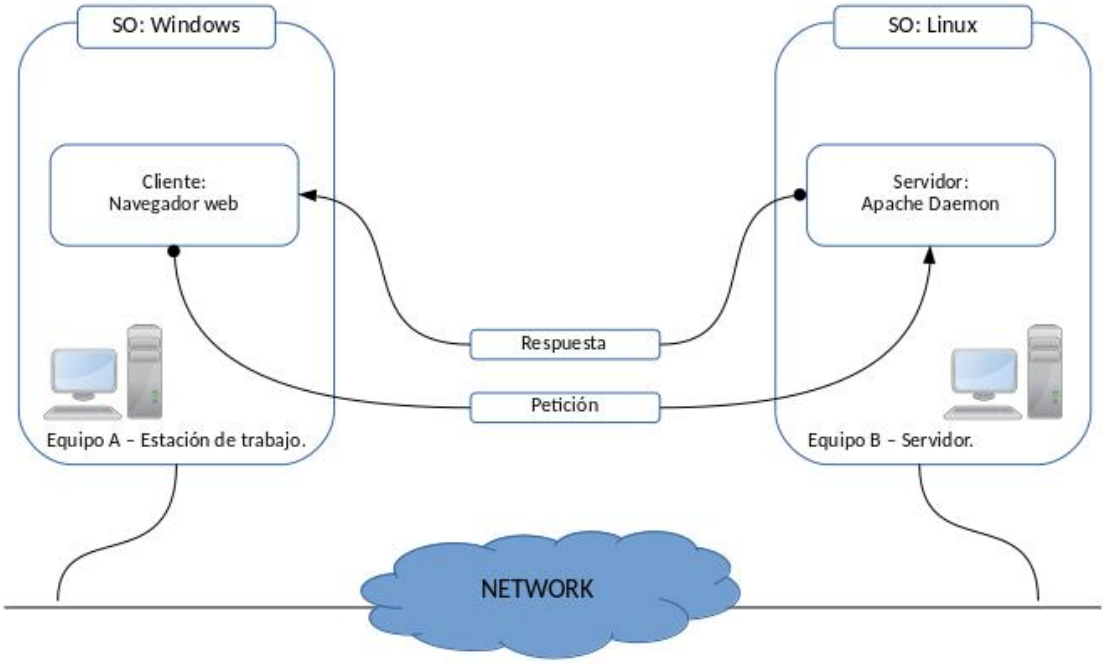
Con Vagrant, todas estas tareas (y muchas otras) se pueden automatizar. El comando `$ vagrant up` puede ejecutar las siguientes acciones (dependiendo del fichero de configuración):

- Descargar e instalar una MV, si es necesario.
- Arrancar la MV.
- Configurar varios recursos: RAM, CPU's, conexiones de red y carpetas compartidas.
- Instalar software adicional en la MV a través de herramientas como Puppet, Chef, Ansible o Salt.

CAPÍTULO 2

Paradigma cliente-servidor en aplicaciones web.

Las aplicaciones web utilizan el esquema cliente/servidor, haciendo uso del protocolo HTTP para la gestión de la comunicación. Cuando ejecutamos una aplicación web, existen dos partes bien diferenciadas: el cliente, que es el proceso que envía las peticiones; y el servidor, que el proceso responsable de responder. Dichas aplicaciones suelen estar en diferentes máquinas, equipadas con diferente hardware y sistema operativo. El cliente se puede estar ejecutando en un equipo portátil, y el servidor es el proceso que se ejecuta en una máquina remota.

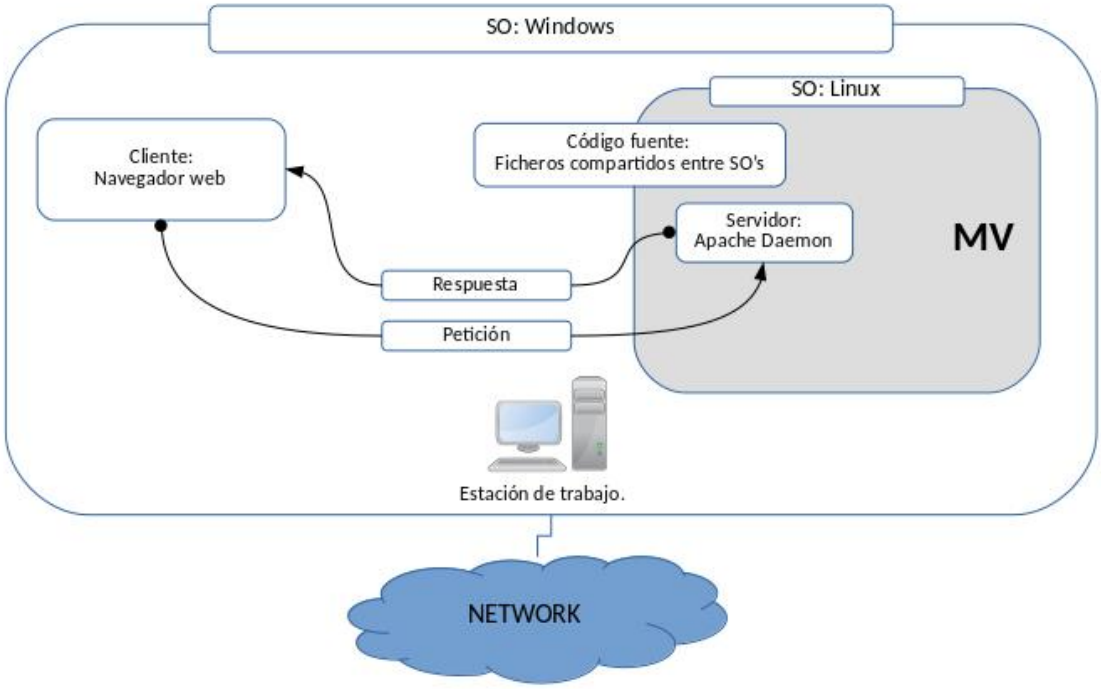


Los dos componentes de una aplicación web suelen denominarse como **front end** y **back end**. El **front end** es ejecutado en un navegador web en el lado del cliente; y el **back end** se ejecuta en el lado del servidor.

CAPÍTULO 3

Virtualizando la solución

Cuando trabajamos con MVs, podemos disponer del «front end» y del «back end» en la misma máquina; el equipo del desarrollador. El proceso cliente se ejecuta de igual forma que en el esquema anterior, como un proceso normal en el sistema operativo del equipo. El lado del servidor, es el que virtualizamos en una MV. Aunque ambos entornos se están ejecutando en la misma máquina, podemos reproducir el comportamiento real gracias a la virtualización.



Cuando trabajamos de este modo, tendremos que gestionar dos sistemas operativos. El original, instalado en la máquina del desarrollador, al que llamaremos «SO anfitrión» (host), y el que están instalado en la MV, que serán el «SO invitado» (guest).

CAPÍTULO 4

Vagrant

Vagrant es un software disponible bajo una licencia MIT, que fue desarrollado originalmente por Mitchell Hashimoto. El código fuente está disponible en Github.

Siguiendo la terminología de Vagrant, la solución de virtualización que está en la capa inferior se denomina «proveedor» (provider). Para poder usar Vagrant debemos instalar por lo menos un «proveedor». La lista oficial de proveedores es la siguiente:

- VirtualBox
- VMWare
- Docker
- Hyper-V

Aunque es posible trabajar con los entornos de virtualización anteriores sin usar Vagrant, se requiere de un gran esfuerzo para realizar las tareas de forma manual. Vagrant automatiza muchas de ellas a través del comando $ vagrant up. En la mayoría de casos, los desarrolladores sólo deben usar este comando para arrancar la aplicación como si se tratara de una caja negra, de la que desconocen su interior.

4.1 Instalación de software

Para poder realizar las siguientes prácticas, es necesario disponer de los siguiente paquetes:

- Git
- VirtualBox
- Vagrant

4.2 Consultar versión

Podemos comprobar la disponibilidad de Vagrant ejecutando el siguiente comando:

```
$ vagrant --version
Vagrant 1.9.1
```

4.3 Configuración básica

Por defecto, Vagrant almacena las VMs 'boxed' en la ruta *~/.vagrant.d*. Estas 'boxes' suelen ser ficheros de gran tamaño, y podemos considerarlas como una 'imagen modelo' de las MVs que tendremos en ejecución. Se pueden compartir entre diferentes usuarios, con el objetivo de optimizar el espacio utilizado en disco.

La ubicación del directorio principal de Vagrant puede modificarse a través de la variable de entorno **VAGRANT_HOME**. Por ejemplo, a través del siguiente comando:

```
$ export VAGRANT_HOME=/ruta/al/directorio
```

4.4 Ayuda y documentación

Tarde o temprano, tendremos que hacer uso de las fuentes de información para buscar ayuda en el uso de Vagrant. La primera opción puede ser la ayuda interna del propio comando:

```
$ vagrant
$ vagrant -h
```

El uso de Vagrant se realizar a través de 'subcomandos', como por ejemplo:

```
$ vagrant box
$ vagrant package
```

Durante la ejecución de las prácticas, haremos uso de comandos similares a estos:

```
$ vagrant box add
$ vagrant box list
```

Los subcomandos también poseen su propia ayuda si los ejecutamos del siguiente modo:

```
$ vagrant box -h
$ vagrant box add -h
```

Cuando esta primera ayuda no es suficiente, debemos utilizar la ayuda en línea de Vagrant.

CAPÍTULO 5

Framework Web

En este apartado veremos a Vagrant en acción, a través de la configuración de una aplicación web a través de un framework, en este caso, con AngularJS (Javascript).

Nuestro ejemplo hará uso del puerto TCP 8800, por lo que debe estar libre en nuestro equipo anfitrión.

5.1 Songs for kids

Este proyecto, que está escrito en AngularJS, está formado por tres páginas, cada una de ellas muestra el texto de una canción infantil en Inglés. Para poder ejecutar el proyecto debemos ejecutar los siguientes comandos:

```
$ cd directorio/para/ejercicios
$ git clone https://github.com/gaioso/songs-app-angularjs.git
$ cd songs-app-angularjs
$ vagrant up
...
# Run webbrowser and visit http://localhost:8800/
```

Dependiendo de la conexión de nuestro equipo, el proceso completo puede tardar varios minutos. Una vez finalizado, podremos acceder a los contenidos del proyecto a través de la URL que aparece en la última línea [http://localhost:8800/].

En este instante, hay dos sistemas operativos ejecutandose en nuestro equipo:

- Anfitrión (host): El sistema operativo principal.
- Invitado (guest): El sistema operativo iniciado mediante $ vagrant up.

Podemos comprobarlo a través de la interfaz de administración de VirtualBox, o bien con el siguiente comando:

```
$ vagrant global-status
```

La aplicación web está siendo servida por un servidor HTTP que se ejecuta dentro del sistema operativo invitado: Ubuntu 14.04. El código fuente de la aplicación está disponible tanto para el SO anfitrión como para el invitado a

través de un directorio compartido songs-app-angularjs/web/api/song/. Usando un editor de texto podemos modificar alguno de los ficheros json del directorio y comprobar los cambios a través del navegador del sistema anfitrión.

5.1.1 Ejercicio

Realizar las mismas operaciones que en el ejemplo anterior, pero con el código disponible en el siguiente repositorio:

```
https://github.com/gaioso/songs-app-rails
```

5.2 ¿Qué hemos conseguido?

Para poder ejecutar el primer ejemplo en nuestro sistema anfitrión sin hacer uso de la virtualización, tendríamos que instalar y configurar un servidor web (Apache, por ejemplo). Aunque no es una tarea complicada, requiere de un proceso manual y dependiente del sistema operativo que estemos usando. Del mismo modo que se mencionaba más arriba, esta tarea es responsabilidad de un perfil técnico, y un desarrollador no tiene porque saber instalar y/o configurar un servidor web para usar correctamente su aplicación.

CAPÍTULO 6

Destruír las máquinas virtuales

Cuando hayamos acabado de realizar el trabajo con entorno servidor, podemos destruír las máquinas, o bien, detenerlas:

```
$ vagrant halt [ID]
$ vagrant destroy [ID]
```

Los [ID] podemos consultarlos en la salida de `$ vagrant global-status`, o bien, situarnos dentro del directorio donde está ubicado en fichero Vagrantfile, que es donde se ejecutó el comando de creación de la MV `$ vagrant up`.

CAPÍTULO 7

Los estados de una Máquina Virtual

Comenzaremos por analizar las acciones involucradas en la ejecución de $ vagrant up, y las agruparemos en tres etapas. Prestaremos atención a varios ficheros y directorios que están involucrados en el arranque así como el momento en el que se inicia cada una de las etapas. Luego, analizaremos los cinco estados en los que se puede encontrar un sistema operativo invitado.

Haremos uso de un ejemplo similar al anterior, pero en esta ocasión, realizado con el framework Sinatra.

Antes de empezar

Comprobamos que no se están ejecutando otras MVs en nuestro equipo:

```
$ vagrant global-status
```

Descarga del código fuente de la aplicación

Para poder realizar el ejercicio debemos descargar el código fuente, que se encuentra disponible en https://github.com/gaioso/songs-app-sinatra/:

```
$ git clone https://github.com/gaioso/songs-app-sinatra.git
```

7.1 Vagrantfile

La configuración de las MVs usadas en nuestros ejercicios está guardada en el fichero Vagrantfile. Cada proyecto tiene su propio fichero, donde se guardan las propiedades de la MV. Este fichero está escrito en Ruby, y contiene, entre otras cosas, el nombre de la 'box' inicial desde la que construír la MV.

Como el fichero *Vagrantfile* almacena los detalles de la configuración de la MV, cualquier desarrollador que tenga acceso a dicho fichero, podrá acceder al mismo entorno que el resto de compañeros.

El fichero Vagrantfile de nuestro proyecto actual 'songs-app-sinatra' está configurado como Version 2, y está formado por dos instrucciones:

```
config.vm.box = "http://boxes.gajdaw.pl/sinatra/sinatra-v1.0.0.box"
config.vm.network :forwarded_port, guest: 4567, host: 45670, host_ip: "127.0.0.1"
```

7.2 Imagen de una Máquina Virtual

La imagen usada por Vagrant para crear y ejecutar el sistema invitado está formada por un único fichero, al que se suele denominar como «box file», o fichero imagen. La extensión usada para este tipo de fichero es .box, pero no es obligatorio.

7.3 Iniciando una Máquina Virtual

Cuando se ejecuta el comando $ vagrant up la primera vez, veremos una salida similar a la siguiente:

```
1 Bringing machine 'default' up with 'virtualbox' provider...
1==> default: Box 'http://boxes.gajdaw.pl/sinatra/sinatra-v1.0.0.box' could not be␣
↪found. Attempting to find and install...
1    default: Box Provider: virtualbox
1    default: Box Version: >= 0
1==> default: Adding box 'http://boxes.gajdaw.pl/sinatra/sinatra-v1.0.0.box' (v0) for␣
↪provider: virtualbox
1    default: Downloading: http://boxes.gajdaw.pl/sinatra/sinatra-v1.0.0.box
1==> default: Box download is resuming from prior download progress
1==> default: Successfully added box 'http://boxes.gajdaw.pl/sinatra/sinatra-v1.0.0.
↪box' (v0) for 'virtualbox'!
2==> default: Importing base box 'http://boxes.gajdaw.pl/sinatra/sinatra-v1.0.0.box'..
↪.
3==> default: Matching MAC address for NAT networking...
3==> default: Setting the name of the VM: songs-app-sinatra_default_1425397277271_
↪42545
3==> default: Clearing any previously set network interfaces...
3==> default: Preparing network interfaces based on configuration...
3    default: Adapter 1: nat
3==> default: Forwarding ports...
3    default: 4567 => 45670 (adapter 1)
3    default: 22 => 2222 (adapter 1)
3==> default: Booting VM...
3==> default: Waiting for machine to boot. This may take a few minutes...
3    default: SSH address: 127.0.0.1:2222
3    default: SSH username: vagrant
3    default: SSH auth method: private key
3    default: Warning: Connection timeout. Retrying...
3    default:
3    default: Vagrant insecure key detected. Vagrant will automatically replace
3    default: this with a newly generated keypair for better security.
3    default:
3    default: Inserting generated public key within guest...
3    default: Removing insecure key from the guest if its present...
3    default: Key inserted! Disconnecting and reconnecting using new SSH key...
3==> default: Machine booted and ready!
3==> default: Checking for guest additions in VM...
3==> default: Mounting shared folders...
3    default: /vagrant => /examples/songs-app-sinatra
3==> default: Running provisioner: shell...
3    default: Running: inline script
3==> default: stdin: is not a tty
```

El proceso de arranque de la máquina está dividido en tres etapas:

- Etapa 1. Descarga e instalación de la 'box' en el sistema anfitrión. `~/.vagrant.d/boxes`

- Etapa 2. Importación de la 'box' al proyecto. `~/VirtualBox VMs/`
- Etapa 3. Arranque del sistema.

El nombre del directorio en el cual Vagrant almacena las imágenes descargadas, es el mismo que la URL, pero con el carácter / sustituído por `-VAGRANTSLASH-`.

Una vez finalizado el proceso de arranque de la nueva máquina virtual, se devuelve el control al prompt de nuestro sistema. No se aprecia ningún cambio, ya que la máquina se ha iniciado en modo 'headless', y no disponemos de interfaz GUI. Para comprobar el estado actual de la máquina recién iniciada, ejecutamos el siguiente comando:

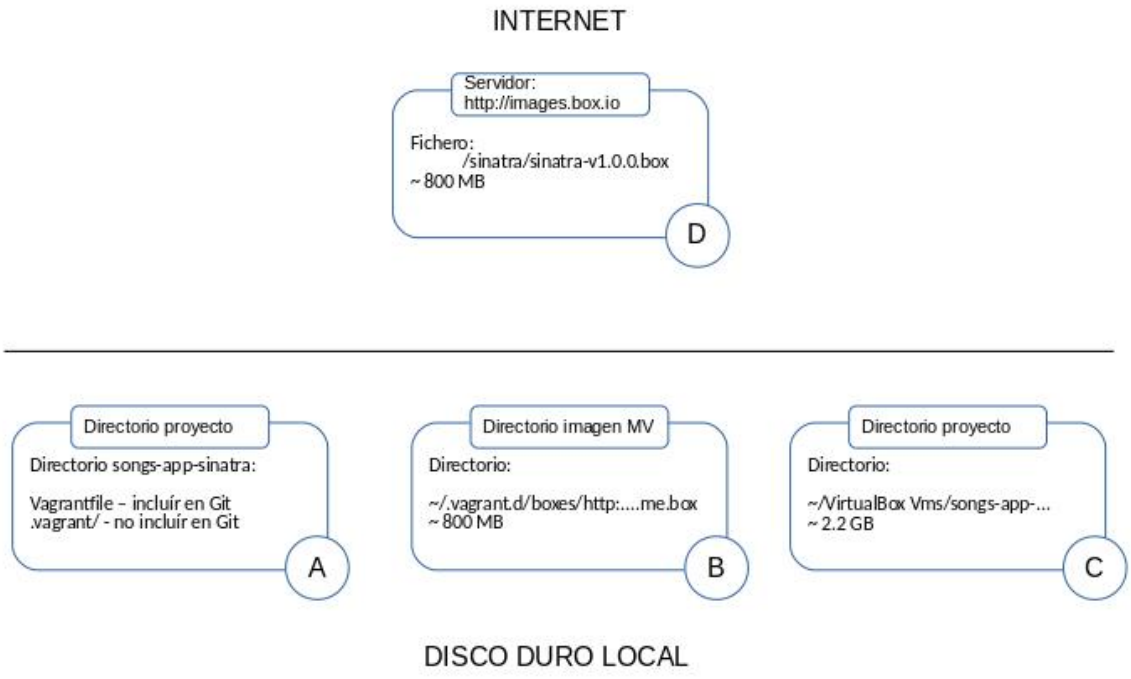
```
$ vagrant status
```

En este directorio existe un subdirectorio oculto con nombre '.vagrant', que es donde se almacena el ID y otra información sobre el sistema invitado. Está preparado para ser creado con el comando $ vagrant up y eliminado con $ vagrant destroy.

CAPÍTULO 8

Ficheros y directorios

La siguiente imagen muestra un resumen de los ficheros y directorios más importantes para `$ vagrant up`, y que nos ayudará a entender mejor el proceso de arranque de un sistema invitado.



El directorio etiquetado como A es donde se guarda el código fuente del proyecto (puede ser creado con el comando `$`

`git clone`). Es el lugar donde podemos ejecutar todos los comandos, tales como `$ vagrant up` o `$ vagrant status`. Aquí podemos encontrar el fichero Vagrantfile y el directorio .vagrant.

Durante la etapa 1, Vagrant descarga el fichero 'box' desde el servidor remoto y lo instala en el sistema dentro del directorio para las imágenes, que está etiquetado como B. Esta imagen puede ser reutilizada en varios proyectos. Puede resultar útil, que este directorio pueda ser accesible al resto de usuarios del sistema para aprovechar las imágenes descargadas.

En la etapa 2, la imagen guardada en el directorio B se copia y descomprime en el direcotrio C, que sólo será utilizado por el sistema invitado.

Y durante la etapa 3, de momento nos fijaremos únicamente en la creación del directorio .vagrant.

8.1 Estados de los sistemas invitados

Existen cinco estados en los que se puede encontrar un sistema invitado, y que se visualizan a través del comando *$ vagrant status*:

- running
- poweroff
- saved
- not created
- aborted

CAPÍTULO 9

Iniciando el proyecto Sinatra

Iniciamos la máquina invitada con el comando:

```
$ vagrant up
```

Para acceder a la aplicación a través del navegador anfitrión, debemos iniciar el servidor HTTP con los comandos siguientes:

```
$ vagrant ssh
$ ruby app.rb -o 0.0.0.0 &
$ logout
```

Si hemos finalizado nuestro trabajo para la jornada de hoy, debemos parar la máquina mediante el comando:

```
$ vagrant halt
```

Cuando necesitemos trabajar de nuevo con esta máquina, la volvemos a arrancar:

```
$ vagrant up
```

Si no tenemos acceso a nuestra página en el sistema invitado, debemos iniciar de nuevo el servidor HTTP.

Ahora, procedemos a la parada de la máquina, pero con el siguiente comando:

```
$ vagrant suspend
```

En este momento, el estado de la máquina es 'saved', y no consume memoria RAM en el sistema anfitrión. Para volver a iniciar el sistema, ejecutamos uno de los siguientes comandos:

```
$ vagrant up
$ vagrant resume
```

9.1 Modificar el puerto

Si el puerto 45670 está ocupado en nuestro sistema anfitrión, debemos realizar un cambio en el fichero Vagrantfile. Por ejemplo, podemos escribir la siguiente línea:

```
config.vm.network :forwarded_port, guest: 4567, host: 9090, host_ip: "127.0.0.1"
```

Aplicamos los cambios con el comando:

```
$ vagrant reload
```

9.2 Eliminar la imagen

Para visualizar las imágenes descargadas durante la ejecución de los comandos `$ vagrant up`, usamos el comando:

```
$ vagrant box list
```

Cada imagen, puede ser eliminada del sistema con el siguiente comando:

```
$ vagrant box remove NOMBRE
```

CAPÍTULO 10

Atlas

Para poder crear un nuevo proyecto que haga uso de un sistema operativo invitado, necesitamos una imagen con un sistema preinstalado. Esta imagen, que se denomina «base box» en la terminología Vagrant, puede ser creada desde cero o descargada desde Internet.

De momento, seguiremos utilizando imágenes descargadas desde Internet, y más adelante veremos como crearlas desde un CDROM, por ejemplo.

El Vagrantfile que se ha utilizado en los ejemplos anteriores, contiene una línea similar a la siguiente:

```
config.vm.box = "http://boxes.gajdaw.pl/sinatra/sinatra-v1.0.0.box"
```

Cuando se inicia la máquina, Vagrant descarga el fichero que aparece en la URL. A partir de la versión 1.6, Vagrant puede hacer uso de un servicio 'cloud' conocido como **Atlas** para descargar 'base boxes'. Este servicio, está disponible en https://atlas.hashicorp.com. Cada imagen se identifica por un nombre que está formado por dos partes: el nombre del fabricante y el nombre del 'box'.

Por ejemplo, el nombre *hashicorp/precise64* se refiere al 'box' creado por Hashicorp y que tiene como nombre *precise64*. Los detalles de este 'box' se puede consultar en https://atlas.hashicorp.com/hashicorp/boxes/precise64. Si queremos buscar otras imágenes diferentes, debemos escribir su nombre en la dirección https://atlas.hashicorp.com/boxes/search.

10.1 Iniciar un nuevo proyecto

El comando Vagrant que usaremos para iniciar un nuevo proyecto es `$ vagrant init`. Por ejemplo, para crear un nuevo proyecto que haga uso de la box ubuntu/trusty32, ejecutamos los siguientes comandos:

```
$ cd directorio/con/ejemplos
$ mkdir nuevo-proyecto
$ cd nuevo-proyecto
$ vagrant init -m ubuntu/trusty32
```

El objetivo del comando `$ vagrant init` es la creación del fichero de configuración *Vagrantfile*. La opción `-m` hace que el fichero *Vagrantfile* tenga un contenido mínimo. En caso de no utilizar esta opción, dicho fichero contendrá

multitud de líneas comentadas con diferentes opciones de configuración. Otra opción de este comando que nos puede resultar útil, es `-f`, que fuerza la creación del fichero *Vagrantfile* aún cuando exista una versión anterior.

CAPÍTULO 11

Seguridad

11.1 Riesgo #1

El primer riesgo potencial se inicia en la descarga de la imagen desde Internet. El uso de una 'base box' se traduce en el despliegue de un fichero empaquetado con un SO completo y que iniciaremos en nuestro equipo. Es decir, cuando se ejecuta el comando `$ vagrant up`, iniciamos un SO configurado por otra persona.

Por norma general, debemos usar 'boxes' con una fuente más o menos confiable, por ejemplo, *hashicorp*, *ubuntu* o *chef*.

11.1.1 Iniciando el SO invitado

Si ya hemos utilizado la 'box' ubuntu/trusty32 en alguno de nuestros proyectos anteriores, tendremos que eliminarla de nuestro sistema con el siguiente comando:

```
$ vagrant box remove ubuntu/trusty32
```

Verificamos que no está presente:

```
$ vagrant box list
$ vagrant box list | grep ubuntu
```

Ahora, si tenemos presente nuestro fichero 'Vagrantfile', e iniciamos el sistema invitado:

```
$ vagrant up
```

En la salida de este comando podemos observar la URL de descarga:

```
defautl: URL: https://atlas.hashicorp.com/ubuntu/trusty32
```

que la expande en la siguiente dirección:

```
default: Downloading: https://atlas.hashicorp.com/ubuntu/boxes/trusty32/versions/14.
↪04/providers/virtualbox.box
```

Si intentamos descargar la URL anterior con el comando curl:

```
$ curl https://atlas.hashicorp.com/ubuntu/boxes/trusty32/versions/14.04/providers/
↪virtualbox.box
```

obtenemos la siguiente respuesta:

```
<html>
   <body>
   You are being <a href="http://cloud-images.ubuntu.com/vagrant/trusty/current/
↪trusty-server-cloudimg-i386-vagrant-disk1.box">redirected</a>.
   </body>
</html>
```

De este modo podemos descargar la imagen que se utilizará en nuestro equipo.

11.1.2 Configuración por defecto de una MV

La configuración por defecto de una MV tiene las siguientes propiedades:

- Un puerto redirigido: El puerto 2222 del anfitrión está redirigido al puerto 22 del invitado.
- Directorio compartido: El directorio donde está guardado el fichero 'Vagrantfile' es compartido entre el anfitrión y el invitado.

Se puede verificar el estado de las configuraciones anteriores a través de la interfaz de administración de VirtualBox.

> Siempre que hagamos uso de la redirección de puertos de un sistema invitado, debemos asegurarnos de que sólo son accesible por el equipo anfitrión, a no ser que estemos seguros de que queremos aplicar otra configuración diferente.

11.2 Riesgo #2

Como hemos visto, Vagrant sólo publica un puerto del sistema invitado. No importan los servicios que puedan estar ejecutandose en el sistema invitado, el único 'daemon' accesible es sshd. Y además, sólo se permiten conexiones que se inicien en el sistema anfitrión.

Cuando activemos otros puertos, debemos replicar esta configuración. Nuestro sistema invitado no debe exponer ningún servicio a toda la red. Una forma de aplicar esta restricción es hacer uso del parámetro host_ip, dentro de la regla :forwarded_port:

```
config.vm.network :forwarded_port, guest: 80, host: 8888, host_ip: "127.0.0.1"
```

11.2.1 Directorios compartidos

Cuando ejecutamos `$ vagrant up`, se comparte el directorio del proyecto, como se puede comprobar en la salida del comando:

```
==> default: Mounting shared folders...
default: /vagrant => /ejemplos/nuevo-proyecto
```

El directorio /ejemplos/nuevo-proyecto, es accesible desde el sistema invitado a través del directorio /vagrant.

```
Directorio anfitrión                                    Directorio invitado
/ejemplos/nuevo-proyecto             <===>              /vagrant
```

La definición de un directorio compartido se refleja en Vagrant a través de la siguiente configuración:

```
config.vm.synced_folder [HOST-PATH], [GUEST-PATH]
```

Ejercicio:

Comprobar la configuración de directorios compartidos en el proyecto *songs-angularjs-app*.

11.2.2 Trabajando con SSH

El método más habitual de uso de SSH es a través de la línea de comandos, por ejemplo:

```
$ ssh bob@example.net
```

Cuando ejecutamos un sistema invitado, podemos iniciar una sesión SSH de una forma más simple $ vagrant ssh. Este proceso es muy simple porque hace uso de diferentes valores que ya están configurados. Estes valores pueden ser comprobados con el siguiente comando:

```
$ vagrant ssh-config
```

En el caso de que tengamos varias máquinas invitadas corriendo al mismo tiempo, Vagrant se encarga de evitar colisiones en los puertos utilizados para conectarnos con las mismas. Por ejemplo, una segunda máquina invitada tendrá acceso a través del puerto 2200, en lugar del 2222, que ya está ocupado.

De todas formas, siempre se puede utilizar el método clásico de SSH para conectarnos a nuestro sistema invitado:

```
$ ssh -p2222 vagrant@127.0.0.1
```

Las credenciales son las siguientes:

```
username: vagrant
password: vagrant
```

11.2.3 Gestionando diferentes sistemas invitados

Cuando trabajamos con sesiones SSH en diferentes máquinas invitadas, podemos llegar a un momento de confusión, y no saber exactamente en que máquina estamos conectados. Para ayudarnos en la tarea de reconocimiento de la máquina invitada, podemos ejecutar el siguiente comando (dentro del invitado):

```
$ guestvm
```

11.3 Riesgo #3

Como ya habremos advertido en las secciones anteriores, las credenciales usadas por defecto en los sistemas invitados, suponen otro riesgo para la seguridad de nuestros sistemas. Sobre todo, porque el usuario 'vagrant' posee permisos de sudo. La única restricción es que se debe iniciar la sesión desde la máquina anfitrión.

Para mitigar este riesgo, debemos modificar la contraseña del usuario 'vagrant':

```
$ passwd
```

11.3.1 Acceso SSH con llave privada

Aparte del método de acceso SSH a través de las credenciales anteriores, también podemos iniciar una sesión haciendo uso de llaves RSA: una pública y una privada, guardadas en diferentes ficheros.

```
~/.ssh/id_rsa     - private key
~/.ssh/id_rsa.pub - public key
```

Este es el método utilizado por $ vagrant ssh, con la siguiente configuración:

- La llave privada se guarda en el sistema anfitrión en `~/.vagrant.d/insecure_private_key`.
- La llave pública se guarda en el sistema invitado en `/home/vagrant/.ssh/authorized_keys`.

Ejercicio:

Crear un nuevo par de llaves para reemplazar/añadir a las creadas por defecto.

11.4 Riesgo #4

Todas las imágenes que se descargan de Internet incorporan las mismas llaves, por lo que cualquier usuario tiene acceso a ese par de llaves. Las últimas versiones de Vagrant minimizan este riesgo creando un nuevo par durante la creación del sistema invitado `$ vagrant up`.

11.4.1 Recargar el sistema invitado

Cuando aplicamos algún cambio en el fichero 'Vagrantfile', para que tengan efecto en el sistema invitado debemos ejecutar el siguiente comando:

```
$ vagrant reload
```

Por ejemplo, podemos modificar el puerto de escucha del proyecto 'Songs for kids', hacia el 9876:

```
config.vm.network :forwarded_port, guest: 80, host: 9876, host_ip: "127.0.0.1"
```

Una vez que recarguemos nuestro sistema, la aplicación estará disponible en http://127.0.0.1:9876.

CAPÍTULO 12

Nuestra primera imagen

En esta sección veremos como crear una imagen a partir de Ubuntu 14.04, sobre la que instalaremos cierto software adicional. Una vez completada la instalación y configuración, la exportaremos a un fichero.

Nos situamos en el escenario en el cual se nos encarga la creación de una imagen para poder crear un blog compartido por el equipo de comunicación. Usaremos Jekyll como herramienta de gestión de los contenidos del blog.

Por lo tanto, el entorno de desarrollo a configurar debe soportar las siguientes características:

- Disponer de las herramientas para procesar los ficheros HTML.
- Poder consultar el resultado del blog a través de la dirección http://localhost:8080.

Se llevarán a cabo diferentes actividades en dos directorios separados:

- ~/box-factory/
- ~/corporate-blog/

En un primer momento, trabajaremos dentro del primer directorio para diseñar la imagen. Este trabajo será ejecutado por los responsables de sistemas. Y, en una segunda etapa, trabajaremos en el segundo directorio, haciendo uso de la imagen creada en el paso anterior, y será gestionado por el equipo de desarrollo.

12.1 Elección de 'base box' y puesta en marcha

Cuando tenemos que crear una nueva imagen, la solución más rápida es partir de una imagen ya existente. Por ejemplo, si queremos preparar un entorno de desarrollo que haga uso de Ubuntu, podemos partir de las imágenes soportadas por Ubuntu:

- ubuntu/trusty64
- ubuntu/trusty32
- ubuntu/precise64
- ubuntu/precise32

En caso de decidirnos por CentOS, Debian, Fedora o FreeBSD, podemos usar las siguientes imágenes:

- centos/7
- debian/stretch64
- fedora/26-cloud-base
- freebsd/FreeBSD-11.1-STABLE

Para nuestro ejercicio haremos uso de Ubuntu 14.04, pero la estrategia es similar para otras distribuciones, y sólo cambiará el modo de instalar software en cada una de ellas.

Empezamos por crear el directorio de trabajo:

```
$ cd
$ mkdir box-factory
$ cd box-factory
```

Iniciamos un nuevo proyecto Vagrant:

```
$ vagrant init -m ubuntu/trusty32
```

Levantamos la MV:

```
$ vagrant up
```

12.2 Instalación del software necesario

Con el SO preparado, iniciamos una sesión SSH para instalar el software necesario. A mayores, instalaremos el navegador Lynx para comprobar el estado del sitio web desde la sesión SSH.

```
$ vagrant ssh
# Invitado
$ sudo apt-add-repository ppa:brightbox/ruby-ng
$ sudo apt-get update
$ sudo apt-get install ruby2.1 ruby2.1-dev lynx-cur
$ sudo gem2.1 install jekyll
```

Una vez instalado el software podemos comprobar la versión de Jekyll disponible:

```
$ jekyll --version
```

Cerramos la sesión SSH, y ya tenemos nuestra MV para poder empaquetar.

12.3 Creando la imagen

Una vez que disponemos del sistema operativo invitado en ejecución, y que incluye el software necesario, es el momento de crear la imagen:

```
# Anfitrión
$ vagrant package --output box-jekyll.box
```

12.4 Consultar, instalar y eliminar 'boxes'

A partir de este momento, tomaremos el papel de desarrollador. Tal y como se vió en secciones anteriores, las imágenes deben estar descargadas en nuestro sistema antes de poder utilizarse en Vagrant. La gestión de las imágenes la llevamos a cabo con los siguientes comandos:

```
# Anfitrión
$ vagrant box list
$ vagrant box add
$ vagrant box remove
```

Para añadir una nueva imagen a nuestro directorio .vagrant.d/boxes tenemos que usar el comando $ vagrant box add con dos parámetros, como se muestra a continuación:

```
# Anfitrión
$ vagrant box add [NOMBRE] [DIRECCION]
```

El **[NOMBRE]** es la etiqueta que deseamos asignarle a la nueva imagen, y la **[DIRECCION]** apunta al fichero que contiene la imagen (con extensión .box).

Por lo tanto, para poder añadir nuestra imagen recién creada:

```
$ vagrant box add box-jekyll box-jekyll.box
```

Y, para eliminar una imagen, usamos el siguiente comando:

```
$ vagrant box remove [NOMBRE]
```

12.5 Usando la imagen

En este momento, ya estamos en disposición de usar la imagen recién añadida a nuestro sistema. Creamos nuestro nuevo directorio de trabajo, y creamos el nuevo proyecto:

```
$ cd
$ mkdir corporate-blog
$ cd corporate-blog
$ vagrant init -m box-jekyll
$ vagrant up
$ vagrant ssh

# Invitado
$ cd /vagrant
$ jekyll new -f .
$ jekyll build
$ jekyll serve -H 0.0.0.0 --detach
$ lynx 127.0.0.1:4000
```

Ejercicio:

Configurar la redirección de puertos para que se pueda consultar el blog desde el sistema anfitrión a través del puerto 8080.

CAPÍTULO 13

Aprovisionamiento

En la sección anterior hemos visto el proceso para configurar una imagen que cumpla con nuestras necesidades, a partir de una 'base box' de origen. Ahora, veremos como automatizar ese proceso de configuración de nuestro entorno. Lo que haremos es guardar los comandos que se deben ejecutar durante el arranque de nuestra máquina en un script. Este proceso automático es lo que se conoce como 'aprovisionamiento'.

Vagrant soporta diferentes herramientas de configuración, como por ejemplo:

- Shell scripts
- Puppet
- Ansible
- Chef

13.1 Proveedores

Las herramientas que se utilizan durante la etapa de aprovisionamiento, se conocen como 'provisioners' (proveedores). Cualquier comando de los que solemos ejecutar en una sesión SSH pueden ser ejecutadas por el proveedor. Configuración del aprovisionamiento

La configuración se guarda en el fichero 'Vagrantfile'. El primer parámetro es el nombre del proveedor que se va utilizar. En nuestro ejemplo, usaremos un script de shell:

```
Vagrant.configure(2) do |config|
   ...
   config.vm.provision "shell", path: "script.sh"
end
```

13.2 Múltiples proveedores

Es posible el uso de diferentes proveedores para configurar nuestra máquina. Por ejemplo, podemos usar Shell, Puppet y Ansible en el mismo proyecto:

```
Vagrant.configure(2) do |config|
   ...
      config.vm.provision "shell", path: "script.sh"
      config.vm.provision "puppet"
      config.vm.provision "ansible", playbook: "playbook.yml"
end
```

13.3 ¿Cuando se ejecuta el aprovisionamiento?

Por defecto, sólo se ejecuta durante la primera ejecución de `$ vagrant up`. Si detenemos el sistema con $ vagrant halt o $ vagrant suspend, la próxima ejecución de $ vagrant up omitirá la etapa de aprovisionamiento. Aunque podemos modificar ese comportamiento con las opciones –provision y –no-provision.

```
$ vagrant up --provision
$ vagrant up --no-provision
```

13.4 Control de versiones de las 'boxes'

Los ficheros de aprovisionamiento que se van utilizar sufrirán diferentes cambios para que se puedan adaptar a la configuración deseada para nuestra imagen. Con la intención de poder controlar las diferentes versiones de dichos ficheros, y poder etiquetarlas, haremos uso de 'git' para la gestión de dichas versiones.

13.5 Jekyll box aprovisionada con Shell Script

En apartados anteriores se usaron diferentes comandos para instalar Jekyll en nuestra 'box' personalizada. Ahora, vamos a automatizar este proceso mediante un script de Shell. La idea es que después de la ejecución del comando `$ vagrant up`, ya tengamos una imagen con Jekyll instalado, y listo para poder utilizar.

Creamos el directorio de trabajo vacío:

```
$ cd /directorio/para/ejercicios
$ mkdir jekyll-shell
```

En este nuevo directorio creamos dos ficheros:

- Vagrantfile
- script.sh

Se deja como tarea la escritura del contenido de dichos ficheros.

Cuando se hayan completado ambos ficheros, tan solo tendremos que ejecutar el comando `$ vagrant up`, para poder disponer de una máquina personalizada con Jekyll.

A continuación, comprobamos en el sistema invitado si podemos usar Jekyll para crear un nuevo blog:

```
$ vagrant ssh
$ jekyll --version
```

Cuando hayamos realizado las comprobaciones de que la configuración de la máquina es correcta, es momento de guardar una versión de nuestros ficheros de configuración:

```
$ git init
$ echo .vagrant > .gitignore
$ echo "*.box" >> .gitignore
$ git add -A
$ git commit -m "Version inicial"
$ git tag -a v0.1.0 -m "Release 0.1.0"
$ git log --oneline --decorate
```

Ejercicio:

Crear una nueva 'box' a partir de la imagen creada en el paso anterior.

CAPÍTULO 14

Creando imágenes desde cero

Hasta el momento, hemos trabajado con ficheros con extensión **.box* que contenían un SO preconfigurado, y que una vez descargados se podían empezar a usar con Vagrant. Pero la pregunta es: ¿Cómo se crean esas imágenes?. Que pasos debemos seguir si queremos usar un SO del cual no encontramos una imagen. En este capítulo veremos como crear una imagen a partir de una imagen ISO, como las que distribuyen en CD/DVD los mantenedores de las distribuciones Linux.

14.1 Packer

Packer es una herramienta de línea de comandos creada por HashiCorp, y cuyo propósito es automatizar la creación de MVs con diferentes sistemas operativos y proveedores. La usaremos para generar imágenes de Vagrant. Lo que hace es descargarse la imagen ISO del SO que queremos instalar. Ejecuta el programa de instalación y se aplica la configuración por defecto. Al finalizar, Packer usa una herramienta de aprovisionamineto (shell, Chef, Puppet o Ansible) para personalizar el sistema. Y finalmente, exporta el sistema para convertirlo en un fichero '.box'.

14.1.1 Instalación de Packer

Packer está disponible para su descarga en https://www.packer.io/downloads.html. Una vez descargado, se descomprime el fichero, dentro del cual está el binario que debemos hacer accesible dentro de las rutas de la variable PATH. Una vez finalizada la instalación, debemos comprobar su correcto funcionamiento:

```
$ packer version
```

14.2 Creando imágenes con chef/bento

El proyecto más popular con definiciones para Packer es chef/bento, que está disponible en Github: https://github.com/chef/bento.

```
# Host
$ cd directorio/para/ejemplos
$ git clone https://github.com/chef/bento.git
$ cd bento
```

Para proceder con la construcción de la imagen de Ubuntu 14.04 para VirtualBox, ejecutamos la siguiente orden:

```
$ packer build --only=virtualbox-iso ubuntu-14.04-i386.json
```

Una vez finalizada la ejecución, estará disponible el siguiente fichero en nuestro sistema:

```
bento/builds/ubuntu-14.04-i386.virtualbox.box
```

Para poder utilizar la imagen recién creada, debemos añadirla a nuestra lista de 'boxes' en Vagrant.

Ejercicio. Incorporar esta imagen a nuestra lista de boxes disponibles, y crear una MV a partir de esta nueva imagen.

14.3 Personalizar las imágenes creadas con chef/bento

Para poder crear una versión modificada de cualquiera de las 'boxes' incluídas en el proyecto 'chef/bento', debemos añadir un nuevo script de aprovisionamiento. Vamos a personalizar las imágenes de Ubuntu con el siguiente contenido en el fichero packer/scripts/ubuntu/customize.sh:

```
#!/bin/bash -eux

apt-get install git
apt-get install mc
```

A continuación, debemos añadir una referencia a dicho script para que sea ejecutado durante el proceso de generación de la nueva imagen, en el fichero JSON de Ubuntu 14.04 (i386).

De este modo, en la siguiente construcción de una imagen de Ubuntu, llevará incorporado 'git' y 'mc'.

A mayores, también podemos modificar otros parámetros de la imagen a generar. Por ejemplo, podemos modificar las credenciales del usuario creado por defecto. Estos datos son gestionados a través del fichero 'pressed.cfg'.

CAPÍTULO 15

Licencia

15.1 Licencia

15.1.1 Reconocimiento-CompartirIgual 3.0 España (CC BY-SA 3.0 ES)

Esto es un resumen inteligible para humanos (y no un sustituto) de la licencia, disponible en los idiomas siguientes: Aranés Asturiano Castellano Euskera Galego

Usted es libre de:

Compartir - copiar y redistribuir el material en cualquier medio o formato

Adaptar - remezclar, transformar y crear a partir del material para cualquier finalidad, incluso comercial

El licenciador no puede revocar estar libertades mientras cumpla los términos de la licencia.

Bajo las condiciones siguientes:

Reconocimiento - Debe reconocer adecuadamente la autoría[1], proporcionar un enlace a la licencia e indicar si se han realizado cambios[2]. Puede hacerlo de cualquier manera razonable, pero no de una manera que sugiera que tiene el apoyo del licenciador, o lo recibe por el uso que hace.

[1] Si se facilitan, debe proporcionar el nombre del creador y las partes a reconocer, un aviso de derechos de explotación, un aviso de licencia, una nota de descargo de responsabilidad y un enlace al material. Las licencias CC anteriores a la versión 4.0 también requieren que facilite el título del material, si le ha sido facilitado, y pueden tener algunas otras pequeñas diferencias.

[2] En la versión 3.0 y anteriores, la indicación de los cambios sólo se requiere si se crea una obra derivada.

Compartir Igual - Si remezcla, transforma o crea a partir del material, deberá difundir sus contribuciones bajo la misma licencia que el original[3].

No hay restricciones adicionales - No puede aplicar términos legales o medidas tecnológicas que legalmente restrinjan realizar aquello que la licencia permite.

[3] También puede utilizar una licencia compatible de la lista de https://creativecommons.org/compatiblelicenses.