Índice general

1 Vagrant

1.1 Instalación de Vagrant en Ubuntu

Actualizamos la lista de paquetes:

```
sudo apt-get update
```

Instalamos Vagrant:

```
sudo apt-get install vagrant
```

También es necesario tener instalado VirtualBox.

```
sudo apt-get install virtualbox
```

1.2 Verificamos la instalación

Comprobamos que Vagrant se ha instalado correctamente consultando la versión:

```
vagrant --version
```

1.3 Comandos útiles

1.3.1 help

Muestra todos los comandos que podemos utilizar con Vagrant.

```
vagrant help
```

1.3.2 up

Inicia la máquina y realiza el aprovisionamiento.

```
vagrant up
```

1.3.3 reload

Reinicia la máquina. Se utiliza cuando hacemos cambios en el archivo `Vagrantfile` y queremos que se apliquen los cambios.

```
vagrant reload
```

1.3.4 provision

Ejecuta sólo la sección de los *provisioners* del archivo `Vagrantfile`. Se utiliza cuando hacemos cambios en los scripts de los *provisioners* y queremos que se apliquen los cambios.

```
vagrant provision
```

1.3.5 init

Inicializa un nuevo archivo `Vagrantfile`.

```
vagrant init
```

1.3.6 halt

Apaga la máquina virtual.

```
vagrant halt
```

1.3.7 destroy

Elimina una máquina virtual.

```
vagrant destroy
```

1.3.8 suspend

Suspende la ejecución de la máquina virtual.

```
vagrant suspend
```

1.3.9 resume

Reanuda la ejecución de una máquina virtual que estaba suspendida.

```
vagrant resume
```

1.3.10 ssh

Nos conectamos por ssh a una máquina virtual. No es necesario introducir ningún password para conectar.

```
vagrant ssh
```

1.3.11 status

Muestra información sobre el estado actual del entorno Vagrant. Se utilizar para comprobar si una máquina virtual se está ejecutando.

```
vagrant status
```

1.3.12 global-status

```
vagrant global-status
```

Por ejemplo, si tenemos más de un entorno Vagrant en ejecución tendremos una salida similar a la que se muestra a continuación.

```
id       name    provider   state    directory
-------------------------------------------------------------------------------

177ee49  default virtualbox poweroff /home/josejuan/vagrant/apache-1
5726eb4  default virtualbox poweroff /home/josejuan/vagrant/apache-2

The above shows information about all known Vagrant environments
on this machine. This data is cached and may not be completely
up-to-date. To interact with any of the machines, you can go to
that directory and run Vagrant, or you can use the ID directly
with Vagrant commands from any directory. For example:
"vagrant destroy 1a2b3c4d"
```

2 Primeros pasos con Vagrant

2.1 Creación de un archivo `Vagrantfile`

Lo primero que necesitamos es un archivo `Vagrantfile` que podemos editarlo manualmente o generarlo automáticamente, que es lo que vamos a hacer en este primer ejemplo.

Vamos a crear un directorio para nuestros experimentos con Vagrant y accedemos a él.

```
mkdir vagrant-experiments
cd vagrant-experiments
```

Utilizamos el comando `vagrant init` para crear un archivo `Vagrantfile` automáticamente:

```
vagrant init
```

El comando anterior nos crea un archivo `Vagrantfile` en el mismo directorio donde nos encontramos, con el siguiente contenido:

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

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure("2") do |config|
  # The most common configuration options are documented and commented
     below.
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search
     for
  # boxes at https://vagrantcloud.com/search.
  config.vm.box = "base"

  # Disable automatic box update checking. If you disable this, then
  # boxes will only be checked for updates when the user runs
  # `vagrant box outdated`. This is not recommended.
  # config.vm.box_check_update = false

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example
     below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
  # NOTE: This will enable public access to the opened port
  # config.vm.network "forwarded_port", guest: 80, host: 8080

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine and only allow
     access
  # via 127.0.0.1 to disable public access
  # config.vm.network "forwarded_port", guest: 80, host: 8080, host_ip:
     "127.0.0.1"

  # Create a private network, which allows host-only access to the machine
```

```
  # using a specific IP.
  # config.vm.network "private_network", ip: "192.168.33.10"

  # Create a public network, which generally matched to bridged network.
  # Bridged networks make the machine appear as another physical device on
  # your network.
  # config.vm.network "public_network"

  # Share an additional folder to the guest VM. The first argument is
  # the path on the host to the actual folder. The second argument is
  # the path on the guest to mount the folder. And the optional third
  # argument is a set of non-required options.
  # config.vm.synced_folder "../data", "/vagrant_data"

  # Provider-specific configuration so you can fine-tune various
  # backing providers for Vagrant. These expose provider-specific options.
  # Example for VirtualBox:
  #
  # config.vm.provider "virtualbox" do |vb|
  #   # Display the VirtualBox GUI when booting the machine
  #   vb.gui = true
  #
  #   # Customize the amount of memory on the VM:
  #   vb.memory = "1024"
  # end
  #
  # View the documentation for the provider you are using for more
  # information on available options.

  # Enable provisioning with a shell script. Additional provisioners such
     as
  # Puppet, Chef, Ansible, Salt, and Docker are also available. Please see
      the
  # documentation for more information about their specific syntax and use
     .
  # config.vm.provision "shell", inline: <<-SHELL
  #   apt-get update
  #   apt-get install -y apache2
  # SHELL
end
```

Todas las opciones de configuración aparecen comentadas excepto una:

```
config.vm.box = "base"
```

que es donde podemos indicar el **box** que queremos utilizar para crear nuestra máquina virtual. La lista de **boxes** que podemos utilizar está disponible en la siguiente url:

https://app.vagrantup.com/boxes/search

En nuestras prácticas vamos a utilizar el **box**: `ubuntu/xenial64`.

Para usar esta **box**, tenemos dos opciones:

1) Editar el archivo `Vagrantfile` y cambiar el valor de `config.vm.box = "base"` por `config.vm.box = "ubuntu/xenial64"`.
2) Ejecutar el comando `vagrant init ubuntu/xenial64`.

En mi caso voy a utilizar la segunda opción:

```
vagrant init ubuntu/xenial64
```

Ahora nuestro archivo `Vagrantfile` tiene el siguiente contenido:

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

Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/xenial64"
end
```

2.2 Iniciamos la máquina virtual

Desde el mismo directorio donde tenemos creado el archivo `Vagrantfile` ejecutamos el comando:

```
vagrant up
```

La primera vez que ejecutamos Vagrant con una **box** nueva, se descargará y se importará en nuestro sistema. El tiempo de descarga de la **box** dependerá de nuestra conexión a Internet. Una vez descargada e importada en nuestro sistema no se volverá a descargar y el tiempo de espera para iniciar una máquina virtual será más breve.

2.3 Nos conectamos a la máquina por `ssh`

Una vez que la máquina virtual está en ejecución nos podemos conectar a ella por `ssh` ejecutando el comando:

```
vagrant ssh
```

3 Configuración del archivo `Vagrantfile`

3.1 Configuración de la red

Por defecto Vagrant crea una red NAT entre la máquina virtual y el host anfitrión. Con esta configuración de red podremos tener conexión a Internet desde la máquina virtual pero la máquina virtual no será visible fuera de la red NAT.

Si queremos que la máquina virtual sea accesible por las máquinas de mi red local deberemos utilizar la configuración de red privada.

Ejemplo:

```
config.vm.network "private_network", ip: "192.168.33.10"
```

Puedes encontrar más información en la web oficial.

3.2 Configuración de los *provisioners*

Vagrant puede trabajar con diferentes herramientas de automatización de aprovisionamiento como: Puppet, Ansible, Salt y Chef. Para tareas sencillas también podemos hacer uso de la `shell`. Lo que harán todas estas herramientas será ejecutar una serie de comandos para instalar el software y los paquetes necesarios en nuestra máquina virtual.

Ejemplo de aprovisionamiento para una máquina con **Apache**:

```
config.vm.provision "shell", inline: <<-SHELL
  apt-get update
  apt-get install -y apache2
```

```
SHELL
```

También podemos crear un script que incluya todos los comandos que queremos ejecutar y pasarlo como parámetro en `inline`.

Ejemplo:

```
config.vm.provision "shell", path: "script.sh"
```

En este caso `script.sh` será un archivo ubicado en el mismo directorio del archivo Vagrantfile.

3.3 Directorios compartidos entre el *host* anfitrión y la máquina virtual

Podemos compartir un directorio entre el host anfitrión y la máquina virtual de una forma sencilla.

```
config.vm.synced_folder "./", "/vagrant"
```

El primer parámetro indica la ruta del directorio compartido en el host anfitrión y el segundo parámetro el directorio donde aparecerá en la máquina virtual.

4 Ejemplos

4.1 Ejemplos disponibles en GitHub

Puede encontrar una colección de configuraciones de ejemplos en el siguiente repositorio en GitHub:

- https://github.com/josejuansanchez/iaw-practica-vagrant

4.2 Vagrantfile para dos máquinas virtuales: Apache y MySQL

Vagrantfile

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

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

```
  config.vm.box = "ubuntu/xenial64"

  # Apache HTTP Server
  config.vm.define "web" do |app|
    app.vm.hostname = "web"
    app.vm.network "public_network"
    app.vm.provision "shell", path: "provision-for-apache.sh"
  end

  # MySQL Server
  config.vm.define "db" do |app|
    app.vm.hostname = "db"
    app.vm.network "public_network"
    app.vm.provision "shell", path: "provision-for-mysql.sh"
  end

end
```

provision-for-apache.sh

```
#!/bin/bash
apt-get update
apt-get install -y apache2
apt-get install -y php libapache2-mod-php php-mysql
sudo /etc/init.d/apache2 restart
cd /var/www/html
wget https://github.com/vrana/adminer/releases/download/v4.3.1/adminer
    -4.3.1-mysql.php
mv adminer-4.3.1-mysql.php adminer.php
```

provision-for-mysql.sh

```
#!/bin/bash
apt-get update
apt-get -y install debconf-utils

DB_ROOT_PASSWD=root
debconf-set-selections <<< "mysql-server mysql-server/root_password
    password $DB_ROOT_PASSWD"
```

```
debconf-set-selections <<< "mysql-server mysql-server/root_password_again
    password $DB_ROOT_PASSWD"

apt-get install -y mysql-server
sed -i -e 's/127.0.0.1/0.0.0.0/' /etc/mysql/mysql.conf.d/mysqld.cnf
/etc/init.d/mysql restart

mysql -uroot mysql -p$DB_ROOT_PASSWD <<< "GRANT ALL PRIVILEGES ON *.* TO
    root@'%' IDENTIFIED BY '$DB_ROOT_PASSWD'; FLUSH PRIVILEGES;"
```

5 Referencias

- **Vagrant CookBook**. Erika Heidi. Leanpub.
- **Vagrant: Up and Running**. Mitchel Hashimoto.
- debconf. Sistema de configuración de paquetes de Debian.

6 Licencia