

Procedimiento

Índice

1. [Idea Principal](#)
2. [Instalación](#)
 - 2.1. [Vagrant](#)
 - 2.2. [VirtualBox](#)
3. [Boxes](#)
 - 3.1. [Añadir un box](#)
 - 3.2. [Lanzar un box](#)
 - 3.3. [Directorio Sincronizado](#)
 - 3.4. [Reempaquetar un box](#)
 - 3.5. [Eliminación e instalación de una imagen local](#)
 - 3.6. [Actualización de una imagen](#)
4. [Configuración Vagrantfile](#)
 - 4.1. [Configuración Simple](#)
 - 4.2. [Interfaz Gráfica](#)
 - 4.3. [Aprovisionamiento Ligero](#)
 - 4.4. [Redirección de puertos](#)
 - 4.5. [Configuración red privada](#)
 - 4.6. [Configuración red pública](#)
 - 4.7. [Multimáquinas](#)
5. [Ejercicio Final](#)
6. [Conclusiones](#)
7. [Bibliografía](#)

1. Idea Principal

La idea principal de este documento es hacer la instalación de **Vagrant**, seguidamente hacer una serie de ejemplos prácticos para finalmente hacer un ejercicio final donde se aplican los ejemplos explicados anteriormente.

2. Instalación

Como se ha comentado anteriormente, para usar **Vagrant** se necesita tener, al menos, un sistema de virtualización, en estos ejercicios se utilizará el sistema de virtualización inicial con el cual se desarrolló **Vagrant** que es [VirtualBox](#) y que además es donde se ofrece más funcionalidad.

2.1. Vagrant

La instalación de vagrant es muy sencilla, en el caso de sistemas operativos como **Ubuntu**, **Debian**, **CentOS**, **Fedora**, **Amazon Linux** o **Homebrew** lo único que tenemos que hacer es:

- Añadir el repositorio
- Instalar Vagrant

```
$ sudo dnf install -y dnf-plugins-core
$ sudo dnf config-manager --add-repo https://rpm.releases.hashicorp.com/fedora/hashicorp.repo
$ sudo dnf -y install vagrant
```

2.2. VirtualBox

- Para la instalación de VirtualBox, tenemos que ir a la página web de [VirtualBox](#), aquí encontraremos las diferentes opciones de descarga dependiendo del sistema operativo y la distribución.
- En este caso lo instalaremos para un Fedora 32.

```
$ wget https://download.virtualbox.org/virtualbox/6.1.22/VirtualBox-6.1-6.1.22_144080_fedora32-1.x86_64
$ sudo rpm -i VirtualBox-6.1-6.1.22_144080_fedora32-1.x86_64.rpm
```

- Nos añadimos al grupo de virtualbox por tal de poder ejecutarlo correctamente:

```
$ sudo usermod -a -G vboxusers ${USER}
```

Una vez hecha la instalación y los demás pasos tendremos que ejecutar el setup que proporciona VirtualBox:

```
$ sudo /usr/lib/virtualbox/vboxdrv.sh setup
```

3. Boxes

El funcionamiento de **Vagrant** consiste en dividir la distribución de la aplicación en dos, el sistema operativo en un determinado formato, que viene a ser, entre otras cosas, la imagen de este, y por otra

lado distribuimos la configuración y modificaciones de este escenario en un fichero de texto plano, que viene a ser **Vagrantfile**.

Vagrant tiene su propia [comunidad](#) en la cual todo el mundo puede subir sus boxes y ahí es donde se pueden obtener los boxes o también podemos crear nuestro propio box con ayuda de otro software, por ejemplo **packer**.

3.1. Añadir un box

Para añadir un box, **Vagrant** nos proporciona el subcomando **"box"** que a su vez tiene más subcomandos y entre ellos encontramos **"add"** que es los que nos permite descargar el box que queramos:

```
$ vagrant box add [nombre del box] [opciones]
```

Como hemos dicho antes los boxes que **Vagrant** descarga, a no ser que se especifique que los queremos de una ruta local, se obtienen desde su página web de su [comunidad](#), y el nombre de estos boxes se rigen por: **username/boxname**

Ejemplo:

```
[didi@localhost projecte-edt]$ vagrant box add ubuntu/trusty64
==> box: Loading metadata for box 'ubuntu/trusty64'
      box: URL: https://vagrantcloud.com/ubuntu/trusty64
==> box: Adding box 'ubuntu/trusty64' (v20190514.0.0) for provider: virtualbox
      box: Downloading: https://vagrantcloud.com/ubuntu/boxes/trusty64/versions/20190514.0.0/providers/virtualbox.box
Download redirected to host: cloud-images.ubuntu.com
==> box: Successfully added box 'ubuntu/trusty64' (v20190514.0.0) for 'virtualbox'!
[didi@localhost projecte-edt]$ vagrant box list | grep ubuntu/trusty64
ubuntu/trusty64 (virtualbox, 20190514.0.0)
```

Hay que tener en cuenta que los boxes que se suben a la comunidad de Vagrant no están verificados, por lo tanto es necesario que tengamos cuidado al momento de qué box descargar

3.2. Lanzar un box

Para el arranque de un box que hayamos descargado anteriormente tenemos que usar el subcomando **"init"** dentro del subcomando de *****"box"*****, esto lo que hará es crear un fichero **Vagrantfile** en el cual se encuentra la configuración base para poder, posteriormente, arrancar nuestro box con `vagrant up`, por lo tanto, tenemos que tener en cuenta que el uso de este fichero de configuración que nos permite arrancar el box tiene que estar alojado en un directorio y en este directorio no pueden ir otros ficheros de configuración **Vagrantfile**.

En conclusión, por cada box que queramos arrancar necesitaremos un directorio diferente, esto nos permite poder tener una mejor organización de los boxes que tengamos activos.

```
[didi@localhost pruebal]$ vagrant init -m ubuntu/trusty64
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
[didi@localhost pruebal]$ cat Vagrantfile
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
end
```

Como podemos ver en la imagen, hemos hecho un `vagrant init` de un box llamado "ubuntu/trusty64" y automáticamente nos ha creado un fichero **Vagrantfile** en el cual especifica que al momento de hacer `vagrant up` se usará como *box* el box que hemos especificado.

```
[didi@localhost pruebal]$ vagrant up --provider virtualbox
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'ubuntu/trusty64'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'ubuntu/trusty64' version '20190514.0.0' is up to date...
==> default: Setting the name of the VM: pruebal_default_1620227330811_14620
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
default: Adapter 1: nat
==> default: Forwarding ports...
default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
default: SSH address: 127.0.0.1:2222
default: SSH username: vagrant
default: SSH auth method: private key
default:
default: Vagrant insecure key detected. Vagrant will automatically replace
default: this with a newly generated keypair for better security.
default:
default: Inserting generated public key within guest...
default: Removing insecure key from the guest if it's present...
default: Key inserted! Disconnecting and reconnecting using new SSH key...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
default: The guest additions on this VM do not match the installed version of
default: VirtualBox! In most cases this is fine, but in rare cases it can
default: prevent things such as shared folders from working properly. If you see
default: shared folder errors, please make sure the guest additions within the
default: virtual machine match the version of VirtualBox you have installed on
default: your host and reload your VM.
default:
default: Guest Additions Version: 4.3.40
default: VirtualBox Version: 6.1
==> default: Mounting shared folders...
default: /vagrant => /home/didi/Documents/proyecto-edt/vagrant/parte_2/pruebal
```

Al hacer el `vagrant up` podemos ver que lo que **Vagrant** hace es importar el box que hemos especificado, es decir hace una copia y sobre esta copia hace una serie de adaptaciones para que podamos utilizar y acceder a la máquina, las adaptaciones que hace son:

- Conectarla en una red interna que proporciona VirtualBox y que tiene acceso a internet mediante **NAT**
- Abrir el puerto 2222 (en caso de que esté en uso, usa otro puerto) de la máquina anfitriona por tal de poder hacer un redirección al puerto 22 de la máquina virtual y que nosotros nos podamos conectarnos por ssh.
- Hace una serie de comprobaciones
- Monta un directorio compartido

Todos estos pasos se hacen automáticamente.

```
[didi@localhost pruebal]$ vagrant ssh
vagrant@127.0.0.1's password:
Welcome to Ubuntu 14.04.6 LTS (GNU/Linux 3.13.0-170-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

System information as of Wed May  5 15:09:32 UTC 2021

System load:  0.05               Processes:            80
Usage of /:   3.6% of 39.34GB    Users logged in:     0
Memory usage: 25%               IP address for eth0: 10.0.2.15
Swap usage:   0%

Graph this data and manage this system at:
  https://landscape.canonical.com/

UA Infrastructure Extended Security Maintenance (ESM) is not enabled.

0 updates can be installed immediately.
0 of these updates are security updates.

Enable UA Infrastructure ESM to receive 64 additional security updates.
See https://ubuntu.com/advantage or run: sudo ua status

New release '16.04.7 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

vagrant@vagrant-ubuntu-trusty-64:~$
```

Y, como podemos ver, al hacer un simple `vagrant ssh` nos conectamos a la máquina lanzada y esta, por medio de NAT, puede acceder al exterior y la máquina anfitriona puede contactarse con ella mediante comandos de vagrant sin ningún problema, esto lo que resuelve es tener que hacer el tedioso proceso de levantar una máquina, instalar el sistema operativo, configurarlo, etc.

Una vez tengamos la máquina encendida podemos hacer un `vagrant status` para poder ver el estado de esta, este comando también nos proporciona información sobre qué comandos podemos utilizar para parar, suspender o reiniciar la máquina.

```
[didi@localhost pruebal]$ vagrant status
Current machine states:

default                  running (virtualbox)

The VM is running. To stop this VM, you can run `vagrant halt` to
shut it down forcefully, or you can run `vagrant suspend` to simply
suspend the virtual machine. In either case, to restart it again,
simply run `vagrant up`.
```

En caso de que ya no necesitemos este escenario, lo que podemos hacer es `vagrant destroy`, que lo que hace es destruir la configuración que hayamos hecho en la máquina una vez iniciada.

```
[didi@localhost pruebal]$ vagrant destroy
default: Are you sure you want to destroy the 'default' VM? [y/N] y
==> default: Forcing shutdown of VM...
==> default: Destroying VM and associated drives...
```

3.3. Directorio Compartido

La gran mayoría de los boxes de **Vagrant** suelen crearse con un directorio compartido, es decir, un directorio que está sincronizado entre la máquina anfitriona y la máquina virtual, este directorio se suele encontrar en la ruta `/vagrant` de las máquinas virtuales y en el directorio dónde se encuentra el **Vagrantfile** en caso de la máquina anfitriona.

Por lo tanto, lo que permite este directorio es compartir de un extremo a otro, todo tipo de archivos.

```
==> default: Mounting shared folders...
default: /vagrant => /home/didi/Documents/proyecto-edt/vagrant/parte_2/pruebal
```

Al momento de compartir los archivos, **Vagrant** lo que hace es mapear el usuario y grupo activo del origen al usuario y grupo activo del destino.

<pre>[didi@localhost pruebal]\$ ls -l > tranferido.txt [didi@localhost pruebal]\$ ls -lh total 8.0K -rw-rw-r-- 1 didi didi 113 May 5 17:44 tranferido.txt -rw-rw-r-- 1 didi didi 75 May 5 17:02 Vagrantfile</pre>	<pre>vagrant@vagrant-ubuntu-trusty-64:/vagrant\$ ls -lh total 8.0K -rw-rw-r-- 1 vagrant vagrant 113 May 5 15:44 tranferido.txt -rw-rw-r-- 1 vagrant vagrant 75 May 5 15:02 Vagrantfile vagrant@vagrant-ubuntu-trusty-64:/vagrant\$</pre>
---	---

3.4. Reempaquetar un box

Al momento de descargar un box, **Vagrant** lo que en verdad hace es, en cuanto descarga el `.box`, lo descomprime y lo guarda en `~/vagrant.d/boxes` ya descomprimido que contiene:

- Un fichero `.ovf` que es de información
- La imagen
- Un fichero de metadatos
- Un **Vagrantfile** que hace la configuración básica para que la máquina pueda arrancar

Por lo tanto en verdad nosotros no tenemos el `.box`, ahora bien, digamos que nosotros necesitamos el `.box` para poder transportarlo a una máquina que no tiene acceso a internet y, por lo tanto no, puede descargar de la página de [Vagrant Cloud](#) los boxes.

Vagrant tiene una opción llamada **repackage** que lo que hace es, a partir de los elementos del box, volver a crear el `.box`, entonces con esto podemos llevar este `.box` a cualquier lugar.

```
[didi@localhost ~]$ vagrant box list | grep ubuntu
ubuntu/trusty64 (virtualbox, 20190514.0.0)
[didi@localhost ~]$ vagrant box repack ubuntu/trusty64 virtualbox 20190514.0.0
[didi@localhost ~]$ ls *.box
package.box
```

Como podemos ver, para poder hacer el repackaging necesitamos el nombre del box, el proveedor que utiliza y la versión que tiene.

3.5. Eliminación e instalación de una imagen local

En caso de que tengamos una imagen propia en local y queremos que **Vagrant** use esta en vez de alguna que esté en [Vagrant Cloud](#) lo que podemos hacer es indicarle el .box que queremos que use y especificarle un nombre:

```
[didi@localhost ~]$ vagrant box list
There are no installed boxes! Use `vagrant box add` to add some.
[didi@localhost ~]$ ls *.box
package.box
[didi@localhost ~]$ vagrant box add package.box --name diego-ubuntu/trusty64
==> box: Box file was not detected as metadata. Adding it directly...
==> box: Adding box 'diego-ubuntu/trusty64' (v0) for provider:
       box: Unpacking necessary files from: file:///home/didi/package.box
==> box: Successfully added box 'diego-ubuntu/trusty64' (v0) for 'virtualbox'!
[didi@localhost ~]$ vagrant box list
diego-ubuntu/trusty64 (virtualbox, 0)
```

3.6. Actualización de una imagen

Vagrant a través de [Vagrant Cloud](#) nos proporciona boxes los cuales con el tiempo se van actualizando.

Pero, ¿cómo podemos ver si el box que tenemos instalado necesita una actualización?, **Vagrant** proporciona una opción `outdated` la cual, con el argumento `--global` hace un chequeo de todos los boxes que tenemos instalados y nos dice si necesitan actualizarse o no.

```
[didi@localhost ~]$ vagrant box list
debian/jessie64          (virtualbox, 8.11.0)
diego-ubuntu/trusty64   (virtualbox, 0)
ubuntu/trusty64         (virtualbox, 20190514.0.0)
[didi@localhost ~]$ vagrant box outdated --global
* 'ubuntu/trusty64' for 'virtualbox' (v20190514.0.0) is up to date
* 'diego-ubuntu/trusty64' for 'virtualbox' wasn't added from a catalog, no version information
* 'debian/jessie64' for 'virtualbox' is outdated! Current: 8.11.0. Latest: 8.11.1
```

Como podemos ver hay el box "ubuntu/trusty64" está actualizado, detecta que el box "diego-ubuntu/trusty64" no pertenece al catálogo de la comunidad de **Vagrant** y no tiene versión y por lo tanto no hay información, y, por último, detecta que hay una versión más nueva para el box "debian/jessie64" y por lo tanto necesita actualizarse.

Para actualizar un box lo que tenemos que hacer es crear un `vagrant init` con lo cual nos creará un fichero **Vagrantfile**, tenemos que hacer un `vagrant up --provider virtualbox` y por último tenemos que hacer un `vagrant box update` y entonces **Vagrant** se conectará a [Vagrant](#)

[Cloud](#), comprobará la versión descargada con la versión más nueva de la página, la descargará y la instalará.

```
[didi@localhost actualizar]$ vagrant box update
==> default: Checking for updates to 'debian/jessie64'
      default: Latest installed version: 8.11.0
      default: Version constraints:
      default: Provider: virtualbox
==> default: Updating 'debian/jessie64' with provider 'virtualbox' from version
==> default: '8.11.0' to '8.11.1'...
==> default: Loading metadata for box 'https://vagrantcloud.com/debian/jessie64'
==> default: Adding box 'debian/jessie64' (v8.11.1) for provider: virtualbox
      default: Downloading: https://vagrantcloud.com/debian/boxes/jessie64/versions/8.11.1/providers
/virtualbox.box
Download redirected to host: vagrantcloud-files-production.s3-accelerate.amazonaws.com
==> default: Successfully added box 'debian/jessie64' (v8.11.1) for 'virtualbox'!
```

También podemos especificar el box que queremos actualizar con

vagrant box update --box [boxname] , esto nos permite actualizar el box desde cualquier lugar y sin necesidad de hacer un vagrant init y vagrant up :

```
[didi@localhost actualizar]$ vagrant box outdated --global
* 'ubuntu/trusty64' for 'virtualbox' (v20190514.0.0) is up to date
* 'diego-ubuntu/trusty64' for 'virtualbox' wasn't added from a catalog, no version information
* 'debian/jessie64' for 'virtualbox' is outdated! Current: 8.11.0. Latest: 8.11.1
[didi@localhost actualizar]$ vagrant box update --box debian/jessie64
Checking for updates to 'debian/jessie64'
Latest installed version: 8.11.0
Version constraints: > 8.11.0
Provider: virtualbox
Updating 'debian/jessie64' with provider 'virtualbox' from version
'8.11.0' to '8.11.1'...
Loading metadata for box 'https://vagrantcloud.com/debian/jessie64'
Adding box 'debian/jessie64' (v8.11.1) for provider: virtualbox
Downloading: https://vagrantcloud.com/debian/boxes/jessie64/versions/8.11.1/providers/virtualbox.b
ox
Download redirected to host: vagrantcloud-files-production.s3-accelerate.amazonaws.com
Successfully added box 'debian/jessie64' (v8.11.1) for 'virtualbox'!
```

Vagrant no borrará la versión antigua por si se necesita volver a usar por cualquier motivo, pero este box ocupa espacio, entonces lo que podemos hacer es un `vagrant prune` para que borre las versiones antiguas y deje solo la más actual.

```
[didi@localhost actualizar]$ vagrant box outdated --global
* 'ubuntu/trusty64' for 'virtualbox' (v20190514.0.0) is up to date
* 'diego-ubuntu/trusty64' for 'virtualbox' wasn't added from a catalog, no version information
* 'debian/jessie64' for 'virtualbox' (v8.11.1) is up to date
* 'debian/jessie64' for 'virtualbox' is outdated! Current: 8.11.0. Latest: 8.11.1
[didi@localhost actualizar]$ vagrant box prune
The following boxes will be kept...
debian/jessie64      (virtualbox, 8.11.1)
diego-ubuntu/trusty64 (virtualbox, 0)
ubuntu/trusty64      (virtualbox, 20190514.0.0)

Checking for older boxes...
Removing box 'debian/jessie64' (v8.11.0) with provider 'virtualbox'...
[didi@localhost actualizar]$ vagrant box outdated --global
* 'ubuntu/trusty64' for 'virtualbox' (v20190514.0.0) is up to date
* 'diego-ubuntu/trusty64' for 'virtualbox' wasn't added from a catalog, no version information
* 'debian/jessie64' for 'virtualbox' (v8.11.1) is up to date
```

3. Configuración Vagrantfile

4.1. Configuración Simple

Hasta ahora solo hemos usado el **Vagrantfile** para indicarle qué box tiene que levantar, pero tenemos muchas más opciones como por ejemplo indicarle un nombre de hostname, la memoria que puede usar, los cpus que puede usar, etc que suelen ser los primeros pasos al crear una máquina virtual.

Normalmente cada proveedor tiene sus valores determinados para estos casos y no hace falta configurar nada, pero en caso de queramos editar los valores podemos hacerlo.

En el caso de **virtualbox**, que es el proveedor con el que estamos trabajando, tiene un listado en la página de [Vagrant](#) en la cual nos especifica que opciones podemos usar para configurar nuestro box.

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
  config.vm.hostname = "diego-ubuntu"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = "2"
  end
end
```

Como podemos ver en el **Vagrantfile**, le hemos especificado que al momento de levantar la máquina, tenga el hostname: "diego-ubuntu" (que de esta parte se encargan las opciones de configuración de virtual machine de **Vagrant**) y también le hemos especificado al proveedor, que en este caso es **VirtualBox** que de memoria tenga 1024MB y que trabaje con 2 cpus.

Al momento de hacer `vagrant up` podemos ver que se han realizado los cambios que hemos especificado en el fichero **Vagrantfile**:

```
vagrant@diego-ubuntu:~$ hostname
diego-ubuntu
vagrant@diego-ubuntu:~$ lscpu | grep Core
Core(s) per socket:    2
vagrant@diego-ubuntu:~$ free -h
              total        used        free      shared    buffers      cached
Mem:           993M        378M        615M         364K          12M         245M
-/+ buffers/cache:        120M        873M
Swap:           0B           0B           0B
```

En caso de hacer otros cambios más específicos se puede usar el comando `vagrant reload` para no tener que hacer un `vagrant destroy` y un `vagrant up` cada vez que queramos aplicar los cambios que hayamos hecho en el Vagrantfile.

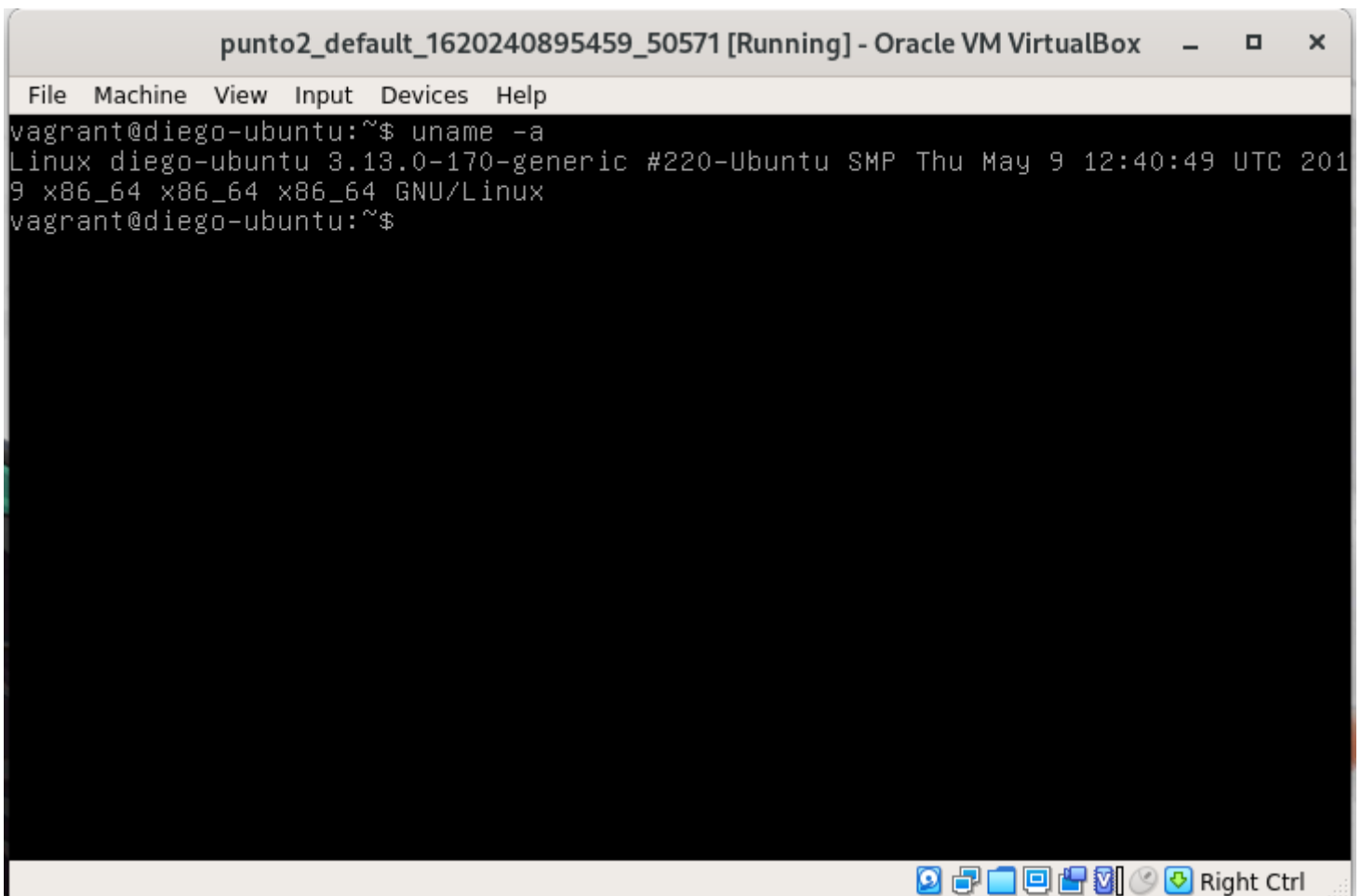
4.2. Interfaz Gráfica

Vagrant también nos da la posibilidad de usar una interfaz gráfica que puede ser muy útil en algunos casos, por ejemplo, supongamos que estamos haciendo configuraciones con ssh y hacemos algo mal que no nos deja conectarnos por ssh a la máquina virtual, entonces no tenemos otra forma de conectarnos, aquí es cuando entra la interfaz gráfica que, siempre que tengamos al menos un usuario creado, podemos entrar sin problema.

Para hacer que la máquina, al levantarse, ejecute una interficie gráfica es muy sencillo:

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
  config.vm.hostname = "diego-ubuntu"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = "2"
    vb.gui = true
  end
end
```

y al hacer un `vagrant up` nos abre la interfaz gráfica:



```
punto2_default_1620240895459_50571 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
vagrant@diego-ubuntu:~$ uname -a
Linux diego-ubuntu 3.13.0-170-generic #220-Ubuntu SMP Thu May 9 12:40:49 UTC 201
9 x86_64 x86_64 x86_64 GNU/Linux
vagrant@diego-ubuntu:~$
```

Esto funcionará siempre y cuando el proveedor que se esté utilizando soporte la interfaz gráfica

4.3. Aprovisionamiento Ligero

Imaginemos que tenemos 10 máquinas lanzadas y todas usan la misma imagen, lo que pasará es que por cada máquina creada habrá una imagen ocupando espacio en disco real del host anfitrión.

El aprovisionamiento ligero o *"thin provisioning"* consiste en que en lugar de clonar el disco que contiene la imagen de la máquina para cada una de ellas lo que hace es crear un fichero de imagen que almacena únicamente las diferencias que tiene respecto a la imagen inicial, por lo cual creamos un fichero de imagen mucho más pequeño y ya este fichero irá creciendo en el futuro conforme hayan mas diferencias.

Por lo tanto lo que nos permite este método es ahorrar mucho espacio en el disco real.

Este recurso depende de cada proveedor que lo soporte, en **VirtualBox** la opción que nos permite usar aprovisionamiento ligero es [linked_clone](#)

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
  config.vm.hostname = "diego-ubuntu"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = "2"
    vb.linked_clone = true
  end
end
```

A hacer `vagrant up` podemos ver que se prepara para hacer el aprovisionamiento ligero:

```
[didi@localhost punto3]$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Preparing master VM for linked clones...
    default: This is a one time operation. Once the master VM is prepared,
    default: it will be used as a base for linked clones, making the creation
    default: of new VMs take milliseconds on a modern system.
```

Al inspeccionar lo que se crea al arrancar la máquina podemos ver que en lugar de crear una imagen de nuevo lo que ha hecho es crear un directorio de Snapshot donde guarda el fichero de la imagen que unicamente contiene las diferencias con el fichero de imagen original:

```
[didi@localhost VirtualBox VMs]$ ll punto3_default_1620243095023_28156/Snapshots/
total 83500
-rw----- 1 didi didi 90439680 May  5 21:34 {5fa47273-6747-41a0-9427-b799e843532d}.vmdk
[didi@localhost VirtualBox VMs]$ ll -h punto2_default_1620240895459_50571/
total 1.6G
-rw----- 1 didi didi 1.6G May  5 20:59 box-disk1.vmdk
drwx----- 1 didi didi  16 May  5 20:54 Logs
-rw----- 1 didi didi 3.6K May  5 20:59 punto2_default_1620240895459_50571.vbox
-rw----- 1 didi didi 3.0K May  5 20:59 punto2_default_1620240895459_50571.vbox-prev
```

4.4. Redirección de puertos

Vagrant por defecto ya hace redireccionamiento:

- En un principio todas la máquinas virtuales están conectadas a una red interna de **VirtualBox**, a partir de aquí se le asigna a la máquina virtual una dirección ip dentro del segmento de la red privada y se le pone como gateway una dirección ip que VBox conecta al exterior y para que esta máquina virtual pueda acceder al exterior se hace un proceso de **Source NAT**.
- Por otra parte, si queremos conectarnos desde el exterior (teniendo en cuenta también el host anfitrión) a la máquina virtual lo que hace **Vagrant** por defecto es redirigir por el puerto 2222 (en caso que esté en uso se usa otro puerto) de la máquina anfitriona al puerto 22 de la máquina virtual.

Pero si nosotros queremos redirigir manualmente un puerto para poder acceder a otro servicio, a apache por ejemplo, ¿Cómo se haría?

Para ello vagrant tiene la opción **vm.network** que, entre cosas, nos permite hacer el port forwarding:

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
  config.vm.hostname = "diego-ubuntu"
  config.vm.network "forwarded_port", guest: 80, host: 8080
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = "2"
  end
end
```

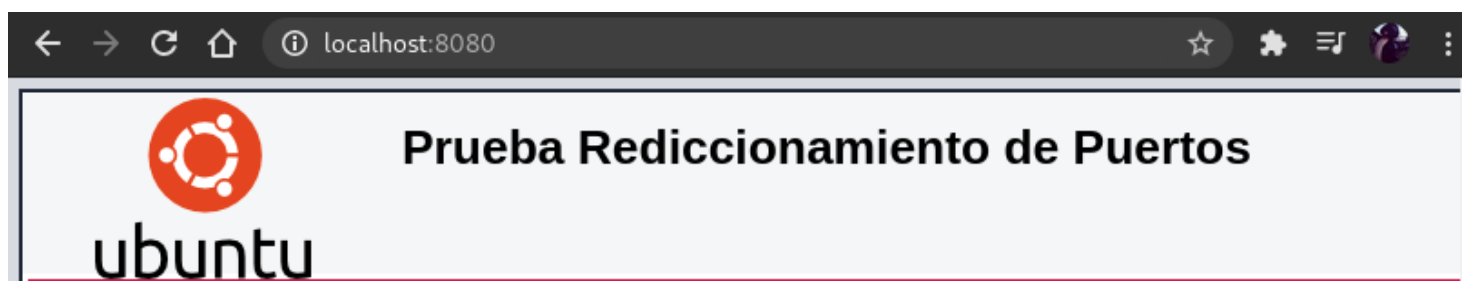
Aquí le indicamos que desde el puerto 8080 del host anfitrión (**host**) se redirigirá al puerto 80 de la máquina virtual (**guest**).

Lo que queda hacer es un `vagrant up` :

```
==> default: Forwarding ports...
default: 80 (guest) => 8080 (host) (adapter 1)
default: 22 (guest) => 2222 (host) (adapter 1)
```

Aquí podemos ver como, en los pasos que hace vagrant para preparar la máquina, hace el redireccionamiento de puertos.

Para poder demostrar el funcionamiento he instalado apache2 para ubuntu y he editado el **index.html** para que aparezca "Prueba Redireccionamiento de Puertos":



Un comando muy útil para saber que puertos han sido redireccionados es: `vagrant port` :

```
[didi@localhost punto4]$ vagrant port
The forwarded ports for the machine are listed below. Please note that
these values may differ from values configured in the Vagrantfile if the
provider supports automatic port collision detection and resolution.

22 (guest) => 2222 (host)
80 (guest) => 8080 (host)
```

4.5. Configuración red privada

En este apartado vamos a añadir una red privada, a parte de la red que **Vagrant** utiliza por defecto para todas la máquinas, la cual va a estar conectada a la red privada donde se encuentra la máquina anfitriona y por lo tanto también podrán tener conexión la máquina anfitriona con la máquina virtual.

Para poder añadir esta nueva ip tenemos que hacer uso de la opción `vm.network` que proporciona **Vagrant**:

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/xenial64"
  config.vm.network "private_network", ip: "192.168.100.50"
end
```

Como podemos ver en este caso he añadido la ip "192.168.100.50".

```
==> default: Preparing network interfaces based on configuration...
default: Adapter 1: nat
default: Adapter 2: hostonly
```

Aquí podemos comprobar como **Vagrant**, a parte de la red por defecto que permite acceder al exterior, añade otra red que solo sirve par el entorno privado.

```
vagrant@ubuntu-xenial:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 02:ed:98:3c:8e:82 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global enp0s3
        valid_lft forever preferred_lft forever
    inet6 fe80::ed:98ff:fe3c:8e82/64 scope link
        valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:b9:d0:07 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.50/24 brd 192.168.100.255 scope global enp0s8
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feb9:d007/64 scope link
        valid_lft forever preferred_lft forever
```

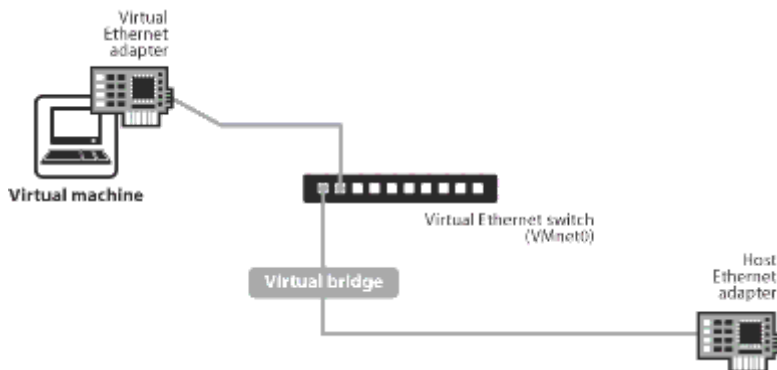
Y al acceder a la máquina con `vagrant ssh` y hacer un `ip a` podemos comprobar que ahora tiene 3 interfaces: la de loopback, la que permite acceder al exterior por medio de NAT y la que está en el entorno privado.

4.6. Configuración red pública

La configuración de una red pública es bastante parecida a la privada, lo que cambia es que ahora le especificamos que tiene que hacer una conexión de modo **bridge**.

Lo que hace el modo **bridge** es replicar permitir a la máquina virtual acceder a la red física, es decir la red en la que está el router que permite el acceso al exterior, **VirtualBox** consigue hacer esto mediante un driver en la máquina anfitriona que filtra los datos que van hacia la máquina virtual por medio de la interficie de red de la máquina anfitriona, este driver es llamado **net filter**.

Por lo tanto así hace la simulación de que es otro host que se encuentra en la red del router y este le proporciona una IP y acceso al exterior por medio de SNAT.



```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/xenial64"
  config.vm.network "public_network", bridge: "enp0s31f6"
  config.vm.provider "virtualbox" do |vb|
    end
  end
end
```

```
==> default: Preparing network interfaces based on configuration...
default: Adapter 1: nat
default: Adapter 2: bridged
```

Podemos ver que ahora se ha añadido otra interficie, pero está vez en modo "puente".

```
vagrant@ubuntu-xenial:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 02:ed:98:3c:8e:82 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global enp0s3
        valid_lft forever preferred_lft forever
    inet6 fe80::ed:98ff:fe3c:8e82/64 scope link
        valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:b9:2c:b3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.24/24 brd 192.168.0.255 scope global enp0s8
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feb9:2cb3/64 scope link
        valid_lft forever preferred_lft forever
```

También podemos ver en esta imagen que tenemos una ip que nos ha proporcionado el servidor DHCP del router.

4.7. Multimáquinas

El uso de multimáquinas puede ser muy útil para algunos casos, por ejemplo para hacer una simulación de un entorno de producción que contiene múltiple máquinas.

Para poder lanzar multimáquinas en el fichero de configuración de **Vagrantfile** tenemos que usar la opción que nos proporciona **Vagrant** `vm.define` que sirve para indicar la máquinas que queremos levantar.

```
Vagrant.configure("2") do |config|
  config.vm.define "web" do |maquina_1|
    maquina_1.vm.box = "ubuntu/xenial64"
    maquina_1.vm.hostname = "web-server"
    maquina_1.vm.network "public_network", bridge: "eth0"
    maquina_1.vm.network "private_network", ip: "10.200.100.101"
  end
  config.vm.define "db" do |maquina_2|
    maquina_2.vm.box = "ubuntu/xenial"
    maquina_2.vm.hostname = "db-server"
    maquina_2.vm.network "private_network", ip: "10.200.100.102"
  end
end
```

Hemos hecho la simulación de que tenemos dos servidores, uno es el servidor web que tiene acceso a internet y también tiene una ip en la red privada "10.200.100.101" y el otro es un servidor de base de datos que no tiene acceso a internet y tiene la ip en la red privada "10.200.100.102" para que sea posible la conexión entres estos dos servidores.

```
[didi@localhost punto7]$ vagrant up
Bringing machine 'web' up with 'virtualbox' provider...
Bringing machine 'db' up with 'virtualbox' provider...
[didi@localhost punto7]$ vagrant status
Current machine states:

web                running (virtualbox)
db                 running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

5. Ejercicio Final

Para este ejercicio final vamos a levantar una máquina con las siguientes características:

- Usará un box de [ubuntu/xenial64](#)

- Usará como proveedor **VirtualBox**
 - Tendrá 2GB de memoria asignados
 - Tendrá 2 cores virtuales asignados
- Tendrá, a parte de la ip por defecto, una ip en la red privada "192.68.33.0/24"
- A parte de directorio compartido **/vagrant**, tendrá otro en **/var/www/html** con los permisos 777 para directorios y 666 para ficheros
- Usará un recurso de provisión para ejecutar comandos de shell, los cuales son transferidos por un fichero **.sh**

Este entorno está preparado para hacer pruebas en apache2, que a su vez cuenta con módulos de php, y también para hacer test en ldap.

6. Conclusiones

Con **Vagrant** me he encontrado que es muy fácil de utilizar si se tiene un conocimiento básico de la programación ya que usa el lenguaje **Ruby** que es de alto nivel y muy fácil de entender.

También me ha sido de mucha ayuda para hacer las pruebas para los otros temas lanzando máquinas virtuales con diferentes sistemas operativos e instalando y configurando software en estas para poder asegurarme que todo vaya bien antes de hacer el despliegue.

7. Bibliografía

<https://www.vagrantup.com/docs>

<https://phoenixnap.com/kb/vagrant-beginner-tutorial>

<https://medium.com/swlh/hashicorp-vagrant-101-6f7e613d8af>

<https://www.vagrantbox.es/>

<https://openwebinars.net/>

<https://github.com/hashicorp/vagrant>

<https://ashki23.github.io/vagrant.html>

https://www.virtualbox.org/manual/ch06.html#network_bridged