



Máquina virtual para proceso de datos (*Python, R, Spark*)

2021-01-13

Con el fin de facilitar prácticas en procesos de BigData, entre ellas las correspondientes a Spark, se proporciona una máquina virtual (VM) con software preinstalado:

- Python 3.6 con paquetes de procesamiento de datos: **NumPy**, **SciPy**, **Matplotlib**, **Pandas**, **Scikit-Learn**, etc
- R 3.6.2 con paquetes adicionales (**tidyverse**, **ggplot2**, **caret**, etc)
- Spark 2.4, con el software necesario para conectarlo a Python y R
- IPython Notebook para el desarrollo de *notebooks* de proceso, con kernels disponibles en 4 lenguajes:
 - Python 3.6
 - Pyspark (Python 3.6 + Spark)¹
 - Scala 2.11 + Spark
 - R, con posibilidad de conectar a Spark mediante **SparkR** y **sparklyr**
- Opcionalmente, **RStudio Server 1.3**

La máquina se ha provisionado con *Vagrant* usando *VirtualBox* como proveedor de entorno virtual, y está pre-empaquetada de forma que su despliegue es rápido, una vez descargada la *box* base, que sí que lleva tiempo por su tamaño (alrededor de 2 GB).

¹ Si sólo se quiere usar Python sin Spark, es conveniente usar el kernel Python simple *Python 3*, que usa menos memoria que el kernel *Pyspark* (ya que este último arranca siempre un driver de Spark, aunque luego no se use).

1. Prerrequisitos

- Hardware: un ordenador con arquitectura de 64 bits, suficiente memoria RAM (al menos 2 GB libres, cuanto más mejor) y unos 10GB de espacio libre en disco.
- Sistema operativo: Windows 7 o superior de 64 bits, Linux 64 bits (probado con Ubuntu y CentOS, seguramente funcione con otros) o Max OS X.
- El procesador debe soportar virtualización (en general los procesadores modernos lo soportan, aunque a veces viene desactivado; en el proceso de instalación de la sección 2.2 se explica cómo comprobarlo).
- Software: se requiere tener instalado lo siguiente:
 - [VirtualBox](#) 5.0 o superior
 - Una versión reciente de [Vagrant](#). Se recomienda una igual o mayor que 2.0.1 (si se usa Virtualbox 6.x, será necesario Vagrant 2.2.7 o superior)

Nota: Vagrant descarga la box base en el directorio dado por `VAGRANT_HOME` (por defecto `~/.vagrant.d`), y VirtualBox crea por defecto los ficheros de máquinas virtuales en `~/VirtualBox Vms`

Es decir, por defecto ambas son carpetas situadas bajo la carpeta principal del usuario. Por lo tanto la carpeta principal del usuario (`$HOME`, o en Windows `%UserProfile%`) es la que tiene que estar en un disco con espacio suficiente.

Si es necesario pueden cambiarse estos directorios:

- *La carpeta de Vagrant, definiendo la variable de entorno `VAGRANT_HOME` para que apunte al sitio deseado*
- *La carpeta de VMs de VirtualBox, ejecutando `VBoxManage setproperty machinefolder <directorio>`, o cambiando la opción en la interfaz gráfica de VirtualBox (menú Archivo > Preferencias > General)*

2. Instalación

2.1 Fichero de configuración

Para su instalación se requiere el fichero **Vagrantfile**, que especifica los detalles y configuración de la máquina virtual de **Vagrant**. Este fichero está empaquetado en un ZIP que contiene además algunas carpetas de apoyo, y que se puede descargar de la siguiente dirección:

uam-2.5.1.zip	Vagrantfile para arquitectura de 64 bits
-------------------------------	---

Este fichero de especificación **Vagrant** permite la instalación en cualquier máquina conectada a Internet, ya que descarga la *box* de **Vagrant** desde la nube pública de [Vagrant Cloud](#).

2.2 Variables de configuración

El **Vagrantfile** es un fichero de texto que define y controla la creación, provisionado y arranque de la máquina virtual (VM).

La configuración y funcionamiento de la VM es controlable parcialmente mediante variables incluidas en el **Vagrantfile**. Estas variables están definidas al principio del fichero, y son de dos tipos:

- **Variables de ejecución:** leídas y procesadas al arrancar la VM, por lo tanto para que actúen basta modificar el fichero y rearrancar la máquina (ver sección 3.1 *Gestión*)
- **Variables de creación:** leídas y procesadas en el momento de creación y configuración de la VM. Una vez creada la VM para que actúen es necesario destruirla y regenerarla.

Entre las primeras (variables de ejecución) están:

- la cantidad de memoria RAM asignada a la VM
- el número de procesadores (cores) asignados a la VM
- el puerto que usa **Jupyter** para publicar la interfaz de *Notebooks*

Entre las segundas (variables de creación):

- La contraseña usada en **Jupyter** para acceder a los *Notebooks*

2.3 Proceso de instalación

1. Asegurarse de que el procesador del PC donde se va a instalar soporta virtualización. Por ejemplo, para procesadores Intel puede consultarse la [página de ayuda sobre virtualización](#). En caso de que la virtualización esté desactivada, hay que activarla en [la BIOS del ordenador](#).
2. Usando el enlace anterior descargar el fichero ZIP **en una máquina con arquitectura de 64 bits**, crear una carpeta local, y descomprimir el ZIP en ella (manteniendo la estructura de directorios del ZIP).

Nota: es preferible, especialmente en Windows, que el nombre completo del directorio (la ruta completa desde C:\) no contenga espacios ni caracteres no ASCII (i.e. sin acentos o ñes), ya que pueden causar problemas.

3. Si se desea modificar la configuración de la máquina virtual, abrir el fichero **Vagrantfile** con un editor de texto y configurar las opciones deseadas, situadas todas al comienzo del fichero (ver sección 2.2). *Nota: esto es opcional, no es necesario realizar cambios para tener una máquina virtual funcional.* Opciones potencialmente interesantes son la cantidad de memoria o CPUs asignadas a la máquina virtual, o la contraseña de acceso a *notebooks*.
4. Arrancar una consola del sistema² (**cmd** en Windows, una shell en Linux), cambiar (**cd**) al directorio base que haya creado el ZIP al descomprimirse (**m1-vm-notebook-uam-2.5**), y ejecutar **vagrant up**
5. **Vagrant** comenzará el proceso y se descargará la *box* del repositorio (un proceso que puede tardar, en función de la conexión de red). Este proceso sólo se realiza una vez.
6. A continuación arrancará y configurará la máquina virtual. Durante esta operación va escribiendo mensajes a pantalla con el progreso alcanzado.
7. Una vez terminado, aparece el mensaje final, invitando a conectarse a la interfaz de *notebooks*, que estará disponible en <http://localhost:8008> . Para entrar se necesita una contraseña, que por defecto es **vmuser**

2.4 Problemas durante la instalación

Algunas situaciones conflictivas se producen por problemas con el **Vagrantfile**:

- *Versión de vagrant no compatible*: debería ser **Vagrant** versión 2.0.1 o superior (ejecutando **vagrant -v** se puede saber la versión que tenemos)
- *Vagrantfile no se encuentra*: es necesario ejecutar **vagrant up** en una sesión de consola que esté situada en el mismo directorio que el **Vagrantfile** (más exactamente, en ese directorio o en un subdirectorio suyo)
- *Vagrantfile corrompido*: es importante no ejecutar **vagrant init** una vez descargado el **Vagrantfile**, ya que esa orden inicializa el entorno creando un **Vagrantfile** nuevo a partir de una plantilla, y por tanto destrozando el que habíamos descargado

Otros posibles problemas de instalación pueden deberse a características particulares del ordenador anfitrión donde se instala la máquina virtual. En caso de dificultades, puede consultarse el apéndice C. *Depuración del proceso de instalación y arranque*, que detalla diferentes posibles contingencias y cómo resolverlas.

² Los que no estén familiarizados con la operación de la interfaz de consola de Windows pueden consultar el apéndice B. Interfaz de consola en Windows para una mini-introducción.

3. Operación de la VM

3.1 Gestión

Una vez arrancada, el servidor de *Notebooks* estará disponible en la dirección <http://localhost:8008>. La contraseña de acceso (a no ser que se haya cambiado al instalar) es `vmuser`. La instalación contiene unos pequeños *notebooks* de ejemplo. En el menú de la derecha se pueden crear *Notebooks* con los kernels instalados.

Como se mencionó antes, el kernel *Python 3* no está conectado a Spark; para desarrollar aplicaciones en Pyspark hay que usar el kernel *Pyspark*, que sí arranca un driver de Spark (local) en la máquina virtual (obviamente el kernel *Pyspark* exige más recursos que el *Python 3* base³)

La VM se puede controlar mediante el comando `vagrant <operacion>`. Esta orden actúa sobre la máquina virtual cuyo `Vagrantfile` está situado en el directorio actual. Los comandos más importantes son:

- `vagrant up` (ya visto, para arrancar la VM)
- `vagrant halt` (para pararla). Si ya no vamos a usar la VM durante un rato, es conveniente pararla para que deje de ocupar recursos del PC (lo mismo si vamos a apagar el ordenador, para que se cierre de forma limpia).
- `vagrant reload` (ciclo completo parada-arranque). Útil cuando cambiamos algún parámetro del `Vagrantfile` para recargar la máquina con la nueva configuración.
- `vagrant destroy` (para eliminar completamente la máquina virtual, limpiando sus ficheros)
- `vagrant ssh` (para abrir una sesión de consola dentro de ella, ver siguiente apartado).

La máquina virtual **no arranca automáticamente** con el ordenador, por tanto si se reinicia el ordenador anfitrión será necesario volver a ejecutar `vagrant up` para arrancarla de nuevo. Estos procesos de arranque solo tardan unos segundos, porque la VM ya está instalada y configurada. Asimismo, como se ha mencionado al terminar de usarla conviene pararla (`vagrant halt`), para que no consuma recursos del sistema mientras usamos el ordenador para otras tareas.

La estructura del paquete descargado es la siguiente:

<code>./Vagrantfile</code>	fichero de construcción de la máquina virtual
<code>./vmfiles/</code>	subcarpeta local que la máquina virtual monta internamente (aparece como <code>/vagrant/</code> dentro de ella)
<code>./vmfiles/IPNB</code>	subcarpeta local que la máquina virtual usa como directorio de <i>notebooks</i>
<code>./vmfiles/R</code>	subcarpeta local pensada para que la máquina virtual la use en RStudio como directorio de trabajo (ver más abajo)

Por tanto para pasar un fichero a la máquina virtual lo más sencillo es dejarlo en `./vmfiles`. Y para

³ Si se abre desde *Jupyter* un notebook *pyspark* (fichero `*.ipynb`) que ha sido creado en otro sistema, puede darse el caso que de *Jupyter* no lo interprete bien e intente procesarlo usando el kernel Python en vez de Pyspark; en ese caso no estará disponible *Spark* y los intentos de usar las variables predefinidas `sc` o `spark` darán error. En ese caso puede cambiarse el kernel manualmente (menú *Kernel* → *change kernel*) para asignar el correcto.

instalar un *notebook* de forma que aparezca inmediatamente en la interfaz, se deja en `./vmfiles/IPNB` (en el fichero ZIP original, esa carpeta contiene los *notebooks* de ejemplo).

Es importante tener en cuenta esta circunstancia a la hora de escribir notebooks: los nombres de fichero que tenemos que usar en el código que hay dentro de un notebook (si necesitamos abrir un fichero de datos, por ejemplo) no son los que tendría en el host, sino los de la máquina virtual, puesto que el notebook se está ejecutando dentro de la máquina virtual. Por tanto un fichero de datos dejado en `./vmfiles/xxx` será visible en el notebook como `/vagrant/xxx`

Lo más sencillo es situarlos en la misma carpeta que los notebooks (o en una subcarpeta) y en el código del notebook usar nombres relativos. Así funcionan de forma fiable.

3.2 Interacción

El modo de arranque de la máquina es sin interfaz gráfica (*headless*). En este modo la VM no tiene una ventana gráfica asociada de trabajo; la interacción con la VM se realiza mediante tres accesos:

1. La interfaz de notebooks, publicada en la máquina anfitrión en <http://localhost:8008> por defecto (es decir, basta arrancar un navegador en la máquina anfitrión y usar esa dirección para entrar en *Jupyter*).

La contraseña de acceso (a no ser que se haya cambiado al instalar) es `vmuser`. La instalación contiene unos pequeños *notebooks* de ejemplo. En el menú de la derecha se pueden crear *Notebooks* con los kernels instalados.

Como se mencionó antes, el kernel *Python 3* no está conectado a Spark; para desarrollar aplicaciones en Pyspark hay que usar el kernel *Pyspark*, que sí arranca un driver de Spark (local) en la máquina virtual (obviamente el kernel *Pyspark* exige más recursos que el *Python 3* base⁴)

2. Sesiones de consola, creadas mediante `vagrant ssh` (o también dentro de la interfaz de *notebooks*). Ver sección 3.3 *Sesión de consola*
3. Ficheros compartidos: la subcarpeta `vmfiles` de la máquina anfitrión está montada automáticamente dentro de la VM (como `/vagrant`), por tanto todo lo que hagamos en la máquina anfitrión en esa carpeta se reflejará en la VM.

Eliminar la necesidad de una ventana dedicada alivia la carga de proceso en la máquina anfitrión. Para editar ficheros que no estén en una carpeta compartida hay tres editores de consola instalados dentro de la máquina virtual: `nano`, `vim` y `emacs`, que se pueden ejecutar dentro de una sesión de consola.

3.3 Sesión de consola

Además de operando con *notebooks* o en *RStudio*, también se puede interactuar con la máquina

⁴ Si se abre desde *Jupyter* un notebook *pyspark* (fichero `*.ipynb`) que ha sido creado en otro sistema, puede darse el caso que de *Jupyter* no lo interprete bien e intente procesarlo usando el kernel Python en vez de Pyspark; en ese caso no estará disponible *Spark* y los intentos de usar las variables predefinidas `sc` o `spark` darán error. En ese caso puede cambiarse el kernel manualmente (menú *Kernel* → *change kernel*) para asignar el correcto.

virtual abriendo una sesión de consola en ella (lo que permite por ejemplo inspeccionar la máquina, cambiar su configuración, examinar ficheros de log o lanzar tareas Spark de consola vía **spark-submit**). Para ello hay al menos tres vías:

1. Ejecutando **vagrant ssh** en un terminal de consola, teniendo cuidado de situarse antes en la carpeta donde está situado el **Vagrantfile**, para que **Vagrant** sepa localizar la VM. Este método exige que haya definida una aplicación para SSH en el sistema (lo cual suele venir por defecto en Linux, pero no siempre en Windows).
2. Usando la interfaz gráfica del *Notebook* en el navegador y creando una ventana de terminal (opción *New* → *Terminal*)
3. Utilizando una aplicación SSH estándar (**ssh** en Linux, en Windows puede por ejemplo usarse **PuTTY**), y conectándose con los siguientes datos:
 - Host: 127.0.0.1 (localhost)
 - Port: 2222
 - User: **vagrant**
 - Password: **vagrant**

En los dos primeros casos el usuario con que se conecta a la plataforma es *vmuser* (el usuario apropiado para lanzar tareas Spark); en el tercero es *vagrant* (puede cambiarse a usuario *vmuser* ejecutando **sudo -u vmuser bash**). Ambos usuarios, *vmuser* y *vagrant*, tienen permisos para convertirse en usuario *root* (ejecutando **sudo bash**) y con ello realizar tareas de sistema.

3.4 Reconstrucción de la VM

La máquina virtual es una máquina estándar Linux (concretamente está ejecutando Ubuntu Linux 18.04) y puede gestionarse como tal, instalando o modificando componentes. Si debido a la manipulación en algún momento la máquina queda inestable, lo más eficaz es regenerarla. Para ello, ejecutando ...

```
vagrant destroy
vagrant up
```

.. se borrará la máquina y se reconstruirá de nuevo, desde cero, limpiando todos los posibles problemas. Por supuesto al hacerlo se perderá todo el contenido instalado dentro de ella; sin embargo los ficheros de usuario (*notebooks*) serán preservados, ya que están instalados en **./vmfiles/IPNB**, que es un directorio externo a la máquina virtual (montado en ella como una carpeta externa).

3.5 RStudio Server

La máquina virtual puede también instalar automáticamente el servidor de **Rstudio**, si se efectúa como provisionamiento adicional (ver sección siguiente). El servidor de **RStudio** exporta la funcionalidad sobre el puerto 8787 de la VM, por tanto para poder acceder a él será necesario modificar el **Vagrantfile** para exportar ese puerto. Esto quiere decir que el proceso completo de instalación y configuración de **RStudio** es:

1. provisionar su sección correspondiente:

```
vagrant provision --provision-with rstudio
```

2. cambiar el **Vagrantfile** para exportar el puerto 8787 de la VM. Para ello buscar una línea que diga:

```
#vgrspark.vm.network :forwarded_port, host: 8787, guest: 8787
```

y modificarla, eliminando el **#** inicial

3. ejecutar **vagrant reload**

Una vez realizado todo esto, para acceder al servidor de **RStudio** es necesario conectarse a la dirección <http://localhost:8787>, con usuario **vmuser** y contraseña **vmuser**

RStudio (y R) tienen acceso a todos los paquetes de R instalados en la VM, incluidos los que le permiten conectar con Spark.

Al estar corriendo dentro de la VM, en principio no puede acceder a ficheros que haya en el ordenador anfitrión. Para solventar esto se ha habilitado un subdirectorio **~/R** en la VM que está enlazado a la carpeta **./vmfiles/R** del host; de esta forma todo lo que pongamos en esa carpeta será visible por **RStudio**. Al provisionar la máquina ese directorio se configura automáticamente como el directorio de trabajo por defecto de **RStudio** (puede cambiarse en el menú de **RStudio**, en *Tools* → *Global Options* → *Default working directory*)

3.6 Provisionamiento adicional

El **Vagrantfile** contiene varias secciones que instalan software adicional de forma automatizada. Estas secciones no se ejecutan por defecto al arrancar la máquina; si se desean es necesario utilizar, con la máquina levantada, el comando

```
vagrant provision --provision-with <nombre>
```

para que se ejecuten (este proceso sólo es necesario realizarlo una vez). La siguiente tabla contiene las secciones disponibles para provisionamiento extra:

Nombre	Componente	Notas
rstudio	<i>Rstudio Server</i>	Instala RStudio Server (ver sección 3.5 <i>RStudio Server</i>)
nbc	<i>Notebook Convert</i>	Instala los componentes necesarios para poder descargar notebooks en formato PDF (vía LaTeX)
nbc.es	<i>nbconvert language</i>	Modifica el funcionamiento de nbconvert para adaptar la conversión a castellano
nlp	<i>Python NLP packages</i>	Instala en el virtualenv de Python algunos paquetes extra para procesado de lenguaje (nltk, sklearn_crfsuite, spacy)
mvn	<i>Maven</i>	Instala Maven
scala	<i>Scala development</i>	Instala Scala 2.11 y sbt. Nota: es para desarrollo y compilación en Scala, pero no es necesario para ejecutar Notebooks en Scala
dl	<i>Deep Learning libraries</i>	Instala Tensorflow y PyTorch en el virtualenv de Python, de forma que puedan usarse en notebooks

4. Administración de Spark

4.1 Arranque/parada

El servicio de *Notebooks* (*Jupyter*), que es quien arranca los kernels vía su interfaz Web (incluido el kernel de PySpark), está integrado como servicio de sistema, y por tanto se controla con `systemctl`:

- `sudo systemctl status notebook` informa del estado actual de funcionamiento
- `sudo systemctl stop notebook` lo para (por defecto arranca automáticamente)
- `sudo systemctl start notebook` lo arranca

4.2 Configuración

Dentro de la máquina virtual (es decir, tal y como se ve en una sesión de consola), Spark está instalado en `/opt/spark/current/`. En particular, los dos ficheros de configuración de Spark son:

<code>/opt/spark/current/conf/spark-env.sh</code>	Variables de entorno usadas por la instalación de Spark en la máquina virtual
<code>/opt/spark/current/conf/spark-defaults.conf</code>	Propiedades de configuración usadas por Spark

Los *notebooks* de Spark arrancan con la configuración definida por estos dos ficheros. Si se cambia alguno de ellos, será necesario rearrancar los kernels en los *notebooks* que en ese momento estén activos para que releen la configuración.

Si se está trabajando con tablas y Spark SQL, a estos ficheros hay que añadir el fichero de configuración de Hive:

<code>/opt/spark/current/conf/hive-site.xml</code>	Definición de Hive <i>warehouse</i> y <i>metastore</i>
--	--

Las tareas de Spark lanzadas dentro de la máquina virtual desde la consola (vía `spark-submit`, `pyspark`, etc) también usan las mismas propiedades, aunque pueden cambiarse en cada ejecución concreta usando [opciones de ejecución](#).

4.3 Solución de problemas

Si al crear o cargar un Notebook de PySpark parece como si no existiese un proceso de Spark:

- *no existe el `SparkContext`, que debería haberse creado automáticamente en la variable `sc`,*
- *ni la `SparkSession` que debería haberse creado en la variable `spark` (se obtienen errores del tipo `NameError: name 'spark' is not defined`)*

... puede deberse a problemas de concurrencia. El metastore DB definido por defecto en la máquina virtual en `hive-site.xml` es monoproceso, y usa un fichero único. Con la configuración por defecto sólo es posible tener arrancado un contexto SQL al mismo tiempo.

Por tanto si no se puede crear un nuevo contexto, es posible que ya haya otro notebook abierto de PySpark, que por tanto tiene un contexto SQL en uso. Es necesario cerrar ese notebook (apagando el kernel) antes de poder arrancar el nuevo⁵.

*La interfaz del servidor de Notebooks (Jupyter) tiene una pestaña que ofrece el listado de todos los Notebooks arrancados. Puede consultarse para saber si es necesario cerrar alguno. De hecho, al terminar de usar un Notebook es importante **cerrarlo apagando el kernel**, de forma que se libere la memoria. Para ello hay que usar la opción "Close and halt" del menú File.*

Otros posibles problemas son:

- La grabación de DataFrames como tablas del sistema precisa que exista el almacenamiento de destino. Por defecto ese almacenamiento es el directorio `/vagrant/hive/warehouse`, que en la VM está enlazado a la carpeta montada desde el host `./vmfiles/hive/warehouse`. Si la operación `saveAsTable()` produce un error, comprobar que esa carpeta existe.

4.4 Diagnóstico

Los errores que se produzcan durante la ejecución del código normalmente producirán mensajes de error dentro del *notebook* mismo que se pueden analizar (aunque analizar los errores producidos por Spark al ejecutar código en Pyspark no es fácil, ya que suele presentar una cadena de fallos mezclando los errores en los procesos Python lanzados por los ejecutores con las excepciones al nivel de la JVM).

Adicionalmente se escriben mensajes de log que también se pueden comprobar. Para ello es necesaria una sesión dentro de la máquina virtual (usando uno de los procedimientos indicados arriba); los ficheros de log son:

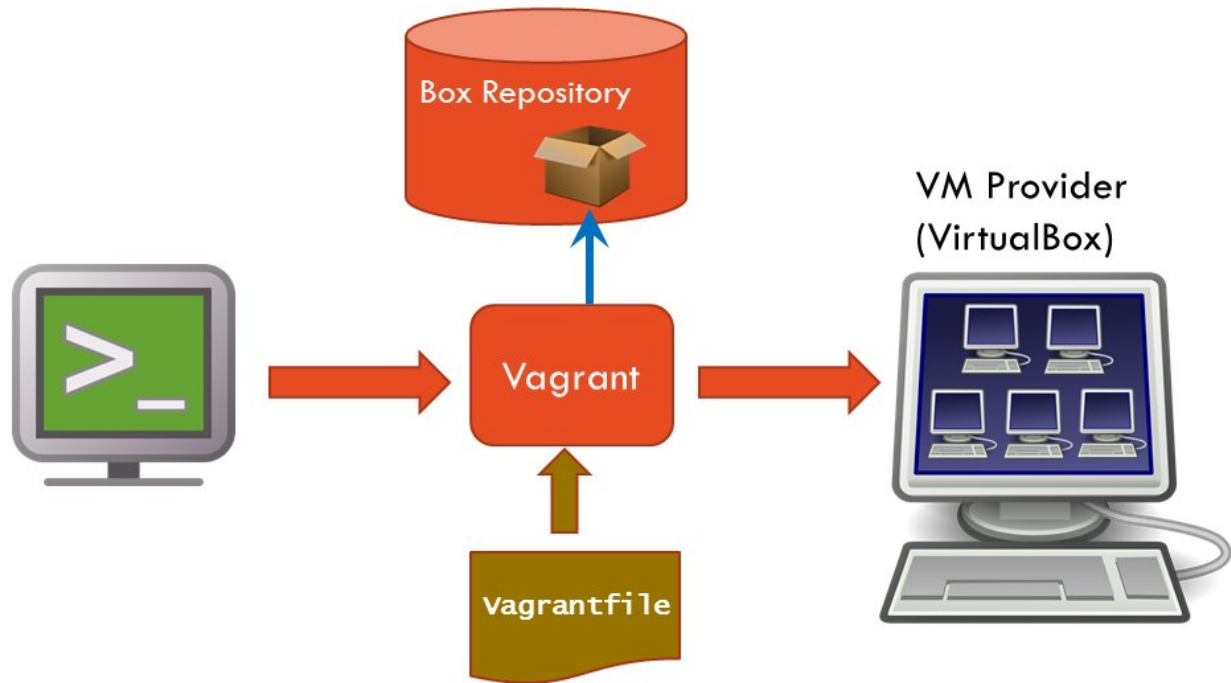
- El servidor de *Notebooks* escribe sus logs a `/var/log/ipnb/jupyter-notebook.out`

⁵ Se puede modificar la configuración de Hive usada por Spark para cambiar el lugar donde se escribe ese metastore haciéndolo local (lo que permitiría tener varios procesos Spark al mismo tiempo si los *Notebooks* están en directorios distintos) , o para usar un metastore con una base de datos más potente que admita concurrencia. Pero cada *Notebook* arrancado es un kernel conectado a un driver Spark, y ese proceso consume memoria. Tener varios *Notebooks* arrancados puede suponer dejar sin memoria a la máquina virtual, así que en una configuración con Spark local en una máquina virtual tampoco es recomendable tener varios procesos Spark simultáneos.

- El driver de Spark escribe a `/var/log/ipnb/spark.log`
- Los ejecutores de Spark locales escriben a `/var/log/ipnb/local-XXXXX`

A. Funcionamiento del proceso de instalación

Se incluye como referencia un diagrama del proceso de descarga, instalación y provisionado de la VM, con los componentes involucrados.



A rasgos generales, el **vagrantfile** contiene la especificación de la máquina virtual que necesita **Vagrant** para descargarla y configurarla:

- especifica la imagen base (*box*) que tiene que descargarse del repositorio de imágenes (**Vagrantcloud**)
- define las opciones de provisionado (instalación de software y configuración que se realiza una vez descargada la imagen base)

Los pasos son:

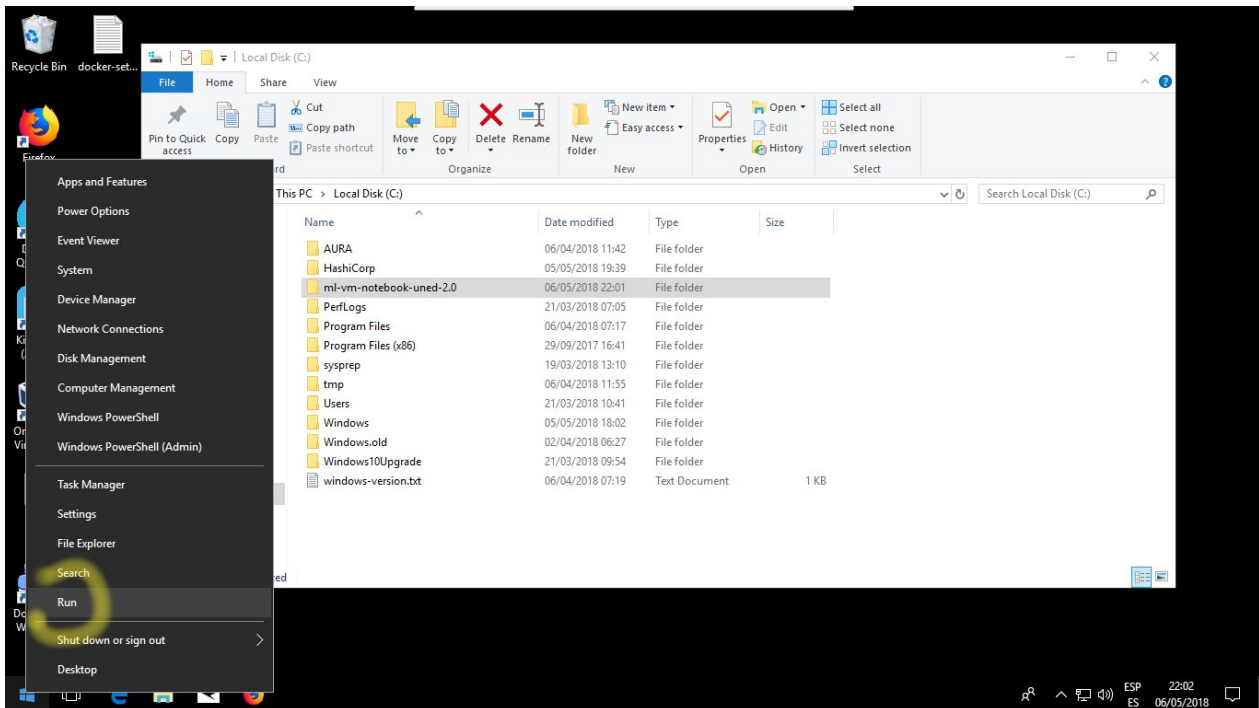
1. **Vagrant** analiza el **vagrantfile** y averigua los pasos necesarios
2. Se descarga de **Vagrantcloud** la *box* que se usará como base
3. Una vez descargada la imagen, se la pasa al ejecutor de máquinas virtuales, en este caso **VirtualBox**, para que la arranque
4. Cuando ya se ha arrancado, accede a ella e instala el software adicional necesario y la configura (usando las opciones de provisionado definidas en el **vagrantfile**)

Esto ocurre solo la primera vez que se ejecuta **vagrant up**; a partir de entonces las siguientes veces **Vagrant** simplemente arranca la imagen ya configurada vía **VirtualBox** y la deja funcionando

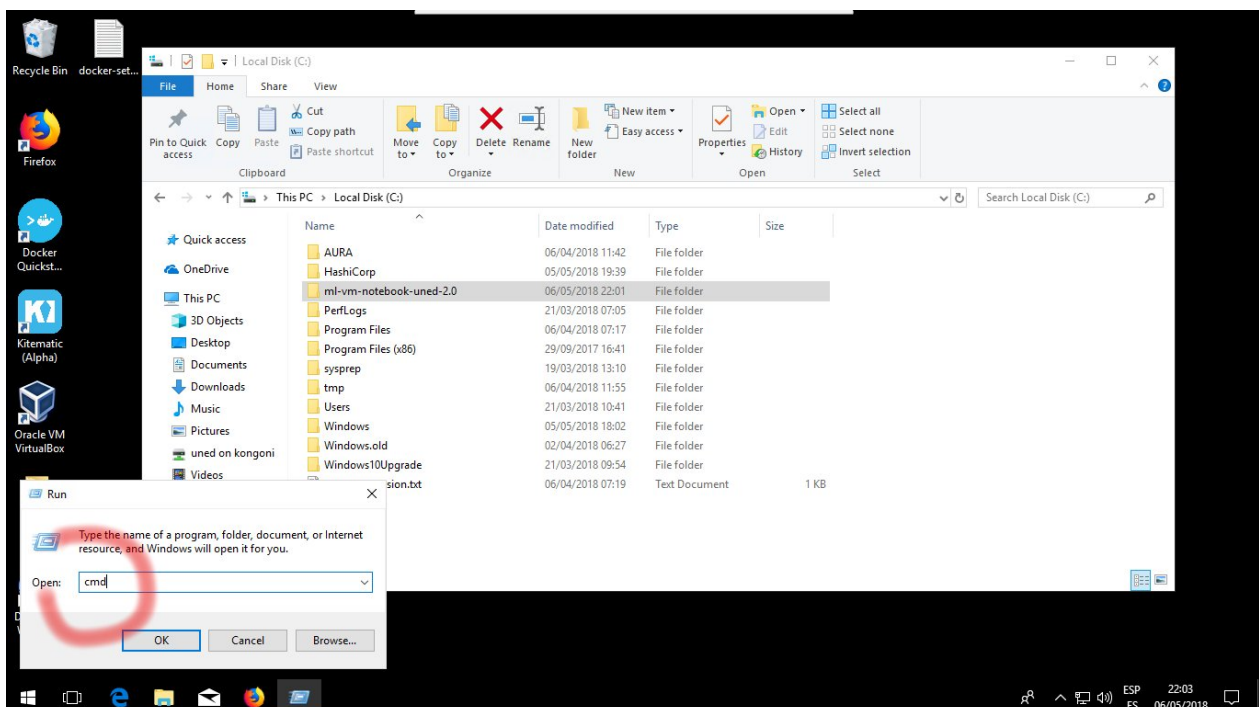
B. Interfaz de consola en Windows

La interfaz de consola en Windows es una aplicación que permite ejecutar algunas tareas administrativas por medio de comandos escritos por teclado, en vez de con una interfaz gráfica de ventanas. En Windows 10 se puede arrancar de esta forma:

- pulsar con el botón derecho sobre el icono de Windows para que aparezca el menú de opciones; seleccionar la opción “Ejecutar” o “Run”



- En la ventana que se abre, teclear `cmd` y pulsar *Enter*



La [página de la Wikipedia](#) sobre “Símbolo del sistema” tiene una lista de los comandos administrativos disponibles. Los comandos normalmente tienen parámetros adicionales, que se escriben a continuación del comando separándolos con espacios.

Al ser *Vagrant* una aplicación de consola, debe ser ejecutada dentro de una interfaz de consola de Windows. Para hacerlo, aparte de los comandos propios de *Vagrant* (“**vagrant up**”, “**vagrant halt**”, etc) sólo es necesario otro comando más de la consola: el de cambio de directorio **cd**, para cambiar la consola a la carpeta donde se encuentra el **Vagrantfile**.

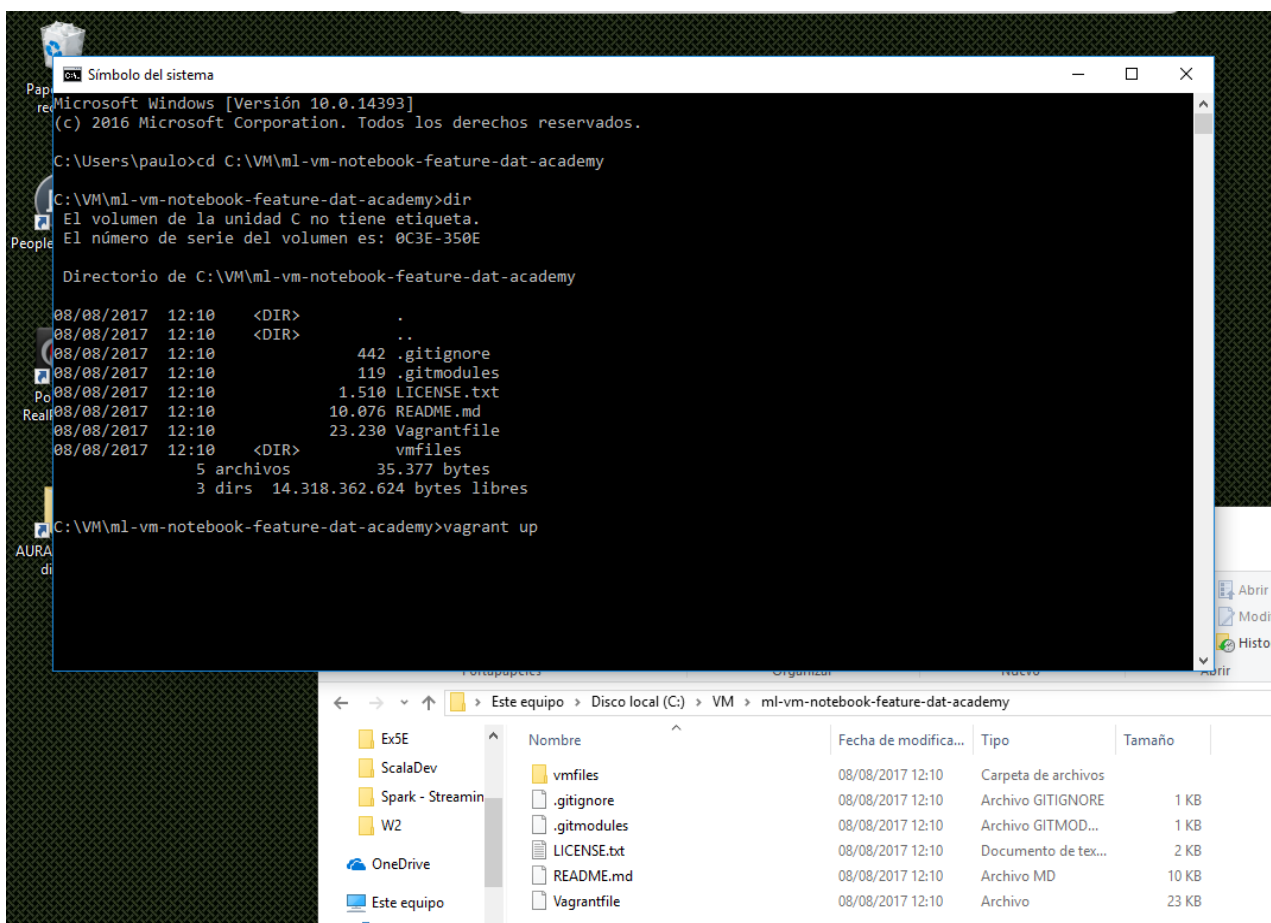
Dicho comando **cd** toma un parámetro más, el directorio al que queremos cambiar, por ejemplo

```
cd C:\VM\ml-notebook-uam-2.4
```

Si el nombre del directorio contiene espacios, entonces es necesario usar comillas⁶:

```
cd "C:\Users\Pedro Lopez\ml-notebook-uam-2.4"
```

Un comando adicional, **dir**, muestra los contenidos de la carpeta actual de la consola, con lo que puede servir para asegurarnos de que estamos en el directorio correcto.



⁶ Aunque se desaconsejan los directorios con espacios para albergar máquinas virtuales, porque pueden dar problemas.

C. Depuración del proceso de instalación y arranque

A continuación se describe el proceso normal de instalación de la VM usando *Vagrant*, y se comentan algunos posibles problemas que pueden surgir por distintos aspectos de la configuración del ordenador, cómo diagnosticarlos y su posible solución.

C.1 Instalación normal

Durante el proceso de instalación normal *Vagrant* va escribiendo mensajes de progreso, según

1. Descarga la Box (imagen base) de Internet
2. La instala como VM
3. La arranca y la configura (provisiona)

A continuación se muestran los mensajes de este último paso (provisionado)

```
==> vm-machinelearning-box: Running 'pre-boot' VM customizations...
==> vm-machinelearning-box: Booting VM...
==> vm-machinelearning-box: Waiting for machine to boot. This may take a few minutes...
vm-machinelearning-box: SSH address: 127.0.0.1:2222
vm-machinelearning-box: SSH username: vagrant
vm-machinelearning-box: SSH auth method: private key
vm-machinelearning-box: Vagrant insecure key detected. Vagrant will automatically replace
vm-machinelearning-box: this with a newly generated keypair for better security.
vm-machinelearning-box: Inserting generated public key within guest...
vm-machinelearning-box: Removing insecure key from the guest if it's present...
vm-machinelearning-box: Key inserted! Disconnecting and reconnecting using new SSH key...
==> vm-machinelearning-box: Machine booted and ready!
==> vm-machinelearning-box: Checking for guest additions in VM...
vm-machinelearning-box: The guest additions on this VM do not match the installed version of
vm-machinelearning-box: VirtualBox! In most cases this is fine, but in rare cases it can
vm-machinelearning-box: prevent things such as shared folders from working properly. If you see
vm-machinelearning-box: shared folder errors, please make sure the guest additions within the
vm-machinelearning-box: virtual machine match the version of VirtualBox you have installed on
vm-machinelearning-box: your host and reload your VM.
vm-machinelearning-box: Guest Additions Version: 5.2.18
vm-machinelearning-box: VirtualBox Version: 6.1
==> vm-machinelearning-box: Setting hostname...
==> vm-machinelearning-box: Mounting shared folders...
vm-machinelearning-box: /vagrant => /vm/Vagrant/ml-vm-notebook-uned-3.0/vmfiles
==> vm-machinelearning-box: Running provisioner: 01.nbuser (shell)...
vm-machinelearning-box: Running: inline script
==> vm-machinelearning-box: Running provisioner: 10.config (shell)...
vm-machinelearning-box: Running: inline script
vm-machinelearning-box: Creating Jupyter config
==> vm-machinelearning-box: Running provisioner: 13.ir (shell)...
```

El mensaje final “*The Vagrant Machine is up*” indica que la instalación ha funcionado bien:


```

vm-machinelearning-box: shared folder errors, please make sure the guest additions within the
vm-machinelearning-box: virtual machine match the version of VirtualBox you have installed on
vm-machinelearning-box: your host and reload your VM.
vm-machinelearning-box: Guest Additions Version: 5.2.18
vm-machinelearning-box: VirtualBox Version: 6.1
==> vm-machinelearning-box: Setting hostname...
==> vm-machinelearning-box: Mounting shared folders...
vm-machinelearning-box: /vagrant => /vm/Vagrant/ml-vm-notebook-uned-3.0/vmfiles
==> vm-machinelearning-box: Running provisioner: 01.nbuser (shell)...
vm-machinelearning-box: Running: inline script
==> vm-machinelearning-box: Running provisioner: 10.config (shell)...
vm-machinelearning-box: Running: inline script
vm-machinelearning-box: Creating Jupyter config
==> vm-machinelearning-box: Running provisioner: 13.ir (shell)...
vm-machinelearning-box: Running: inline script
vm-machinelearning-box: Installing IRkernel ...
vm-machinelearning-box: [InstallKernelSpec] Installed kernelspec ir in /home/vmuser/.local/share/ju
pyter/kernels/ir
==> vm-machinelearning-box: Running provisioner: 20.extensions (shell)...
vm-machinelearning-box: Running: inline script
vm-machinelearning-box: Installing notebook extensions
==> vm-machinelearning-box: Running provisioner: 21.nbconfig (shell)...
vm-machinelearning-box: Running: inline script
==> vm-machinelearning-box: Running provisioner: 50.nbstart (shell)...
vm-machinelearning-box: Running: inline script

==> vm-machinelearning-box: Machine 'vm-machinelearning-box' has a post `vagrant up` message. This is a
message
==> vm-machinelearning-box: from the creator of the Vagrantfile, and not from Vagrant itself:
==> vm-machinelearning-box:
==> vm-machinelearning-box: **** The Vagrant ML-Notebook machine is up. Connect to http://localhost:800
8
kongoni#17 ml-vm-notebook-uned-3.0>

```

C.2 Posibles errores

- Si al intentar descargar la imagen se genera un mensaje del tipo **Error: The requested URL returned error: 404 Not Found** puede deberse a una versión de *Vagrant* demasiado antigua: las versiones anteriores a 1.9.6 apuntan a un servidor *Vagrant* que ya no está operativo
- La descarga lleva tiempo (son 1,5 GB), pero con una conexión razonable es raro que tarde más de 15-20 minutos. Si se atasca (el informe de progreso no muestra variación, la velocidad de descarga es cero o muy baja y el tiempo estimado de terminación es muy grande) puede tener más sentido interrumpir el proceso y volver a lanzarlo (**vagrant up** de nuevo), a ver si al realizar una nueva petición de red va más ágil.

Interrumpir la operación y volver a lanzarla no supone perder lo descargado hasta el momento: la nueva operación retoma la descarga a partir de lo que ya estaba descargado, así que no hay problema en hacerlo.

- Si la provisión de la VM (la instalación y configuración de componentes durante el primer arranque) se interrumpe o falla por alguna razón, la máquina puede quedar en un estado inconsistente. En ese caso es mejor regenerarla mediante

```

vagrant destroy
vagrant up

```

esta regeneración tardará mucho menos que la primera instalación ya que la imagen base que fue descargada se mantiene cacheada en disco (es importante también asegurarse de que no hay problemas de espacio en disco)

- En ocasiones la máquina arranca pero luego *Vagrant* se para informando que no puede comunicarse con ella (con un mensaje de error sobre problemas con la conexión ssh). Puede intentarse obtener más información arrancando la máquina virtual en modo depuración (ver sección *C.3 Información de diagnóstico* más abajo).

Otra posibilidad es comprobar el estado de la máquina en la interfaz gráfica de *VirtualBox*, donde debería aparecer una entrada para la máquina creada (con el nombre que tiene la máquina virtual, *vm-machinelearning*). Trate de arrancar la máquina a través de esa interfaz (si no figura como encendida) o de abrir una sesión de consola en ella (si ya figura como en ejecución). Los ficheros de log de *VirtualBox* también pueden dar una pista (de nuevo, ver la sección *C.3 Información de diagnóstico*)

- Uno de los posibles problemas de arranque está motivado por tener desactivada la virtualización en el ordenador. En esos casos es frecuente ver en el arranque de *Vagrant* mensajes del tipo:

`timeout during server version negotiating`

Se debe a que, al no estar la virtualización hardware disponible, *VirtualBox* pasa a modo de emulación, mucho más lento, lo que produce los timeouts (la máquina virtual no contesta a tiempo).

Este error debería verse reflejado en los logs de *VirtualBox* (o en la interfaz gráfica, cuando se trata de arrancar la máquina directamente desde *VirtualBox* como se ha mencionado arriba); de nuevo ver apartado C.3 Información de diagnóstico para obtenerlos. El error que se produce indica que “*no está disponible la aceleración Intel VT-x o AMD-V*” (el tipo concreto de aceleración depende del hardware). Ejemplos posibles de estos mensajes de error que pueden aparecer en los logs son:

`No VT-x or AMD-V CPU extension found. Reason VERR_VMX_NO_VMX`

`Falling back to raw-mode: VT-x is disabled in the BIOS for all CPU modes`

Para solucionarlo es necesario configurar la virtualización en [la BIOS del ordenador](#).

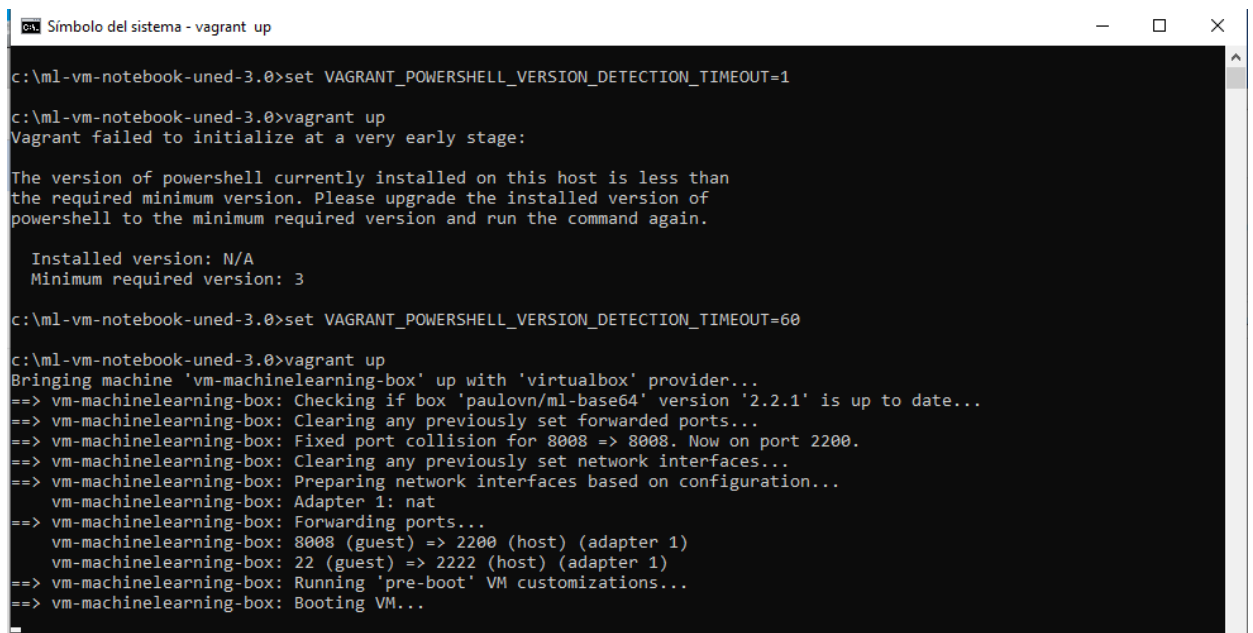
- El mismo problema puede ocurrir en Windows si se encuentra activada la virtualización Hyper-V, una tecnología de virtualización específica de Windows, que no es compatible con *VirtualBox* ya que bloquea la aceleración VT-x/AMD-V. Su efecto puede ser más rotundo (se pueden producir bloqueos completos de Windows, con pantallas azules, si se intenta arrancar una máquina virtual de *VirtualBox* estando Hyper-V activado).

La solución es [desactivar Hyper-V en la configuración de Windows](#) y rearrancar (las mismas instrucciones están en la [documentación de Vagrant](#)). Una vez desactivado, si la configuración de *Vagrant* quedó corrupta al colgarse Windows, puede ser necesario borrar los directorios de trabajo de *Vagrant* (`.vagrant` en la carpeta donde está el *vagrantfile*) y/o eliminar la máquina virtual antigua (usando `vagrant destroy` o en la interfaz de *VirtualBox*).

- Permisos en claves privadas (problema que puede surgir en ordenadores con Linux): los ficheros que contienen las claves SSH privadas que *Vagrant* usa para conectar con la VM tienen que tener permisos de acceso restringidos; si no los tienen *Vagrant* intenta modificarlos. En Linux esto implica que el sistema de ficheros tiene que soportar gestión de permisos; algunos sistemas (VFAT, NTFS) no lo admiten, y esto provoca un error del tipo `The`

private key to connect to this box via SSH has invalid permissions set on it. Para solucionarlo es necesario albergar los ficheros internos de *Vagrant* en un sistema de ficheros distinto (en el caso de que el sistema de ficheros sea NTFS, también sirve montarlo con la opción `permissions`).

- En Windows, las versiones modernas de *Vagrant* (1.9.6 en adelante) exigen que esté presente PowerShell (un componente estándar de Windows) en versión 3 o superior. Windows 7, sin embargo, venía por defecto con la versión 2.0. Por ello es posible que en ordenadores con Windows 7 sin actualizar aparezca el mensaje “The version of powershell currently installed on this host is less than the required minimum version. Please upgrade the installed version of powershell to the minimum required version and run the command again.”. La solución es [actualizar Powershell](#).
- Este último mensaje (*versión de PowerShell inválida*) también se ha dado algunas veces en Windows 10. Sin embargo en Windows 10 el mensaje es engañoso, porque W10 **sí** tiene la versión correcta de PowerShell. Aquí lo que puede pasar es que *Vagrant* no es capaz de averiguar la versión de PowerShell por un problema de timeouts (PowerShell no le contesta a tiempo). En ese caso se puede solucionar especificando un timeout mayor que el valor por defecto. La siguiente imagen muestra la situación:



```
c:\ml-vm-notebook-uned-3.0>set VAGRANT_POWERSHELL_VERSION_DETECTION_TIMEOUT=1
c:\ml-vm-notebook-uned-3.0>vagrant up
Vagrant failed to initialize at a very early stage:

The version of powershell currently installed on this host is less than
the required minimum version. Please upgrade the installed version of
powershell to the minimum required version and run the command again.

    Installed version: N/A
    Minimum required version: 3

c:\ml-vm-notebook-uned-3.0>set VAGRANT_POWERSHELL_VERSION_DETECTION_TIMEOUT=60
c:\ml-vm-notebook-uned-3.0>vagrant up
Bringing machine 'vm-machinelearning-box' up with 'virtualbox' provider...
==> vm-machinelearning-box: Checking if box 'paulovn/ml-base64' version '2.2.1' is up to date...
==> vm-machinelearning-box: Clearing any previously set forwarded ports...
==> vm-machinelearning-box: Fixed port collision for 8008 => 8008. Now on port 2200.
==> vm-machinelearning-box: Clearing any previously set network interfaces...
==> vm-machinelearning-box: Preparing network interfaces based on configuration...
    vm-machinelearning-box: Adapter 1: nat
==> vm-machinelearning-box: Forwarding ports...
    vm-machinelearning-box: 8008 (guest) => 2200 (host) (adapter 1)
    vm-machinelearning-box: 22 (guest) => 2222 (host) (adapter 1)
==> vm-machinelearning-box: Running 'pre-boot' VM customizations...
==> vm-machinelearning-box: Booting VM...
```

Como se puede ver, primero usamos un timeout pequeño (1 segundo) y obtenemos el error. Pero si ampliamos el timeout a 60 segundos, el error ya no sucede y la máquina arranca.

Por tanto para solucionar este problema puede ejecutarse

```
set VAGRANT_POWERSHELL_VERSION_DETECTION_TIMEOUT=60
```

antes de ejecutar `vagrant up`

- Un posible error, también en Windows, es [problemas de codificación de caracteres](#), con mensajes del tipo `join: incompatible character encodings: CP850 and Windows-1252`. Esto puede pasar si la carpeta donde *Vagrant* intenta instalar su directorio interno de trabajo contiene caracteres no ASCII (eñes, acentos, etc). El resultado es una salida como esta:

```
Simbolo del sistema
INFO warden: Beginning recovery process...
INFO warden: Recovery complete.
INFO environment: Released process lock: machine-action-07ed20bf491b3548625ddb5b4955494
INFO environment: Running hook: environment_unload
INFO runner: Preparing hooks for middleware sequence...
INFO runner: 1 hooks defined.
INFO runner: Running action: environment_unload #<Vagrant::Action::Builder:0x0000000046ce158>
C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/childprocess-0.6.3/lib/childprocess/windows/process_builder.rb:43:in `join': incompatible character encodings: Windows-1252 and UTF-8 (Encoding::CompatibilityError)
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/childprocess-0.6.3/lib/childprocess/windows/process_builder.rb:43:in `create_command_pointer'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/childprocess-0.6.3/lib/childprocess/windows/process_builder.rb:27:in `start'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/childprocess-0.6.3/lib/childprocess/windows/process.rb:70:in `launch_process'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/childprocess-0.6.3/lib/childprocess/abstract_process.rb:82:in `start'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/lib/vagrant/util/subprocess.rb:153:in `block in execute'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/lib/vagrant/util/safe_chdir.rb:26:in `block (2 levels) in safe_chdir'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/lib/vagrant/util/safe_chdir.rb:25:in `chdir'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/lib/vagrant/util/safe_chdir.rb:25:in `block in safe_chdir'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/lib/vagrant/util/safe_chdir.rb:24:in `synchronize'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/lib/vagrant/util/safe_chdir.rb:24:in `safe_chdir'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/lib/vagrant/util/subprocess.rb:152:in `execute'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/lib/vagrant/util/subprocess.rb:22:in `execute'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/plugins/providers/virtualbox/driver/base.rb:451:in `block in raw'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/lib/vagrant/util/busy.rb:19:in `busy'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/plugins/providers/virtualbox/driver/base.rb:450:in `raw'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/plugins/providers/virtualbox/driver/base.rb:388:in `block in execute'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/lib/vagrant/util/retryable.rb:17:in `retryable'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/plugins/providers/virtualbox/driver/base.rb:383:in `execute'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/plugins/providers/virtualbox/driver/version_6_0.rb:71:in `import'
    from C:/HashiCorp/Vagrant/embedded/gems/2.2.7/gems/vagrant-2.2.7/plugins/providers/virtualbox/action/import.rb:5
```

... en la que se ve el mensaje de error de **incompatible character encodings** con una gran traza de error debajo.

Como esa carpeta por defecto de **Vagrant** está debajo de la carpeta personal, si el nombre de usuario contiene esos caracteres se puede producir el error (por ejemplo, si la carpeta del usuario es **C:\Users\Iñigo**). Para solucionarlo hay que cambiar los *directorios de trabajo* de **Vagrant** y de **VirtualBox**:⁷

- Borrar los directorios antiguos, por si contienen restos
- El *directorio de trabajo* de **VirtualBox** (la carpeta donde **VirtualBox** guarda sus imágenes) suele ser **HOME\VirtualBox Vms** (por ejemplo, **C:\Usuarios\Iñigo\VirtualBox Vms** daría problemas). Para cambiar esa carpeta se puede hacer en la interfaz de **VirtualBox**, en **File** → **Preferences** → **General** → **Default Machine Folder**. Cambiarla a algo como **C:\VirtualBox**

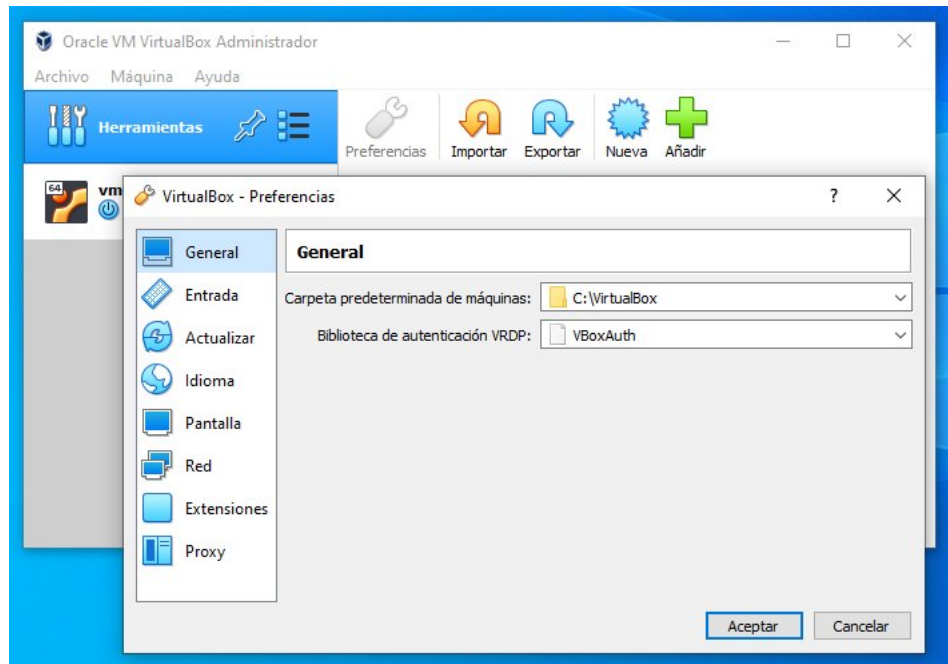
Puede verse esa interfaz en la imagen de la siguiente página

- El *directorio de trabajo* de **Vagrant** se ajusta mediante una variable de entorno. Se puede asignar esa variable de entorno justo antes de arrancar **Vagrant** en la ventana de consola (hay que *crear* primero ese directorio):

```
set VAGRANT_HOME=C:\Vagrant
vagrant up
```

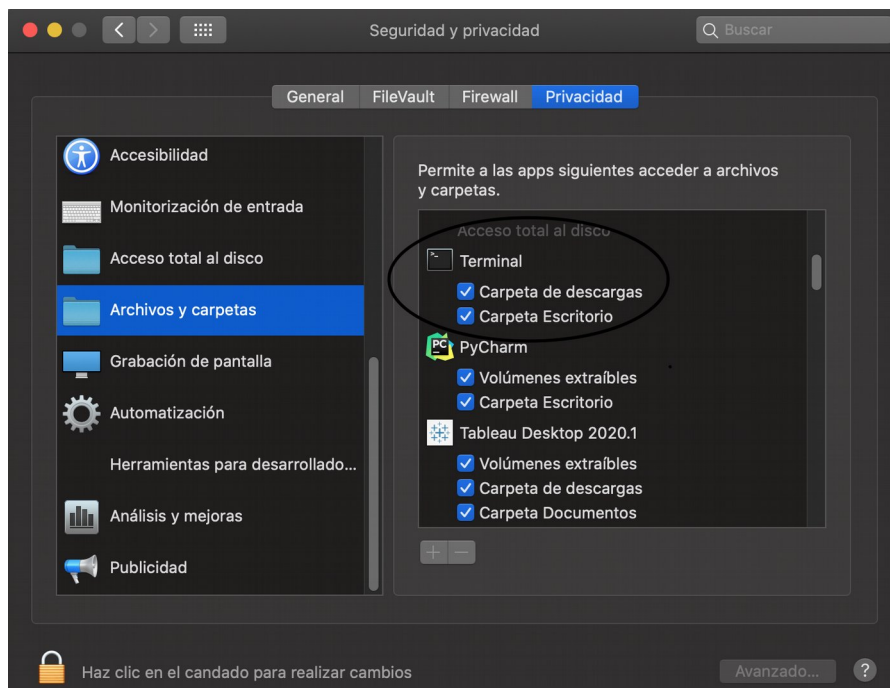
⁷ Nota: los *directorios de instalación* del software, que suelen ser del tipo **C:\Archivos de Programa\VirtualBox**, pueden dejarse a sus valores normales

Si se soluciona, conviene asignar la variable de entorno **VAGRANT_HOME** de forma permanente, ya que el comando **set** solo tiene vigencia en la sesión de consola en la que se ejecuta.



*Nota: otra posible vía de solución a este problema es definir un usuario **distinto** de Windows, que tenga un nombre sin caracteres especiales, y efectuar las operaciones de instalación usando ese usuario*

- Los usuarios de MacOS Catalina pueden necesitar ajustar permisos del sistema para permitir a una ventana de terminal realizar las acciones necesarias. La siguiente imagen muestra esos permisos; se puede encontrar una descripción de los ajustes necesarios en [la documentación de Vagrant](#)



C.3 Información de diagnóstico

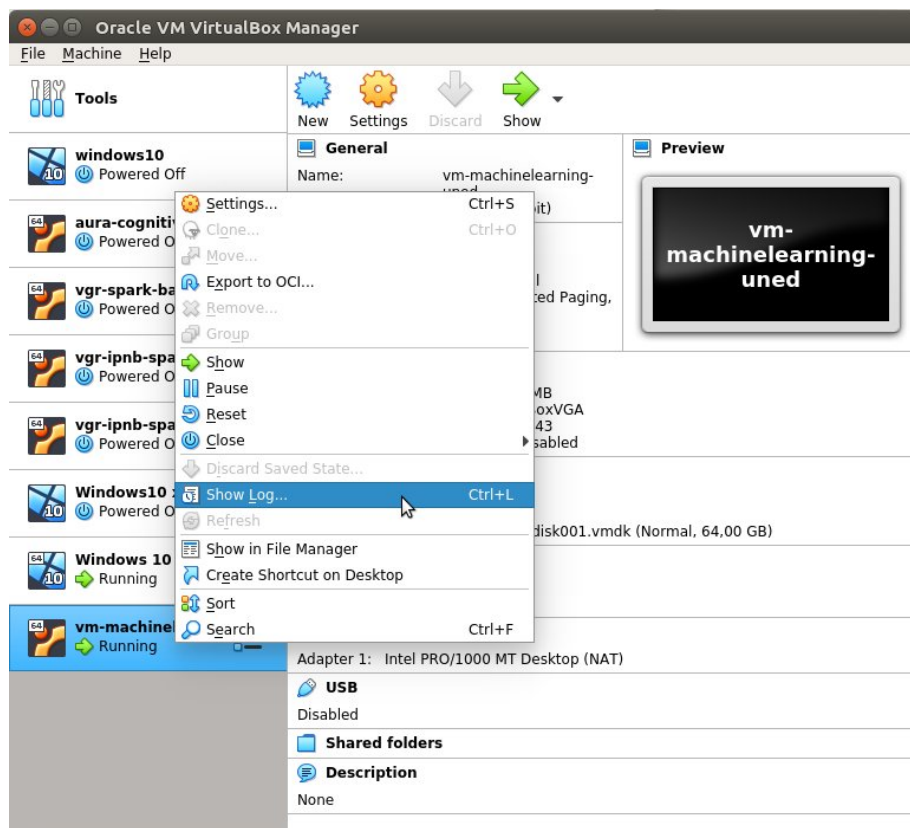
Para ayudar a diagnosticar los problemas mencionados arriba, u otros que pudieran surgir, es útil poder revisar las trazas de arranque y los ficheros de registro (*logs*) de la VM. El modo de conseguirlos, es:

- Ejecutar *Vagrant* en modo *debug*, redirigiendo la salida a fichero. En la consola de Windows, por ejemplo, se puede ejecutar:

```
vagrant up --debug >debug1.txt 2>debug2.txt
```

Esto produce dos ficheros, *debug1.txt* y *debug2.txt* (el primero con la salida estándar de ejecución, el segundo con la salida de depuración, mucho más extensa) que pueden ayudar al diagnóstico (se puede consultar también la página de la [documentación de Vagrant sobre depuración](#))

- En *VirtualBox* es posible acceder al fichero de log de arranque de la máquina virtual usando la interfaz gráfica:
 1. Arrancar la interfaz gráfica de *VirtualBox* (ejecutando el icono que estará instalado en el escritorio)
 2. La máquina virtual creada (*vm-ipnb-spark*) debería aparecer en el listado de VMs.
 3. Seleccionándola con el botón derecho, aparece un menú con una opción "*Show Log*", que muestra el log desde el lado de *VirtualBox*. La ventana contiene un botón "Save" que permite grabar ese log a un fichero.



También debería ser posible acceder a esos logs directamente, buscando la carpeta donde se almacenan. Esa carpeta depende del sistema operativo del ordenador anfitrión y de la versión de **VirtualBox**, pero suele ser en algún sitio similar a

HOME\VirtualBox\Machines\<nombre-de-la-vm>\Logs

donde **HOME** es el directorio de la cuenta de usuario). Hay dos ficheros de log: uno genérico (**VboxSVC**) y otro específico de la máquina virtual que estamos usando (este último fichero puede no existir si algún problema ha impedido arrancar a la máquina)

