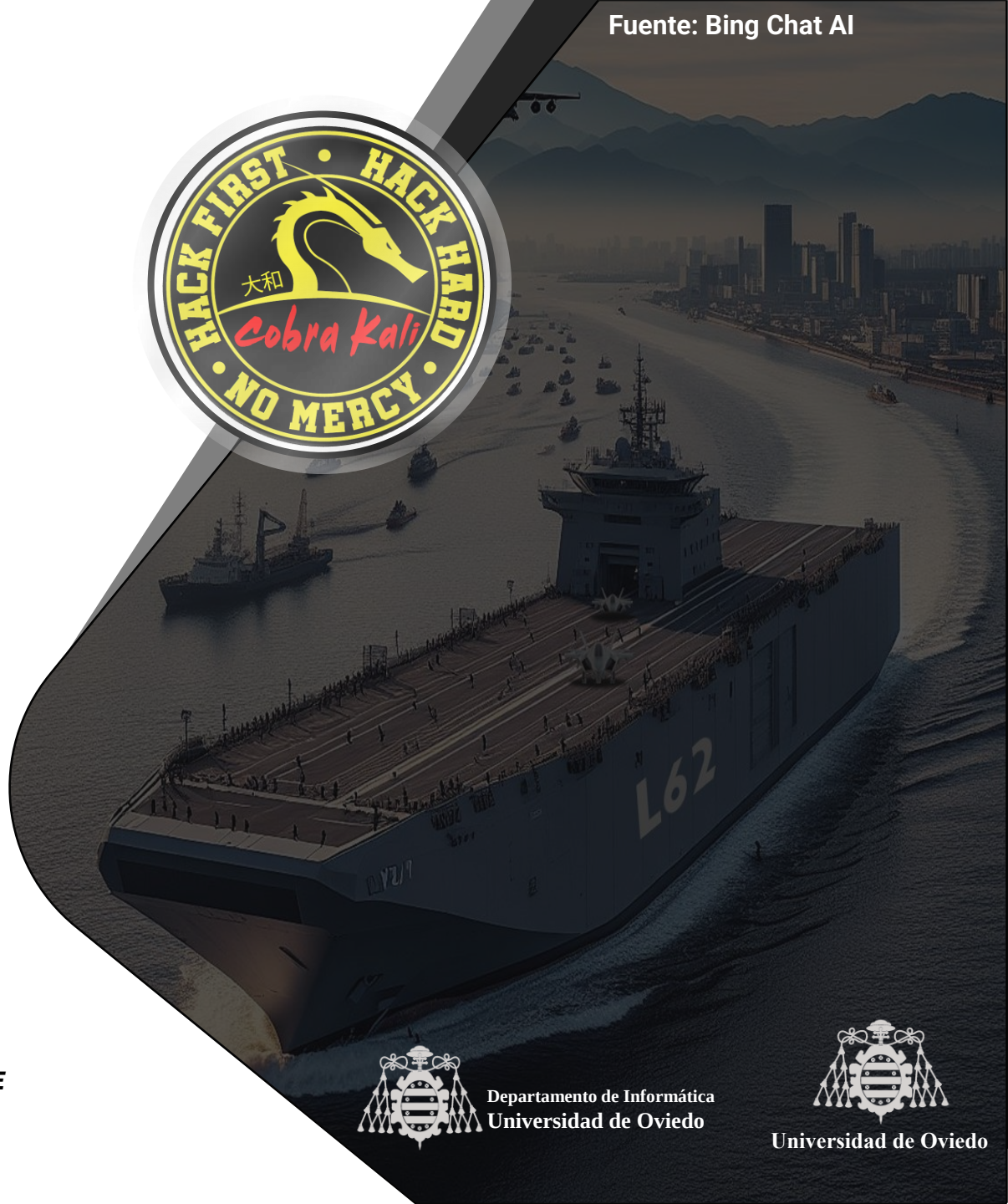
# INTRODUCCIÓN A VAGRANT

## Creando máquinas virtuales a partir de código


**José Manuel Redondo López** Proyecto "*L-62 Princesa de Asturias*" v1.2




Fuente: Bing Chat AI

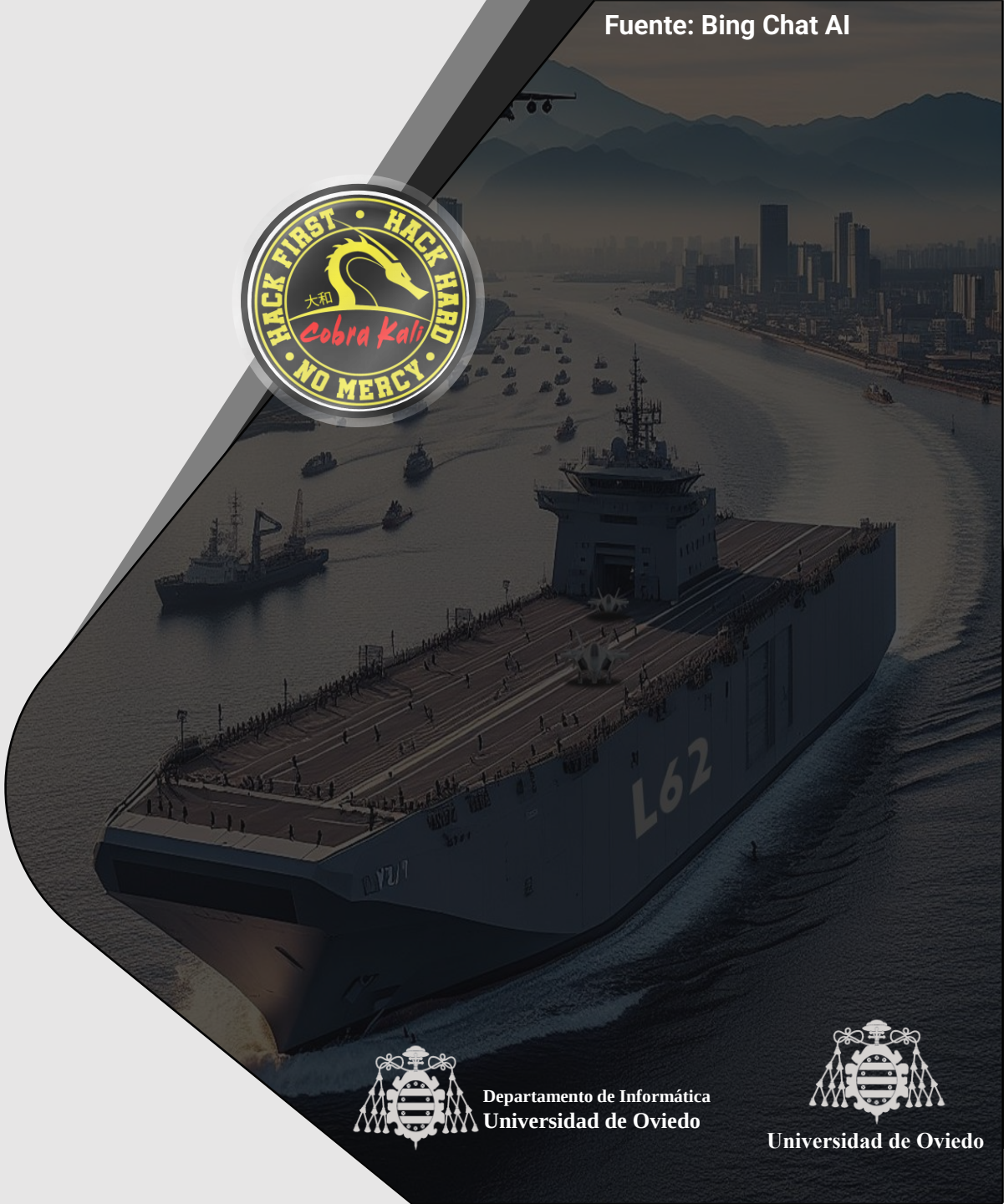



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo


# ÍNDICE

< Ir al Índice




Fuente: Bing Chat AI

# Introducción: Primera toma de contacto con Vagrant

## Empezar a trabajar con la herramienta


Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

# ¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- **Qué es** Vagrant y cómo se usa habitualmente
- A **crear y validar** Vagrantfiles
- Cómo **poner en marcha** una máquina virtual inicial con Vagrant
- El **manejo básico** de una máquina virtual creada con Vagrant

# ¿Qué ventajas me aporta Vagrant?

José Manuel
Redondo López

- **Como dijimos en la introducción, con Vagrant se puede configurar una máquina virtual con ficheros de código**
  - A estos ficheros se les denomina Vagrantfiles
- **¿Qué se puede incluir en estos ficheros?**
  - **Toda la configuración** de la máquina (Cores, RAM, HD, redes, carpetas compartidas…)
  - **Lo que se quiere ejecutar** en la misma (herramientas, librerías,…) (**provisionamiento**)
  - **Características particulares** de los proveedores de virtualización a usar
    - Locales o en nube
    - Una misma máquina Vagrant se puede preparar para varios proveedores de virtualización
    - Y especificar opciones distintas para cada uno de ellos en su Vagrantfile
- **Disponible en Windows, Linux y Mac**
  - Idealmente tendremos **MVs mediante código…¡multiplataforma y multi-proveedor de virtualización!**
  - Todo ello de manera **automatizable y reproducible**

# Usando Vagrant: workflow

José Manuel
Redondo López

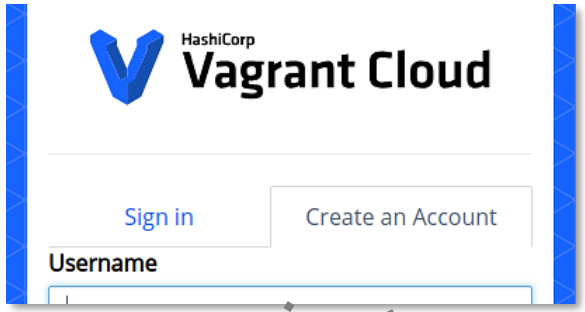
- **Trabajar con Vagrant implica entender este workflow**
  - Especificamos qué MV queremos y qué vamos a instalar en ella (provisionamiento) mediante código
  - La especificación se hace a partir de una **imagen base** llamada **box**
    - Las **boxes** las podemos crear o **descargar de un repositorio** de imágenes prefabricadas (mucho más fácil)
    - Muchas se crean para que sean compatibles con 1 o varias plataformas de virtualización o **hipervisores**
    - VirtualBox es el más usado normalmente, pero también hay Hyper-V, VMWare, libvirt…
  - Este código fuente **lo podemos llevar a cualquier máquina** con Vagrant instalado
  - Vagrant creará la MV sobre alguno de los hipervisores instalados en las máquinas destino
    - Si hay varios, podremos elegir
    - Si sólo hay uno, se elegirá automáticamente (si la imagen base es compatible)
  - Hecho esto, **no tenemos que preocuparnos de configurar nada más**
  - Todos los sitios donde despleguemos la MV **tendrán la misma máquina**
    - Idénticas características, configuración, software, usuarios, ficheros, etc.
    - **¡Y reproducible a voluntad!**

# ESQUEMA GENERAL DE FUNCIONAMIENTO

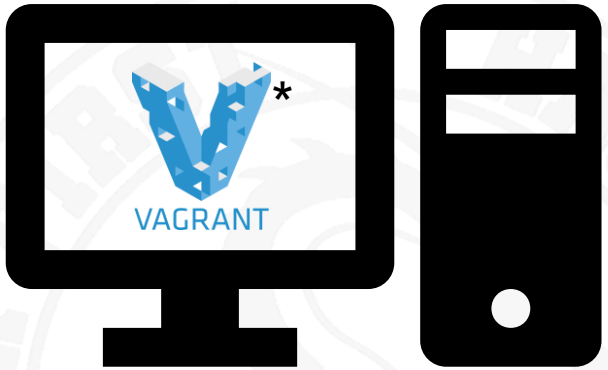
José Manuel
Redondo López

* Logo oficial de Vagrant

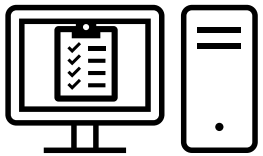
https://app.vagrantup.com/boxes/search




Máquina con Vagrant instalado




4. MV completamente configurada y lista para usar



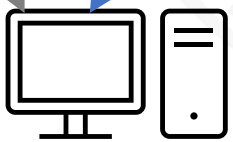
Usa un nombre de box de

Vagrantfile

1. Lee

2. Crea MV

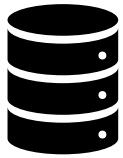
Descarga base box



Clon de la base box

3. Lee qué debe instalar en la MV (provisioner referenciado por la Vagranfile)

Disco Local



# Usando Vagrant: creando código

José Manuel
Redondo López

- **Para crear una máquina Vagrant necesitamos dos tipos de ficheros**
  1. La Vagrantfile: donde especificamos cómo queremos que sea la máquina
  2. **Ficheros de aprovisionamiento**: Scripts bash o de software de aprovisionamiento (Ansible, Chef)
     - Para decir **qué queremos instalar en la máquina**
     - Estos ficheros se ejecutan desde la Vagrantfile
- **¿Cómo creamos una Vagrantfile?**
  - Creamos una carpeta y accedemos a ella
  - `vagrant init <nombre de la imagen base>` crea un fichero `Vagrantfile`
    - Con una **referencia a la imagen base** indicada
    - Y un "**esqueleto**" con opciones comentadas
    - ¡Podemos **personalizarlo** y crear la MV a gusto!

```
# -*- 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 = "ubuntu/focal64"

  # 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"
```

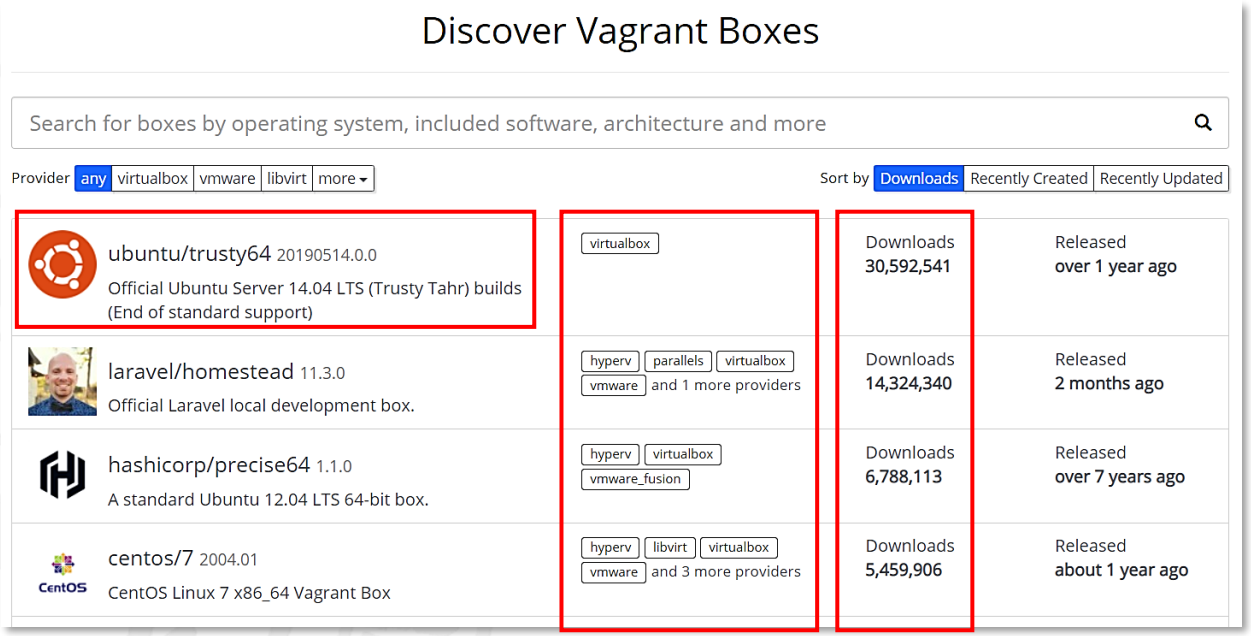
➢ Ejercicio **N1VAG_BASE_VFILES**

# Usando Vagrant: nombre de la máquina

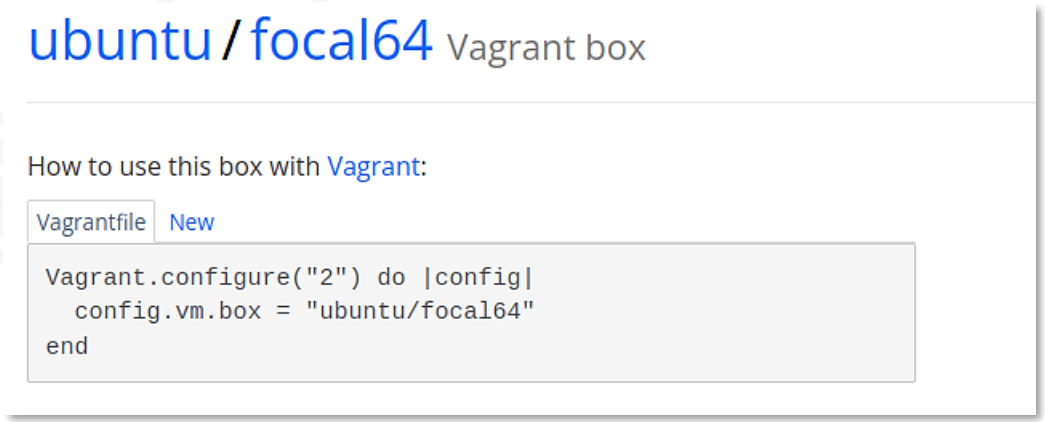
José Manuel
Redondo López

- **Debemos usar un nombre de imagen base adecuado**
  - Todos los nombres de imágenes base disponibles públicamente están en Vagrant Cloud
    - https://app.vagrantup.com/boxes/search
- **Debemos**
  - **Usar sólo imágenes base de proveedores oficiales** para evitar errores o malware
    - ¡Cualquiera puede subir boxes a Vagrant Cloud!
    - Proveedores oficiales como `ubuntu` o `hashicorp` (**prefijo del nombre de la box**) dan más garantías
      - Ejemplo: `hashicorp/bionic64`
  - Usar una que sea **compatible con nuestro proveedor de virtualización** (2ª columna)
    - Las hay compatibles solo con 1 o con muchos

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




https://app.vagrantup.com/ubuntu/boxes/focal64



# Usando Vagrant: lenguaje

José Manuel
Redondo López

- Los ficheros **Vagrant** usan sintaxis del **lenguaje Ruby**
- Sintaxis de POO estándar con objetos predefinidos en bloques (entornos)
  - Tenemos un **contexto global de configuración**, asociado al objeto **`config`**
    - Puedes cambiarle el nombre
  - Dentro del bloque podemos usar los atributos de ese objeto definidos en la documentación
    - https://www.vagrantup.com/docs/vagrantfile/machine_settings
  - El fichero generado ya tiene un gran nº de atributos de **`config`** comentados a consultar

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

**Contexto global de configuración (bloque)**

```
Vagrant.configure("2") do |config|
…
  config.vm.box = "hashicorp/bionic64"
… (gran número de opciones comentadas)
end
```

# Usando Vagrant: Arranque, parada, etc. de MV

José Manuel
Redondo López

- Ahora podemos, **desde el mismo directorio** de la `Vagrantfile`:
  - **Validar que el fichero no tiene problemas conocidos**: `vagrant validate`
  - Arrancar la máquina: `vagrant up`
    - Esto **construirá** la máquina virtual especificada si es la primera vez que lo llamamos y la arrancará
    - Sucesivas llamadas solo la arrancarán, salvo que llamemos a `vagrant destroy`
  - Entrar en sesión con `vagrant ssh` (o con la GUI del proveedor de virtualización)
  - Podemos parar la máquina con `vagrant halt`
  - `vagrant reload` implementará cambios de la configuración en la Vagranfile o de lo que se instale en la máquina (**provisioners**)
    - Sin necesidad de reconstruirla de nuevo
    - Hay más opciones de gestión: https://www.vagrantup.com/docs/getting-started/teardown.html

```
operario@ubuntuserver:~/directory$ vagrant ssh
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-23-generic-pae i686)

 * Documentation:  https://help.ubuntu.com/
New release '14.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Welcome to your Vagrant-built virtual machine.
Last login: Fri Sep 14 06:22:31 2012 from 10.0.2.2
vagrant@precise32:~$ ls
postinstall.sh
vagrant@precise32:~$ _
```

➢ Ejercicio **N1VAG_INTERACCION**

# ¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

?

- *¿Entiendes que con **Vagrant** puedes crear una máquina virtual a partir de un simple fichero de código?*
- *¿Puedes recordar la secuencia de pasos estándar para trabajar con **Vagrant**?*
- *¿Sabes crear una **Vagrantfile** inicial?*
- *¿Sabes qué es la **Vagrant Cloud**?*
- *¿Entiendes qué es un entorno global de configuración en **Vagrant**?*
- *¿Recuerdas cómo arrancar, parar y reiniciar una máquina **Vagrant**?*
- *¿Recuerdas qué es el provisionamiento?*
- *¿Sabes cómo validar una **Vagrantfile** para comprobar que no tiene errores?*

< Ir al Índice

Fuente: Bing Chat AI

Proveedores >

# Manejando imágenes base de Vagrant (Boxes)

## Cómo usar los puntos de partida para nuestras propias MV



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

# ¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- **Qué es una** Vagrant Box y cómo se usa
- Cómo **crear una** Vagrant Box personalizada a partir de una ya creada en funcionamiento
- Como crear una Vagrant Box personalizada **a partir de código**
- El uso de **variables** en las Vagrantfiles
- El concepto de **contexto** en las Vagrantfiles
- Como gestiona Vagrant los **proveedores de virtualización** y como hacer configuraciones para cada uno de ellos

# Boxes

José Manuel
Redondo López

- Las **Vagrant Box** "base" como `hashicorp/bionic64` se usan para clonar máquinas virtuales de forma rápida y eficiente
- Si intentamos usar un nombre de **box** que no exista localmente, se descargará automáticamente (si existe) del **Vagrant Cloud**
  - Cada vez que usemos una **box** se guardará localmente (caché)
- Si vamos a usar una misma imagen base repetidamente, es conveniente instalarla de antemano
  - Con `vagrant box add`
  - Así podrán reutilizarla todos los proyectos que queramos
  - **Ejemplo**: `vagrant box add hashicorp/bionic64`
- Hay varias formas de especificar la **box** base a usar

# Formas de uso de una Box en una Vagrantfile

José Manuel
Redondo López

1. La primera forma es la directa, que ya hemos visto

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"
end
```

2. Las **boxes suelen tener versiones** y podemos especificar una concreta
   - Las versiones aparecen en su ficha en la Vagrant Cloud

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"
  config.vm.box_version = "1.1.0"
end
```

3. Y si la **box** está en una URL (`http://`) o fichero local (`file://`)...

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"
  config.vm.box_url = "http://files.vagrantup.com/bionic64.box"
end
```

# Boxes

José Manuel
Redondo López

- La 3ª forma permite el uso de boxes que no estén en la Vagrant Cloud
  - Suele usarse con las que creamos a mano, personalizadas
- Cada box se guarda globalmente
  - Ningún proyecto que use una box como base puede modificarla
  - Es decir, **los cambios hechos en un proyecto a una box no afectan a otros proyectos que la usen**
- Los nombres de las box tienen dos partes, separadas por /
  - **La primera** es el proveedor de la box o usuario que la subió a la Vagrant Cloud
    - Ej.: Hashicorp (el fabricante de Vagrant), Ubuntu (la oficial de Canonical), etc.
  - **La segunda** es el nombre de la máquina
  - Los nombres son asignados por sus autores
    - **No hay control sobre el contenido** de una box, **podría tener malware**
      - https://blog.ryanjarv.sh/2019/06/08/malicious-vagrant-boxes.html
      - Hashicorp elimina boxes con malware si alguien les informa de ellas previamente (support+vagrantcloud@hashicorp.com), pero...¿y mientras tanto?
    - Por lo que hay que intentar usar siempre **imágenes de proveedores oficiales**
    - **REPITO: Debemos tener especial cuidado con esto al descargar imágenes públicas**

# Box personalizadas sin código

José Manuel
Redondo López

- **Una forma sencilla de crear una box personalizada como base para futuras MVs es hacerlo a partir de una preconstruida**
  - Se arranca una máquina basándose en una imagen que ya tenemos
    - Por ejemplo, `ubuntu/focal64` que usamos antes
  - Se hace `ssh` a ella y se instalan todos los programas / ficheros / etc. que nos hagan falta como lo haríamos habitualmente (`apt`)
  - Una vez hayamos terminado, salimos de la sesión SSH con `exit` y desde el mismo directorio donde está la `Vagrantfile` hacemos `vagrant package`
  - La MV exportada se escribirá en `package.box` (podemos renombrarla luego)
  - Ahora podemos usarla de la tercera forma vista anteriormente
- **Pero…¡así no usamos código para su creación!** ☹
  - Debemos usar las **Vagrantfile** vistas antes

# Personalizando nuestra MV con código

José Manuel
Redondo López

- Las **Vagrantfiles** admiten la creación de variables y valores en cualquier parte del fichero
- Sus valores pueden usarse para dar valor a cualquier propiedad de un objeto
  - Ej.: `vb.memory = RAM`
- Dentro del global se pueden crear **contextos de configuración**
  - **Anidados por proveedor de virtualización**
  - Y en cada uno de ellos se podrán usar objetos y atributos correspondientes sólo a dicho proveedor
  - Lo veremos en la siguiente sección

```
# Variables
RAM = 2048
CORES = 2
…
# Contexto de configuración global
Vagrant.configure("2") do |config|
  config.vm.box = BOX_NAME
```

**Solo se usará si el proveedor elegido es VirtualBox**

```
  # Contexto de configuración de virtualbox
  config.vm.provider "virtualbox" do |vb|
    vb.name = MACHINE_NAME
    vb.memory = RAM
    vb.cpus = CORES
    vb.gui = GUI
    …
  end
  …
end
```

➢ Ejercicio **N2VAG_VARS, N2VAG_BASICPARAM**

# Uso de variables

José Manuel
Redondo López

```
# Please use only Vagrant Boxes from the Vagrant Cloud that come
# from trusted or official providers!

## Machine Info:
## -------------

# This base Vagrant box supports VirtualBox, Hyper-V and VMWare Desktop.
BOX_NAME = "hashicorp/bionic64"

# The name of the box that will appear into the GUI
MACHINE_NAME = "Ubuntu SSI Image"

# The description of the box that will show into the GUI
MACHINE_DESCRIPTION = "The base Ubuntu image for the EII SSI course"

# Type of OS, needed for some of the Hypervisors
OS_TYPE = "Ubuntu_64"

# This determines the set of customization scripts that will be used
OS_CLASS = "debian"

## Machine Characteristics:
## ------------------------

# In Mb, if you can afford it, more memory is always welcome :).
# Be careful not to push this too hard!
RAM = 2048

# If you can afford it, more cores are always welcome :).
# Be careful not to push this too hard!
CORES = 2

# Size of the box disk
DISK_SIZE = "80GB"

# Amount of available VRAM. Consider if your VM is going to have a GUI
# or not to choose this.
VRAM = 128
```

CompleteVagrantfile

- Para hacer código elegante y mantenible, con variables puedes **parametrizar TODO**

```
# The name of the HD controller greatly varies depending on the box you use
HD_CONTROLLER = "SATA Controller"

## Virtualization parameters:
## --------------------------

# Change to false to run the machine in headless mode
# (i. e. no VBox Window will be open)
GUI = true

# Audio device, or none for servers
AUDIO = "none"

# Turns on/off 3D acceleration. This does not behave too well on some Linux
# distros if turned on right now (at least in VirtualBox)
VIDEO_3D = "on"

# VirtualBox remote screen
VRDE = "off"

## Provisioning parameters:
## ------------------------

# Type of provisioner. I only use shell provisioners to not requiring installing
# more software. But this makes it easy to incorporate Ansible, Puppet,
# Chef or whatever automation you like!
PROVISION_TYPE = "shell"

# Provisioner file extension
VISION_FILE_EXTENSION = ".sh"
```

# Proveedores de Virtualización

## Especificando opciones particulares para cada proveedor

# Proveedores de virtualización

José Manuel
Redondo López

- Los proveedores de virtualización a usar deben estar instalados en el host
- Las boxes usadas en las **Vagrantfile** deben ser compatibles con alguno de los instalados
  - Hay que tener mucho cuidado al seleccionarla en la Vagrant Cloud
- Cada proveedor de virtualización tiene sus propias opciones de configuración (objetos con propiedades de distinto nombre y utilidad)
- Para saber estas opciones debemos consultar su documentación
  - VirtualBox: https://www.vagrantup.com/docs/virtualbox/
  - VMWare: https://www.vagrantup.com/docs/vmware/
  - Docker: https://www.vagrantup.com/docs/docker/
  - Hyper-V: https://www.vagrantup.com/docs/hyperv/
  - Amazon AWS (proveedor personalizado): https://www.vagrantup.com/docs/plugins/providers.html

# PROVEEDORES DE VIRTUALIZACIÓN

José Manuel
Redondo López

- **Aunque Vagrant trabaje con muchos proveedores, siempre habrá uno por defecto, según los instalados en el host**
  - Vagrant los examina y elige uno (¡o el único disponible! ☺)
- **Si hay varios se pueden cambiar de varias formas**
  - Añadiendo `ENV['VAGRANT_DEFAULT_PROVIDER'] = "<nombre del proveedor>"` en la `Vagrantfile`
  - Usando la opción `--provider` al hacer `vagrant up`. Por ejemplo:
    - Usar VMWare Fusion: `vagrant up --provider=vmware_fusion`
    - Usar un proveedor en la nube: `vagrant up --provider=aws`
- **Tras `vagrant up`, el resto de los comandos que trabajarán con la MV arrancada se adaptarán al proveedor usado**
- **Más información:** https://www.vagrantup.com/docs/providers/basic_usage.html

# ¿ENV?

José Manuel
Redondo López

- **Sí, las Vagrantfiles admiten variables de entorno para controlar algunos aspectos de su ejecución**
  - Normalmente bastante avanzados y que no se usan en un despliegue normal
- **Podemos encontrar un listado de variables aquí**
  - https://www.vagrantup.com/docs/other/environmental-variables
- **Quizá la más popular es la que controla la funcionalidad de poder redimensionar el tamaño de los discos**
  - La box que usamos como base tiene un tamaño de disco por defecto asignado
    - Que puede ser excesivo o demasiado pequeño...
  - Gracias a una variable de entorno podemos modificarlo

```
# This gives us the ability of resizing disks from the size put in
# the Vagrant Box. Some OS may force you to use gparted or similar
# to really take advantage of the extra space you put here.
ENV['VAGRANT_EXPERIMENTAL'] = 'disks'
```

CompleteVagrantfile

```
config.vm.disk :disk, size: DISK_SIZE, primary: true
```

**Permite usar esta opción**

# CONFIGURACIÓN POR PROVEEDOR DE VIRTUALIZACIÓN

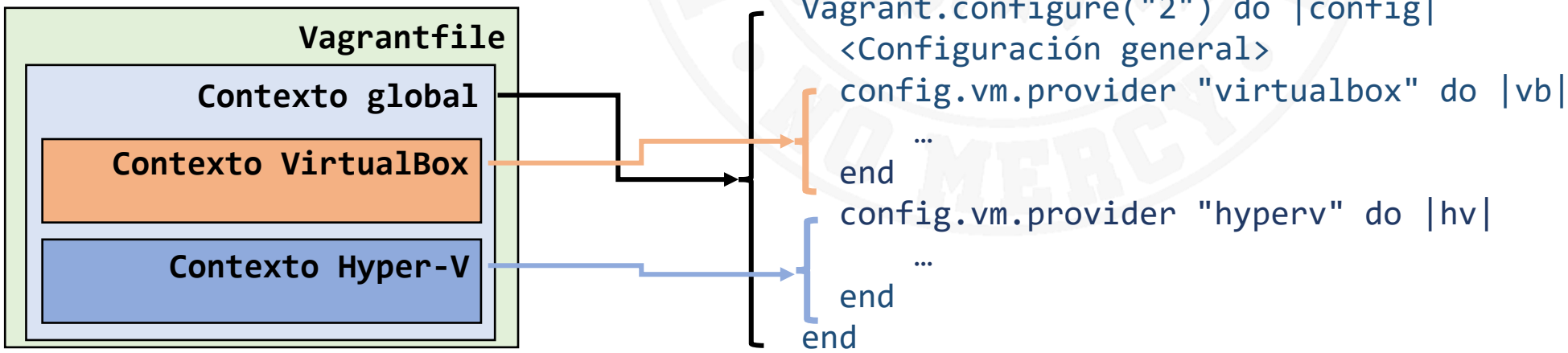
José Manuel
Redondo López

- **En cuanto queramos personalizar lo más mínimo de nuestra MV, debemos usar la configuración por proveedor**
  - No es posible tener una configuración unificada para todos los proveedores de virtualización utilizables por una MV Vagrant
- **Las opciones específicas de cada proveedor permiten**
  - Sacar partido a características particulares/avanzadas de un proveedor (**más optimización**)
  - Poder modificar más características de la máquina con los objetos y propiedades particulares que usa cada proveedor (**más control**)
  - Configurar detalles concretos de una plataforma de virtualización y cómo está construida (**más personalización**)
- **La configuración de Vagrant es portable**
  - Si se copia una `Vagrantfile` a un host que no tenga cierto proveedor, toda la configuración asociada al mismo se ignora

# CONFIGURACIÓN POR PROVEEDOR DE VIRTUALIZACIÓN

José Manuel
Redondo López

- Crear una sección de configuración concreta de un proveedor necesita abrir un contexto con su nombre y su variable correspondiente
  - Ej.: `config.vm.provider "virtualbox" do |vb|`
  - **Una por cada proveedor distinto que tengamos**
- Hecho esto, consultaremos los atributos disponibles en el objeto correspondiente a ese proveedor en su documentación



# CONFIGURACIÓN POR PROVEEDOR DE VIRTUALIZACIÓN

José Manuel
Redondo López

- Los proveedores de virtualización pueden sobrescribir (`override`) partes de la configuración global de **Vagrant**
- Esto se hace añadiendo la palabra `override` al bloque de configuración del proveedor específico. Ej.:

```
Vagrant.configure("2") do |config|
  config.vm.box = “ubuntu18_04_LTS"
  …
  config.vm.provider "vmware_fusion" do |v, override|
    override.vm.box = " ubuntu18_04_LTS_VMWare"
  end
end
```

- Esta configuración indica que todos los proveedores usarán la **box** `ubuntu18_04_LTS`
  - Menos **VMWare Fusion**, que usará una versión especial preparada para él (`ubuntu18_04_LTS_VMWare`)
  - No obstante, lo ideal es usar la misma **box** para diferentes proveedores (en la medida de lo posible)

# USANDO LAS CARACTERÍSTICAS DE CADA PROVEEDOR

José Manuel
Redondo López

```
config.vm.provider "virtualbox" do |vb|
  vb.name = MACHINE_NAME
  vb.memory = RAM
  vb.cpus = CORES
  vb.gui = GUI

  # Base configuration
  vb.customize ["modifyvm", :id, "--audio", AUDIO]
  vb.customize ["modifyvm", :id, "--clipboard-mode", "bidirectional"]
  vb.customize ["modifyvm", :id, "--description", MACHINE_DESCRIPTION]
  vb.customize ["modifyvm", :id, "--ostype", OS_TYPE]

  # High-performance configuration: please note that several of these options are not
  # available in the VirtualBox GUI. Thus, the performance of this machine should be higher
  # than the ones created via GUI

  # Use VBoxManage to customize the VM for high-performance options and other things
  vb.customize ["modifyvm", :id, "--acpi", "on"]
  vb.customize ["modifyvm", :id, "--accelerate3d", VIDEO_3D]
  vb.customize ["modifyvm", :id, "--apic", "on"]
  vb.customize ["modifyvm", :id, "--biosapic", "x2apic"]
  vb.customize ["modifyvm", :id, "--cpu-profile", "host"]
  vb.customize ["modifyvm", :id, "--graphicscontroller", "vmsvga"]
  vb.customize ["modifyvm", :id, "--hpet", "on"]
  vb.customize ["modifyvm", :id, "--hwvirtex", "on"]
  vb.customize ["modifyvm", :id, "--largepages", "on"]
  vb.customize ["modifyvm", :id, "--longmode", "on"]
  vb.customize ["modifyvm", :id, "--nestedpaging", "on"]
  vb.customize ["modifyvm", :id, "--pae", "on"]
  # If you have a MAC host and get an error, comment this line,
  # as this feature is not currently supported on MAC (yet)
  vb.customize ["modifyvm", :id, "--pagefusion", "on"]
  vb.customize ["modifyvm", :id, "--rtcuseutc", "on"]
  vb.customize ["modifyvm", :id, "--spec-ctrl", "off"]
  vb.customize ["modifyvm", :id, "--x2apic", "on"]
  vb.customize ["modifyvm", :id, "--vram", VRAM]
  vb.customize ["modifyvm", :id, "--vtxux", "on"]
  vb.customize ["modifyvm", :id, "--vtxvpid", "on"]
  vb.customize ["modifyvm", :id, "--vrde", VRDE]
  vb.customize ["storagectl", :id, "--name", HD_CONTROLLER, "--hostiocache", "on"]
```

Esta MV es 15-25% más rápida que la equivalente creada “a mano”

**CompleteVagrantfile**

- Como decíamos antes, así podemos “exprimir” cada proveedor de virtualización

```
config.vm.provider "hyperv" do |hv|
  hv.vmname = MACHINE_NAME
  hv.memory = RAM
  hv.cpus = CORES

  # Shared folders in Hyper-V are disabled because it uses the vulnerable SMBv1 protocol.
  # Additionally, the base box used may not support SMB
  config.vm.synced_folder ".", "/vagrant", disabled: true

  # This tries to give you the best possible Hyper-V user experience with the machine
  hv.enable_enhanced_session_mode = true
  hv.vm_integration_services = {
      guest_service_interface: true,
      heartbeat: true,
      key_value_pair_exchange: true,
      shutdown: true,
      time_synchronization: true,
      vss: true,
  }
end
```

Cambia el kernel por uno optimizado y activa las extensiones de Hyper-V

```
# VMWare Desktop very basic configuration
config.vm.provider "vmware_desktop" do |vmw|
  # If you need to update this, search for "VMX file parameters"
  vmw.gui = GUI
  vmw.vmx["memsize"] = RAM
  vmw.vmx["numvcpus"] = CORES
  vmw.vmx["displayName"] = MACHINE_NAME
  vmw.vmx["svga.autodetect"] = "TRUE"
  vmw.vmx["sound.present"] = "FALSE"
end
```

# ¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

?

## Boxes

- *¿Entiendes el concepto de **base box** y para qué se usa?*
- *¿Sabes cómo usar una base box de Vagrant Cloud y una que tengas tú creada en una **Vagrantfile**?*
- *¿Entiendes que las **boxes** son inmutables y su convenito de nombrado?*
- *¿Has comprendido que si no usas una **box** oficial puedes instalar **malware**, porque en la Vagrant Cloud no hay un control real de lo que se sube?*
- *¿Entiendes la forma de crear una box a partir de una que esté en funcionamiento?*
- *¿Entiendes la forma de crear una box vía código a partir de una de **Vagrant Cloud** y cómo personalizarla?*
- *¿Sabes usar variables en una **Vagrantfile**?*

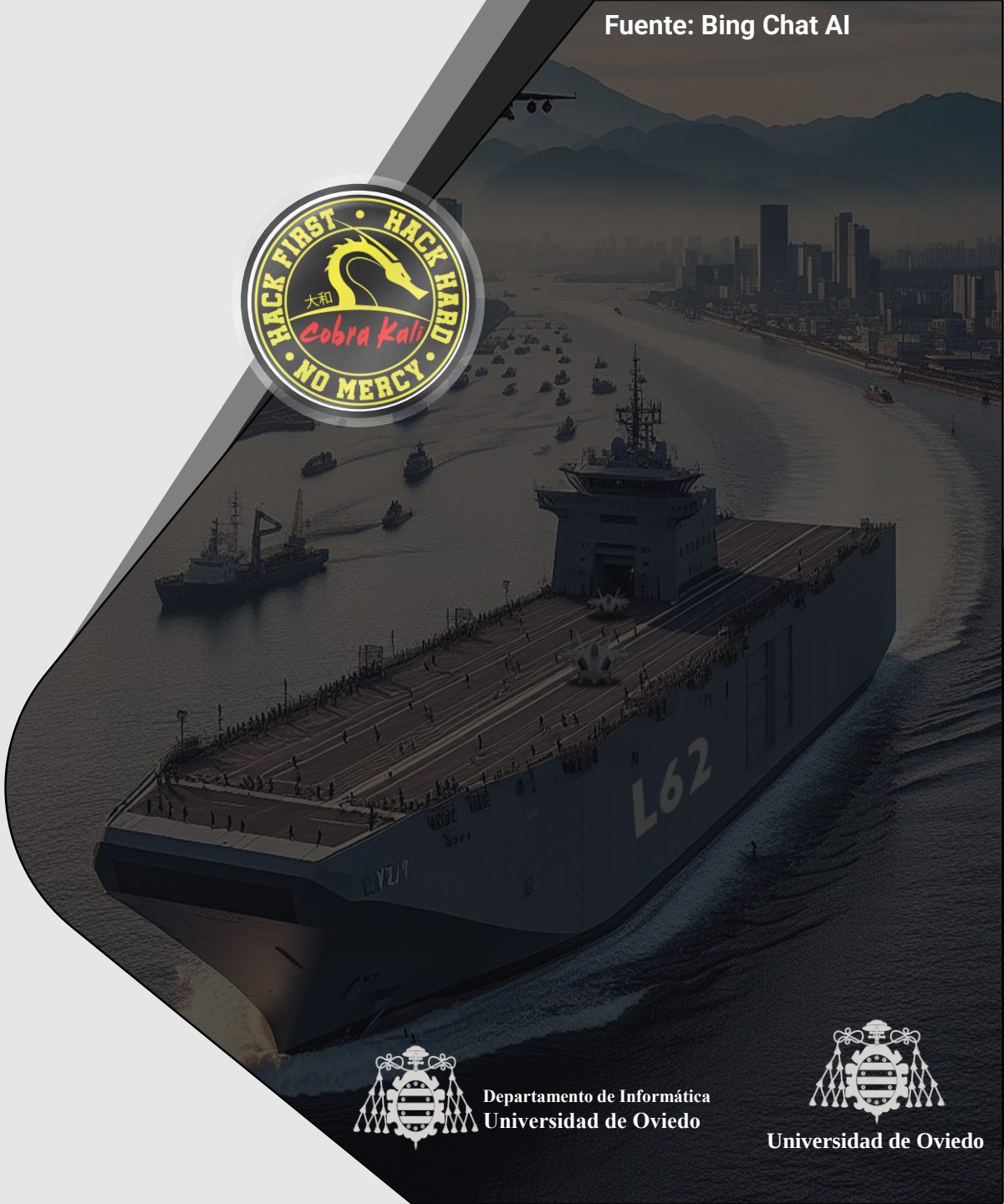
## Proveedores

- *¿Has entendido que es necesario que haya una compatibilidad entre la **box** que vayas a usar y el proveedor de virtualización que elijas?*
- *¿Entiendes que **Vagrant** te permite usar muchos proveedores de virtualización?*
- *¿Comprendes que las variables de entorno permiten personalizar Vagrant?*
- *¿Crees que puedes usar los diferentes contextos de configuración en una **Vagrantfile** y de esta forma hacer una configuración por cada uno de los proveedores que tengas?*
- *¿Entiendes que cada proveedor tiene su configuración particular y que, en general, es imposible hacer una configuración que se aplique para todos en cuanto queremos sacarle algo de partido a los mismos?*

< Ir al Índice

# Instalación de Ficheros y Programas en Máquinas Vagrant

## Los provisioners




Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

# ¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- **En este apartado vamos a explicar distintas funcionalidades de Vagrant que se usan para introducir ficheros a las MV creadas e instarles el software necesario para que cumplan su función**
  - La **gestión de carpetas compartidas** con el host para introducir ficheros en la MV
  - El **uso de provisioners** que permitan instalar todo el software necesario en la MV
  - La forma **de invocar provisioners** en ficheros externos y trucos para introducir en la MV los ficheros necesarios para que funcionen
  - Como conseguir que haya provisioners que instalen ciertas cosas **según el proveedor de virtualización en uso**
  - Entender que Vagrant está hecho en Ruby y que las Vagrantfiles **admiten código en ese lenguaje**
  - Que Vagrant soporta **muchos tipos de provisioners distintos**

# CARPETAS COMPARTIDAS Y SINCRONIZADAS

José Manuel
Redondo López

- **Vagrant sincroniza automáticamente ciertos ficheros entre el host y la máquina virtual creada**
  - Al hacer `vagrant up` por defecto se creará dentro de la MV recién creada un directorio `/vagrant`
  - El contenido de este directorio está sincronizado bidireccionalmente entre la máquina y el host con el directorio que tiene la `Vagrantfile` que creó la máquina
  - El nombre del directorio y el destino del host se pueden cambiar en la Vagrantfile
- **Por tanto:**
  - Si la máquina virtual crea contenidos en ese directorio, el host los verá
  - Si el host escribe ficheros en ese directorio, la máquina virtual también los verá
  - **Por eso es importante separar máquinas en directorios diferentes**

# CARPETAS COMPARTIDAS Y SINCRONIZADAS

José Manuel
Redondo López

- **Se puede sincronizar cualquier directorio del host con cualquier carpeta de la MV**
  - Para ello, hay que añadir al entorno de configuración general tantas entradas `config.vm.synced_folder “<ruta en el host>”, “<ruta en la MV>”` como queramos
  - Ejemplo:

```
Vagrant.configure("2") do |config|
  config.vm.synced_folder “fuentes/", "/var/www/html"
end
```

- **El primer parámetro puede ser una ruta absoluta o relativa del host**
  - **DEBE EXISTIR PREVIAMENTE** y, si es relativa, su origen es el directorio de la `Vagrantfile`
- **Si la ruta en la máquina virtual tiene directorios que no existen, se crean**
- **Con esta opción, podemos tener un contenido para las máquinas virtuales en rutas conocidas y que automáticamente lo incorporen al crearse**
  - Y dicho contenido quedará automáticamente actualizado cuando se hagan cambios en el host
  - **Más información**: https://www.vagrantup.com/docs/synced-folders/basic_usage.html

# INSTALACIÓN DE SOFTWARE

José Manuel
Redondo López

- **Una vez tenemos claro cómo incorporar ficheros a las MV, es necesario saber cómo incorporar programas**
  - Por ejemplo, las **extensiones de virtualización de la MV**
  - Esta operación se hace habitualmente con los denominados **provisioners**
  - Suelen ser productos que instalan automáticamente software como Ansible (visto más adelante en este curso), o scripts de bash
  - Puede haber muchos provisioners, pero **siempre deben estar en el contexto de configuración global**
  - **No podemos poner uno por proveedor, PORQUE SE EJECUTARÁN TODOS ☹**

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"
  config.vm.provision :shell, path: "inicio.sh"
end
```

```
#!/usr/bin/env bash
apt-get update
apt-get install -y apache2
if ! [ -L /var/www/html ]; then
  rm -rf /var/www/html
  ln -fs /vagrant /var/www/html
fi
```

**El siguiente ejemplo instala el paquete apache2 y sincroniza su directorio de webs por defecto con el directorio /vagrant (sincronizado con el host)**

# Ejecución de Provisioners

José Manuel
Redondo López

- **Se ejecutan en tres casos**
  - Si se les añade la opción `run: "always"`, se ejecutan en cada `vagrant up`
  - Si no usan esa opción
    - Al ejecutar el 1er `vagrant up` de una máquina, o el 1er `vagrant up` tras un `vagrant destroy`
    - Al ejecutar `vagrant provision`
    - Al ejecutar `vagrant reload --provision`
- **Podemos tener varias líneas**

  `config.vm.provision “<nombre provisioner>”`
  - El flag `--provision-with` ejecuta uno concreto por su nombre
- **Es mejor escribir el provisioning en ficheros aparte**
  - Y no incluirlos en línea en la Vagrantfile

```
==> default: Reading state information...
==> default: The following extra packages will be installed:
==> default:   apache2-mpm-worker apache2-utils apache2.2-bin apache2.2-common libapr1
==> default:   libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap ssl-cert
==> default: Suggested packages:
==> default:   www-browser apache2-doc apache2-suexec apache2-suexec-custom
==> default:   openssl-blacklist
==> default: The following NEW packages will be installed:
==> default:   apache2 apache2-mpm-worker apache2-utils apache2.2-bin apache2.2-common
==> default:   libapr1 libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap ssl-cert
==> default: 0 upgraded, 10 newly installed, 0 to remove and 189 not upgraded.
==> default: Need to get 1,844 kB of archives.
==> default: After this operation, 5,321 kB of additional disk space will be used.
==> default: Get:1 http://us.archive.ubuntu.com/ubuntu/ precise/main libapr1 i386 1.4.6-1 [9
==> default: Get:2 http://us.archive.ubuntu.com/ubuntu/ precise/main libaprutil1 i386 1.3.12
[75.4 kB]
==> default: Get:3 http://us.archive.ubuntu.com/ubuntu/ precise/main libaprutil1-dbd-sqlite3
3.12+dfsg-3 [10.2 kB]
==> default: Get:4 http://us.archive.ubuntu.com/ubuntu/ precise/main libaprutil1-ldap i386 1
sg-3 [7,962 B]
==> default: Get:5 http://us.archive.ubuntu.com/ubuntu/ precise-updates/main apache2.2-bin i
22-1ubuntu1.11 [1,328 kB]
==> default: Get:6 http://us.archive.ubuntu.com/ubuntu/ precise-updates/main apache2-utils i
22-1ubuntu1.11 [89.2 kB]
==> default: Get:7 http://us.archive.ubuntu.com/ubuntu/ precise-updates/main apache2.2-commo
.2.22-1ubuntu1.11 [225 kB]
==> default: Get:8 http://us.archive.ubuntu.com/ubuntu/ precise-updates/main apache2-mpm-wor
 2.2.22-1ubuntu1.11 [2,290 B]
==> default: Get:9 http://us.archive.ubuntu.com/ubuntu/ precise-updates/main apache2 i386 2.
untu1.11 [1,498 B]
==> default: Get:10 http://us.archive.ubuntu.com/ubuntu/ precise-updates/main ssl-cert all 1
ntu0.1 [12.3 kB]
```

# PROVISIONERS

José Manuel
Redondo López

- Los **provisioner** se distinguen por su **tipo**, que se especifica como primer parámetro de la configuración
  - Los tipos disponibles se pueden ver aquí: https://www.vagrantup.com/docs/provisioning
- Además de la sintaxis anterior (`clave: valor`), se admite otra **basada en bloques de Ruby**, como la del siguiente ejemplo
  - Esta sintaxis es más clara y admite hacer llamadas a métodos (algo necesario para ciertos provisioners)

```
Vagrant.configure("2") do |config|
  …
  config.vm.provision "shell" do |s|
    s.inline = "echo hello"
  end
End
```

- Se puede dar un nombre a los provisioners para distinguirlos en el log de operaciones
  - `config.vm.provision “MiProvisionerDeShell", type: "shell" do |s|`

# PROVISIONERS Y FICHEROS: TRUCOS

José Manuel
Redondo López

- **Si tenemos en cuenta tres cosas**
  1. Toda MV Vagrant tiene un usuario interactivo `vagrant` para permitir que Vagrant la gestione
     - Es decir, la carpeta `/home/vagrant` siempre existirá
  2. El provider de tipo "`file`" puede copiar ficheros del host a la MV
  3. Si una ruta es relativa siempre comenzará en la carpeta donde esté la Vagrantfile
- **Podemos usar este provider para mover ficheros a un sitio conocido de la MV y que otros provisioners los usen**
  - Un provisioner podrá así usar ficheros con datos o llamar a scripts, al estar en sitios conocidos
  - Una vez terminado el aprovisionamiento, se pueden borrar fácilmente
- **Ej. Mover ficheros y scripts a ejecutar dentro de la MV**

```
# Copy the files required by the Box provisioner
config.vm.provision "file", source: "../../../customization_files/" + \
  OS_CLASS + "_customization/", destination: "/home/vagrant/custom_files"
config.vm.provision "file", source: "../../../customization_scripts/" + \
  OS_CLASS + "_customization/", destination: "/home/vagrant/custom_scripts"
```

`CompleteVagrantfile`

# El problema de los provisioners por proveedor

José Manuel
Redondo López

- **Como dijimos, no se pueden hacer provisioners sólo para un proveedor de virtualización concreto**
  - Aunque pongamos provisioners en el contexto de varios provider concretos, **se ejecutarán todos**
- **¿Entonces cómo puedo instalar cosas sólo para ciertos proveedores de virtualización?**
  - Por ejemplo, las Guest Additions particulares de cada proveedor
- **Tendremos que usar las facilidades de nuestro provisioner para saber en qué tipo de proveedor de virtualización se ejecuta la MV**
  - En bash bajo Linux podemos usar el paquete `virt-what` (instalándolo previamente en la MV). Ej.:

```
if [[ $(sudo virt-what | grep 'virtualbox') = *virtualbox* ]]; then
… (paquetes a instalar en VirtualBox)
elif [[ $(sudo virt-what | grep 'hyperv') = *hyperv* ]]; then
… (paquetes a instalar en Hyper-V)
```

# Ejemplo de provisioner de shell

José Manuel
Redondo López

- **Es importante tener en cuenta que Vagrant intentará arrancar la máquina lo antes posible para que podamos trabajar con ella**
  - Esto implica que es muy probable que el trabajo del provisioner **no haya acabado** cuando tengamos acceso a ella por primera vez
  - Por tanto, nos faltará software, configuración, etc.
- **Es importante esperar a que el provisioner acabe**
  - Para lo cual **hacer un `reboot` al final es clave**, no solo por cargar todos los cambios o actualizaciones sino para saber cuándo lo ha hecho
  - Se puede conseguir añadiendo `reboot: true` a los parámetros del provisioner
- **Recuerda: una vez ejecutado el provisioner, tenemos que forzar que se ejecute de nuevo, y no lo hará en los siguientes arranques**
  - Por tanto, el siguiente arranque de la VM será mucho más rápido

```
# Automatically install software
config.vm.provision "Software Installation", type: PROVISION_TYPE, \
  path: "provisioners/" + PROVISION_TYPE + "/provision" + PROVISION_FILE_EXTENSION, \
  reboot: true
```

`CompleteVagrantfile`

# RUBY EN LAS VAGRANTFILES

José Manuel
Redondo López

- **Como dijimos al principio, las Vagrantfiles usan sintaxis Ruby**
- **Eso significa que podemos usar estructuras de control de este lenguaje para hacer determinadas operaciones**
  - Decidir si se ejecuta o no un provisioner
  - Decidir qué provisioner es mejor según los datos obtenidos
  - … (cualquier cosa que se nos ocurra)
- **Esto implica**
  - Poder usar objetos del API de Ruby
  - Definir ficheros `.rb` de código propio donde creemos nuestros objetos y funciones
    - Importables poniendo `require ‘<nombre del fichero>’` al principio de la Vagrantfile
  - Usar estructuras de control de Ruby, como `if-else`, `<colección>.each do |…|`
  - …

# RUBY EN LAS VAGRANTFILES

José Manuel
Redondo López

- **Este ejemplo muestra**
  - Que se puede usar `if-else` y otras estructuras de control en el contexto global o en otro
  - El uso del objeto `File` del API de Ruby
    - https://ruby-doc.org/core-2.5.0/File.html
  - El uso de un objeto especial del API de Ruby llamado Vagrant, con funciones útiles
    - https://www.rubydoc.info/github/mitchellh/vagrant/Vagrant
    - Permite trabajar con características del host
  - El uso de `require` para importar módulos externos en una Vagrantfile
  - Cómo **mezclar** instrucciones de una Vagrantfile con código Vagrant

```
require ‘yaml’
…
if File.exists?(config_file)
  sites = YAML.load_file(config_file)['sites']
  …
else
  …
end

Vagrant.configure('2') do |config|
  …
  if Vagrant::Util::Platform.windows?
    sites.each do |(name, site)|
      config.vm.synced_folder local_site_path(site),
remote_site_path(name), owner: 'vagrant', group: 'www-
data', mount_options: ['dmode=776', 'fmode=775']
    end
  else
    …
  end
end
```

# Tipos de Provisioners

José Manuel
Redondo López

- **Debemos siempre usar los provisioners más adecuados a nuestro entorno (nube, on-premises…) o lo que tengamos instalado**
  - `File` (ya visto): https://www.vagrantup.com/docs/provisioning/file.html
  - `Shell` (ya visto): https://www.vagrantup.com/docs/provisioning/shell.html
  - `Ansible`: https://www.vagrantup.com/docs/provisioning/ansible.html (visto en este curso más adelante)
    - `Ansible Local`: https://www.vagrantup.com/docs/provisioning/ansible_local
  - `CFEngine`: https://www.vagrantup.com/docs/provisioning/cfengine.html
  - `Chef`
    - `Chef Solo`: https://www.vagrantup.com/docs/provisioning/chef_solo.html
    - `Chef Zero`: https://www.vagrantup.com/docs/provisioning/chef_zero.html
    - `Chef Client`: https://www.vagrantup.com/docs/provisioning/chef_client.html
    - `Chef Apply`: https://www.vagrantup.com/docs/provisioning/chef_apply.html
  - `Docker`: https://www.vagrantup.com/docs/provisioning/docker.html
  - `Puppet`
    - `Puppet Apply`: https://www.vagrantup.com/docs/provisioning/puppet_apply.html
    - `Puppet Agent` (Master/Agent): https://www.vagrantup.com/docs/provisioning/puppet_agent.html
  - `Salt`: https://www.vagrantup.com/docs/provisioning/salt.html

# ¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

- *¿Entiende que, por defecto, sólo por arrancar una MV ya tienes una carpeta compartida con el host y cuál es?*
- *¿Entiendes que dicha carpeta compartida es bidireccional?*
- *¿Sabes sincronizar cualquier directorio del host con una máquina virtual Vagrant?*
- *¿Sabes cómo usar un script de bash típico para instalar software?*
- *¿Entiendes en qué casos se ejecutan y no los provisioners de una Vagrantfile?*
- *¿Puedes distinguir entre las dos sintaxis admitidas por una Vagrantfile?*
- *¿Sabes cómo proporcionar ficheros en una ruta conocida de la MV a un provisioner?*
- *¿Sabes cómo distinguir dentro de un provisioner que sistema de virtualización está usando una MV?*
- *¿Entiendes que puedes usar Ruby en una Vagrantfile y la flexibilidad que te da esto?*
- *¿Comprendes que Vagrant puede trabajar muchos provisioners distintos y las ventajas que eso tiene?*

< Ir al Índice

Fuente: Bing Chat AI

Multi-máquina >

Redes >

Discos >

# Infraestructuras en Vagrant

## Creando MVs conectadas entre sí


Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

# ¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- **En este apartado vamos a ver una serie de características de Vagrant que facilitan la creación de infraestructuras**
  - Cómo **crear varias MV** en una misma Vagrantfile (entornos multi-máquina)
    - Cómo se **provisionan máquinas** en un entorno multi-máquina
    - Cómo se **controlan máquinas** individuales en un entorno multi-máquina
  - Cómo definir **redes** para máquinas y mapear puertos de las mismas
    - Los **distintos tipos de redes** existentes en Vagrant
  - Cómo definir **nuevos discos** para máquinas virtuales o **cambiar el tamaño** de los existentes

# Entornos Múlti-Máquina

## Una Vagrantfile para definirlas a todas (las MV)


# PROVISIONERS EN ENTORNOS MULTI-MÁQUINA

José Manuel
Redondo López

- **Se pueden definir varias MV en una misma Vagrantfile**
  - Se le llama multi-machine environment
  - Es típico de máquinas que deban trabajar juntas o estén asociadas de alguna forma
    - Ej.: servidor web y su base de datos
- **Es mucho mejor separar las funciones en distintas máquinas que concentrar todas en una sola**
- **Esta característica permite modelar topologías mucho más complejas y automatizar infraestructuras más avanzadas**
  - Por lo que es necesario saber cómo se hace
  - La gran ventaja es que nos vale un solo fichero para eso
  - **Y que cada máquina puede tener sus propios provisioners, instalando solamente aquello que necesitan**

# ENTORNOS MULTI-MÁQUINA

José Manuel
Redondo López

- **Para definir varias máquinas en la misma Vagrantfile se usa el método** `config.vm.define`
  - Esto crea una **configuración secundaria** dentro de la principal
- **Las máquinas pueden tener la misma imagen base o diferente. Ej.:**

```
vagrant.configure("2") do |config|
  ...
  config.vm.define "web" do |web|
    web.vm.box = "miempresa/mi_servidor_web"
  end
  config.vm.define "db" do |db|
    db.vm.box = "miempresa/mi_servidor_mysql"
  end
end
```

# ENTORNOS MULTI-MÁQUINA

José Manuel
Redondo López

- El ejemplo muestra 2 MV construidas con la misma **box** `ubuntu/focal64`
- Ambas comparten características, y solo definen un **provisioner** propio y una IP fija en una red privada
  - Dentro de una configuración secundaria **SÍ** se puede definir un **provisioner** propio (no como antes)
- Al final de la **Vagrantfile** ambas ejecutan un **provisioner** para limpiar y actualizar la caché de paquetes
- Esto muestra cómo pueden combinarse operaciones globales con particulares de cada MV

```
vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/focal64"
  config.vm.provider "virtualbox" do | vbox |
     vbox.memory = 1024
  end
  config.vm.define "web" do |web|
     web.vm.network "private network", ip: "192.168.20.10"
     … // Provisioners para esta máquina solamente
  end
  config.vm.define "db" do |db|
     db.vm.network "private network", ip: "192.168.20.20"
     … // Provisioners para esta máquina solamente
  end
  config.vm.provision "shell", inline: <<- SHELL
     apt-get clean
     apt-update –y
  SHELL
end
```

# ENTORNOS MULTI-MÁQUINA

José Manuel
Redondo López

- **Cada `config.vm.define` crea un bloque definido por una variable**
  - Esta variable se usa igual que la `config` global anterior, pero en un contexto local
  - Ej.: Todo lo que se asocie a la variable `web` afectará solo a la máquina web
- **Así se pueden usar variables que afecten solo a máquinas concretas**
  - Y `config` se usará para configuraciones que afecten a todas las máquinas
- **Afecta al orden de ejecución de los provisioners**
  - El orden es de fuera a dentro
  - **Es decir, un provisioner en un contexto global se ejecutará siempre antes que uno en uno local**
  - `vagrant provision <nombre de máquina>` ejecuta solo el provisioner de la máquina dada

```
Vagrant.configure("2") do |config|
  config.vm.provision "shell", inline:
"echo global“

  config.vm.define "web" do |web|
    web.vm.provision "shell", inline:
"echo local1"
  end

  config.vm.provision "shell", inline:
"echo global2"
end
```

**Esto imprime “global”, “global2”, “local1” ¡es un sistema muy flexible!**

# Ejecución de Provisioners en entornos multi-máquina

José Manuel
Redondo López

- **Los provisioners más locales pueden sobrescribir la configuración de otros globales**
  - Útil para tener unas acciones del provisioner aplicables a todas las MV
  - Combinadas con otras que se apliquen a una concreta
- **Para ello, los provisioners deben tener el mismo nombre**
  - Este ejemplo imprime "Configurando web" solo para la máquina llamada "web"
  - Y "Configurando máquina" para el resto
- **Aparte, cada máquina puede definir sus propios interfaces de red, parámetros, etc.**

```
Vagrant.configure("2") do |config|
  config.vm.provision "miprovisioner",
type: "shell",
    inline: "echo Configurando máquina“

  config.vm.define "web" do |web|
    web.vm.provision "miprovisioner",
type: "shell",
      inline: "echo Configurando web"
  end
end
```

# CONTROLANDO VARIAS MÁQUINAS

José Manuel
Redondo López

- **Si se definen varias máquinas en una misma Vagrantfile, muchos comandos necesitarán especificar el nombre de la máquina sobre el que van a operar**
  - `vagrant ssh`, necesitará un nombre de máquina: `vagrant ssh web` o `vagrant ssh db`
  - Se puede especificar una **máquina primaria**
    - Será la que se use por defecto en los comandos que necesiten un nombre de máquina en un entorno multi-máquina
  - Solo puede haber una máquina primaria
    - `config.vm.define "web", primary: true do |web|  # ...end`
- **Otros comandos, como `vagrant up` trabajan con todas las máquinas por defecto**
  - Ejecutar un `vagrant up` arrancará tanto la máquina `web` como la `db`
  - No obstante, también admiten nombres si solo queremos arrancar una de ellas

```
Vagrant.configure("2") do |config|
  config.vm.disk :disk, size: DISK_SIZE, primary: true
  … (Resto de configuración común a todas las MV)
  config.vm.provider "virtualbox" do |vb|
    vb.memory = RAM
    vb.cpus = CORES
    … (Resto de configuración común a todas las MV que usen VirtualBox)
  end
  config.vm.define "main" do |main|
    main.vm.box = “Maquina 1”
    main.vm.provision "Software Installation", type: … (Provisioner para esta VM)
    main.vm.network "private_network", ip: "192.168.56.10", adapter: 2
    main.vm.network "private_network", ip: "192.168.1.10", adapter: 3, virtualbox__intnet: "milan"
    main.vm.provider "virtualbox" do |vb1|
        vb1. … (Configuración particular de VirtualBox para esta máquina)
    end
  end
  config.vm.define “secondary" do | secondary |
    secondary.vm.box = “Maquina 1”
    secondary.vm.provision "Software Installation", type: … (Provisioner para esta MV)
    secondary.vm.network "private_network", ip: "192.168.1.10", adapter: 3, virtualbox__intnet: "milan“
    secondary.vm.provider "virtualbox" do |vb2|
        vb2. … (Configuración particular de VirtualBox para esta máquina)
    end
  end
end
```

# Redes en Vagrant

## Conectando máquinas virtuales

# MAPEO DE PUERTOS

José Manuel
Redondo López

- **Hay varias formas de asignar una dirección IP a una MV creada con Vagrant para que sea visible desde el exterior de su host**
  - La primera y más sencilla es el **mapeo de puertos**
  - Podemos asignar cualquier puerto de la máquina creada a un puerto del host
  - En el siguiente ejemplo un acceso al puerto indicado del host (8100) accederá realmente al servicio de la máquina que queramos (80) con un tipo de red especial: "forwarded_port"

```
vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"
  config.vm.provision :shell, path: "inicio.sh"
  config.vm.network "forwarded_port", guest: 80, host: 8100
end
```

- **Hay que hacer `vagrant reload` si cambiamos una `Vagrantfile` de una MV ya creada**
- **Más información:** https://www.vagrantup.com/docs/networking/forwarded_ports.html

# REDES PÚBLICAS

José Manuel
Redondo López

- **Se definen cuando queremos que alguna máquina externa acceda a la que hemos creado, con una IP conocida**
  - Es equivalente a la red de tipo **Bridge** de **VirtualBox**
  - La MV tendrá una IP asignada en la misma red que su host, siendo visible para otras máquinas en la misma red
- **Deben usarse con cuidado, puesto que las box de Vagrant configuradas por defecto podrían ser inseguras**
  - Requiere **hardening** del SO y sus servicios, como el que se verá en "**F-103 Blas de Lezo**" 
- **Es posible asignar una IP concreta a una red pública o bien usar DHCP**
- **Se configuran con** `config.vm.network :public_network` **en la** `Vagrantfile`
  - Si la MV tiene varios adaptadores, se debe elegir uno y la interfaz del host que hará de **bridge**
  - Determina en qué red se asignará una IP a la MV
  - Ej.: `config.vm.network "public_network", adapter: 3, bridge: "wlp3s0"`
- **Más información:** https://www.vagrantup.com/docs/networking/public_network.html

# Redes públicas

José Manuel
Redondo López

- Si esta configuración no funcionase, lo haremos a mano añadiendo `auto_config: false` a la configuración
- El siguiente ejemplo asigna la IP `192.168.23.80` a la máquina
  - La máquina host tiene un interfaz (`eth1`) con una IP en la red `192.168.23.0/24`
  - Al arrancar, Vagrant nos preguntará qué interfaz queremos usar, y tenemos que seleccionar `eth1` por ese motivo
    - También podemos especificarlo en la `Vagrantfile`
  - Ahora cualquiera podrá acceder al servidor web a través de la IP `192.168.23.80`
- **NOTA**: Debemos asegurarnos de que la box creada tiene las interfaces de red necesarias creadas de antemano

```
config.vm.network :public_network, auto_config: false

config.vm.provision "shell",
      run: "always",
      inline: "ifconfig eth1 192.168.23.80 netmask 255.255.255.0 up"
```

# REDES PRIVADAS

José Manuel
Redondo López

- **Se crean cuando el acceso a las máquinas Vagrant desde el exterior debe estar terminantemente prohibido**
  - La idea es asignar a las máquinas IPs inaccesibles por ninguna máquina que no esté en el host
  - Todas las máquinas que tengan una IP privada en la misma red podrán comunicarse entre ellas
- **Se permite de nuevo la asignación usando DHCP o manual**
  - Manualmente sería: `config.vm.network :private_network, ip: “192.168.5.5”`
  - Si se asigna una IP manualmente, esta no puede ser de una red existente ya en el host
  - Podemos elegir el adaptador de la MV: `config.vm.network "private_network", ip: "192.168.56.10", adapter: 2`
  - No obstante, en VirtualBox esto crea una red privada de tipo host-only
  - Para crear una red interna con nombre (“`milan`” en el ejemplo) hay que hacer la siguiente variante:
    - `main.vm.network "private_network", ip: "192.168.1.10", adapter: 3, virtualbox__intnet: "milan"`
  - **Más información**: https://www.vagrantup.com/docs/networking/private_network.html

# Configuración por proveedor

José Manuel
Redondo López

- Con estos dos tipos de redes podemos hacer nuestra SNI del primer tema
  - Un proxy inverso en una MV Vagrant será **lo único accesible públicamente** (public network)
  - Mientras frontend, backend y sus BD se comunicarán entre sí con mediante **redes privadas**
  - Pudiendo así crear la red en varias capas que mencionamos al principio de la asignatura
- Aunque Vagrant intenta no requerir conocimientos detallados de particularidades de cada proveedor, no siempre es posible
- Por tanto, en ocasiones deberemos configurar aspectos de las redes que son exclusivos de proveedores concretos
- En este enlace hay un ejemplo de cómo hacerlo con VirtualBox
  - https://www.vagrantup.com/docs/virtualbox/networking.html
- Se recomienda consultar las opciones existentes con otros proveedores en: https://www.vagrantup.com/docs/providers/

# Discos en Vagrant

## Dotando a las máquinas virtuales de nuevas unidades de disco

# NUEVOS DISCOS PARA MÁQUINAS VIRTUALES

José Manuel
Redondo López

- Aunque todas las **boxes** de base que podemos usar vienen con un disco, es muy probable que queramos usar más
- Se pueden añadir una **box** nuevos discos de distintos tipos
  - Requiere el uso de una opción experimental en la `Vagrantfile`
    - `ENV['VAGRANT_EXPERIMENTAL'] = 'disks'`
  - Fundamentalmente discos de lectura/escritura (`disk`) o de solo lectura (`dvd`)
    - Un tipo `disk` se puede convertir en primario con la opción `primary`
    - Los de tipo DVD necesitan obligatoriamente **un valor en su parámetro** `file:`
- Pueden consultarse las opciones disponibles aquí
  - https://www.vagrantup.com/docs/disks/configuration
  - Algunos proveedores podrían tener opciones particulares

```
config.vm.disk :disk, name: "backup_disk", size: “40GB"
config.vm.disk :dvd, name: "install", file: "./visual_studio.iso"
```

# ¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López


## Entornos Multi-máquina

- *¿Entiendes cómo puedes usar una misma Vagranfile para definir múltiples?*
- *¿Tienes claro cómo construir una Vagranfile de manera que se aprovechen configuraciones comunes para todas las máquinas y luego puedas definir las particulares de cada una de ellas?*
- *¿Entiendes que cada máquina puede tener sus propios provisioners y variables independientes?*
- *¿Entiendes también que puede haber provisioners y variables comunes a todas las máquinas y para qué pueden usarse?*
- *¿Sabes cómo ejecutar Vagrant sobre una máquina particular y sobre todas las definidas?*

## Redes

- *¿Entiendes cómo mapear puertos de una MV al host?*
- *¿Sabes qué es una red pública de Vagrant y sus características?*
- *¿Sabes qué es una red privada de Vagrant y sus características?*
- *¿Entiendes que cada proveedor puede definir tipos de redes nuevos particulares?*

## Discos

- *¿Sabes qué necesitas añadir para redimensionar un disco?*
- *¿Entiendes cómo añadir un disco a una MV?*

< Ir al Índice

Fuente: Bing Chat AI

Plugins >

Optimización >

Preparación boxes >

Packer >

# Opciones Avanzadas

## Ampliando Vagrant


Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

# ¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- **En este apartado vamos a ver una serie de características avanzadas de Vagrant que permiten explotar más su uso**
  - El **uso de plugins** para ampliar sus funcionalidades
  - Como **optimizar las boxes** creadas para que ocupen el menor número de recursos posible
  - Cómo **preparar boxes base** a nuestro gusto desde MV que tengamos, de las cuales partir posteriormente para realizar adaptaciones
  - De dónde sacar **boxes Windows** utilizables para pruebas
  - La función y usos de la **herramienta Packer**

# Plugins de Vagrant

## Vagrant es ampliable

# ¿Qué es un plugin de Vagrant?

José Manuel
Redondo López

- **Aunque Vagrant ya viene con un buen nº de características, es posible ampliarlas por medio de plugins**
  - Realmente, gran parte de Vagrant está hecho a base de plugins ☺
- **Podemos encontrar más información acerca del uso y desarrollo de plugins para Vagrant aquí**
  - https://www.vagrantup.com/docs/plugins/
- **Lista de los plugins disponibles, incluyendo gran cantidad de providers y provisioners**
  - https://github.com/hashicorp/vagrant/wiki/Available-Vagrant-Plugins

# Plugins de Vagrant: Proveedor de virtualización libvirt

José Manuel
Redondo López

- Como ejemplo de plugin, en este enlace tenemos la instalación del soporte para un nuevo proveedor de virtualización (KVM) en Vagrant
  - https://github.com/vagrant-libvirt/vagrant-libvirt#installation
- Por supuesto, hay que instalar el proveedor de virtualización en sí
  - https://help.ubuntu.com/community/KVM/Installation
- Una vez instalado, debemos tener en cuenta varias cosas:
  - Por necesitar acceso a partes del sistema restringidas, hay que ejecutar `vagrant up` con `sudo`
  - Si debido al entorno virtualizado de pruebas, el driver `kvm` de virtualización no funciona correctamente (no hay acceso a VT-X), es necesario cambiarlo a `qemu`

```
# Provider-specific configuration so you can fine-tune various
# backing providers for Vagrant. These expose provider-specific options.
# Example for VirtualBox:
#
config.vm.provider "libvirt" do |libvirt|
   libvirt.driver = "qemu"
   # Customize the amount of memory on the VM:
   libvirt.memory = "512"
end
#
```

# Optimización de boxes

## Sacando el mayor rendimiento posible a las MV generadas

# ¿Cómo disminuir el uso de recursos de las MV?

José Manuel
Redondo López

- **VirtualBox** es el proveedor preferido por **Vagrant** y tiene opciones que nos ahorran muchos recursos
  - **Headless**: Inicializando la opción `gui` a `false` la MV arranca sin interfaz gráfico visible en el host
    - Útil para servidores que arranquen un servicio que quede funcionando permanentemente (webs)
  - **Linked clone**: Por defecto Vagrant importa (clona) la box base de la Vagrant Cloud o local
    - Esto consume mucho espacio y tiempo, especialmente son boxes grandes
    - Un linked clone hace esto 1 vez, el resto crea un archivo con las diferencias a nivel de disco con la "padre"
    - Es decir, solo guarda lo nuevo que se añada a ella y para el resto referencia al padre
    - Esto puede ahorrar enormes cantidades de disco, pero si perdemos el acceso a la box padre, fallará
  - **Page Fusion**: Hace que varias MV compartan las páginas de memoria RAM iguales que tengan
    - Opción avanzada que disminuye mucho el consumo de RAM si tenemos un clúster de MV idénticas, por ejemplo

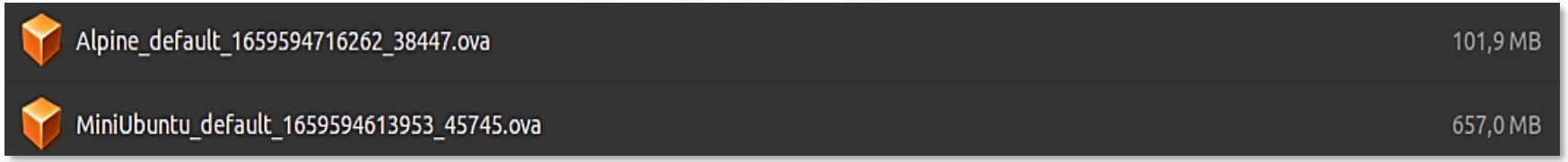
```
config.vm.provider "virtualbox" do |v|
        …
        v.linked_clone = true
        v.gui = true
        vb.customize ["modifyvm", :id, "--pagefusion", "on"]
        …
end
```

➢ Ejercicio **N5VAG_LINKEDCLON**

# ¿Cómo disminuir el uso de recursos de las MV?

José Manuel
Redondo López

- **Existen además una serie de estrategias para hacer que nuestras máquinas virtuales sean lo más pequeñas posible**
  - Facilitan moverlas entre **hosts** y disminuyen la cantidad de **software** en funcionamiento
  - Las hace menos vulnerables y que usen mejor los recursos
- **Las estrategias son**
  - **Usar la imagen base más pequeña posible**
    - En la actualidad la más recomendable es **Alpine Linux**
      - https://app.vagrantup.com/generic/boxes/alpine38
    - Es un SO **Linux** creado para ocupar lo mínimo posible
    - Tiene la mayoría de los paquetes que **Ubuntu/Debian**, pero con un gestor diferente:
      - https://www.cyberciti.biz/faq/10-alpine-linux-apk-command-examples/
    - En la imagen se ve un **Alpine Linux** estándar VS un **Ubuntu** previamente minimizado con el resto de las técnicas que vamos a ver en las siguientes páginas



# ¿Cómo disminuir el uso de recursos de las MV?

José Manuel
Redondo López

- **Convertir la distribución en mínima**
  - Una distribución mínima tiene **el nº menor posible de paquetes instalado**
    - Esto disminuye la superficie de ataque del SO
    - Menos paquetes que puedan tener vulnerabilidades
  - El metapaquete `ubuntu-minimal` contiene solo los paquetes considerados esenciales para tener un Ubuntu básico funcional
  - Se puede instalar automáticamente con
    - `DEBIAN_FRONTEND=noninteractive apt-get --assume-yes --no-install-recommends install aptitude ubuntu-minimal`
  - Luego es necesario desinstalar el resto de los paquetes existentes, salvo unos pocos necesarios para poder interactuar con Vagrant
    - `DEBIAN_FRONTEND=noninteractive aptitude --assume-yes markauto '~i!?name(ubuntu-minimal~|linux-generic~|systemd~|openssh-server|netplan|sudo)'`

# ¿Cómo disminuir el uso de recursos de las MV?

José Manuel
Redondo López

- **Eliminar paquetes no estrictamente esenciales**
  - Aunque tengamos una instalación mínima, hay una serie de paquetes que no son necesarios si solo vamos a ejecutar un servicio y nada más en la MV
  - Estos se desinstalan así
    - `DEBIAN_FRONTEND=noninteractive apt-get --assume-yes purge bzip2 eject iputils-ping less mawk netcat-openbsd ubuntu-advantage-tools vim-tiny rsyslog vim apport-symptoms bc bsdmainutils byobu crda dosfstools ed fonts-lato ftp geoip-database git git-man groff-base info install-info installation-report javascript-common manpages motd-news-config telnet tmux tcpdump unzip vim-common vim-runtime zerofree zip wireless-regdb strace sosreport rsync mdadm lsof usbutils xauth xxd htop mlocate wget nano screen xdelta3 xdg-user-dirs xfsprogs command-not-found command-not-found-data curl gpg gnupg-l10n gnupg-utils gpg-agent gpgconf plymouth libplymouth4`
  - Hay que hacer limpieza del gestor de paquetes luego
    - `DEBIAN_FRONTEND=noninteractive aptitude --assume-yes purge '~c'`

# ¿CÓMO DISMINUIR EL USO DE RECURSOS DE LAS MV?

José Manuel
Redondo López

- **Eliminar paquetes de notificación de errores e instalación superfluos**
  - Se puede hacer así
    - `DEBIAN_FRONTEND=noninteractive sudo apt-get purge -y --auto-remove snapd squashfs-tools friendly-recovery apport at`
  - Y luego
    - `sudo apt-get autoremove –purge`
- **Deshabilitar la telemetría** (notificación de errores a Canonical):
  - `sudo apt purge -y ubuntu-report popularity-contest apport whoopsie >/dev/null 2>&1`
  - `sudo ubuntu-report -f send no`
- **Eliminar ficheros de cache, documentación y otros no necesarios si no hay trabajo interactivo**
  - `sudo rm -rf /var/cache/apt/archives`
  - `sudo rm -rf /usr/share/doc/`
  - `sudo rm -rf /usr/share/man/`
  - `sudo rm -rf /usr/share/locale/`
  - `sudo rm /pkgcache.bin`
  - `sudo rm /srcpkgcache.bin`

# ¿Cómo disminuir el uso de recursos de las MV?

José Manuel
Redondo López

- **Reiniciar los logs**: Al fin y al cabo, si esta imagen va a servir de base para otras, no nos interesan los logs generados al construirla...
  - `rm -rf /var/log/journalcat /dev/null > /var/log/installer/cdebconf/questions.dat`
  - `cat /dev/null > /var/log/installer/cdebconf/templates.dat`
  - `cat /dev/null > /var/log/installer/syslog`
  - `cat /dev/null > /var/log/installer/status`
  - `cat /dev/null > /var/log/installer/partman`
  - `cat /dev/null > /var/log/alternatives.log`
  - `cat /dev/null > /var/log/auth.log`
  - `cat /dev/null > /var/log/bootstrap.log`
  - `cat /dev/null > /var/log/btmp`
  - `cat /dev/null > /var/log/dpkg.log`
  - `cat /dev/null > /var/log/faillog.logcat /dev/null > /var/log/kern.log`
  - `cat /dev/null > /var/log/lastlogcat /dev/null > /var/log/syslog`
  - `cat /dev/null > /var/log/tallylog`

# ¿Cómo disminuir el uso de recursos de las MV?

José Manuel
Redondo López

- **Eliminar las distintas cachés de instalación de paquetes**
  - En caso necesario el sistema volvería a rellenarlas
  - `rm -rf /var/cache`
  - `rm -rf /var/cache/man`
  - `rm -rf /var/lib/apt/lists/*`
  - `rm -rf /usr/share/vim`
  - `rm -rf /usr/share/GeoIP`
- **Eliminar posibles restos de instalación de paquetes**
  - `sudo rm -rf /*.deb`
  - `sudo rm -rf /*cache.bin`
- **Eliminar las fuentes del kernel** (si no se van a compilar más aplicaciones)
  - `sudo rm -rf /usr/src`

# ¿Cómo disminuir el uso de recursos de las MV?

José Manuel
Redondo López

- **Si vamos a exportar la máquina virtual a un `.ova`, podemos hacer que el tamaño del mismo sea mucho más pequeño si rellenamos de ceros el espacio libre**
  - Aunque se hayan borrado ficheros, **sus datos aún permanecen en el disco duro** a la espera de ser sobrescritos
  - Por ello, al compactar el disco el algoritmo de compresión los cuenta como datos útiles y el tamaño del disco es mayor de lo que debería
    - No sabe que han sido borrados, es un proceso externo
  - Esto no ocurre si lo rellenamos de ceros así
    - `sudo dd if=/dev/zero of=/var/tmp/bigemptyfile bs=4096k ; sudo rm /var/tmp/bigemptyfile`
- **Aún con eso, una imagen de Ubuntu pasa de ocupar 1,2Gb a unos 650Mb**
  - Mucho mayor que un Alpine Linux estándar
  - No obstante, si necesitamos exportar un Linux tipo Debian o Red Hat por algún motivo, estos consejos disminuyen dramáticamente el espacio que ocupan

# Preparación de boxes base y boxes Windows

## Más allá de Linux

# ¿Cómo preparar nuestras boxes base?

José Manuel
Redondo López

- **Hasta ahora hemos usado boxes base desde la Vagrant Cloud**
  - Pero estas boxes realmente son máquinas "normales" que alguien preparó especialmente para ser usadas con Vagrant
- **Podemos crear nuestras propias boxes base a partir de MV que tengamos**
  - Esto requiere **cambiar su configuración de una forma especial**, aunque depende del proveedor de virtualización
  - Y **tratar de minimizarla** con las instrucciones vistas para manejarla más fácilmente
- **Para VirtualBox sería**
  - Configurar el primer interfaz como NAT, y apuntar su MAC para escribirla en una Vagrantfile
    - Ver los enlaces de información adicional de la siguiente transparencia
  - Instalar las Guest Additions de VirtualBox en la máquina
  - Crear un usuario "`vagrant`" con clave "`vagrant`"
    - En Linux debe ser un sudoer que no pida clave y hay que modificar el `/etc/sudoers` así: `vagrant ALL=(ALL) NOPASSWD: ALL`
  - Asegurarse de que hay un **gestor de paquetes** funcionando y un servidor `ssh` instalado y funcional

# ¿Cómo preparar nuestras boxes base?

José Manuel
Redondo López

- Para **Windows** además es necesario hacer una serie de pasos adicionales
  - Desactivar UAC
  - Desactivar la complejidad de las claves
  - Desactivar los servicios "**Shutdown Tracker**" y "**Server Manager**" si el **Windows** los usase
  - Habilitar y configurar adecuadamente **WinRM**
  - Crear una **Vagrantfile** que indique la forma de comunicarse con la máquina Windows
  - El proceso completo para **Windows 10** se puede encontrar aquí:
    - https://www.huestones.co.uk/2015/08/creating-a-windows-10-base-box-for-vagrant-with-virtualbox/
- Hecho esto se instala todo el software que queramos incluir como base
- Finalmente, se ejecuta desde dentro de la máquina: `vagrant package --base <nombre de la máquina a crear>`
  - Para **Windows**: `vagrant package --base <nombre de la máquina a crear> --output /path/to/output/windows.box --vagrantfile /path/to/initial/Vagrantfile`

## Más información:

- https://www.vagrantup.com/docs/providers/virtualbox/boxes
- https://www.vagrantup.com/docs/boxes/base

# Boxes Windows modernas

José Manuel
Redondo López

- **Stefan Scherer (contribuidor de Vagrant) tiene una colección de boxes de Vagrant modernas disponibles en la Vagrant Cloud**
  - https://app.vagrantup.com/StefanScherer
  - Tenemos todo tipo de Windows 10, 11 y también versiones Server
  - **Son todas versiones de evaluación**, pero suficientes para hacer pruebas de programas puntuales
    - Es decir, no están registradas (obviamente) y caducan en pocos meses
    - Una especialmente popular es esta: https://app.vagrantup.com/StefanScherer/boxes/windows_10
- **Estas imágenes son mencionadas en la documentación oficial de Microsoft para integrar imágenes Vagrant en la nube de Azure**
  - https://docs.microsoft.com/es-es/azure/cloud-adoption-framework/manage/hybrid/server/best-practices/local-vagrant-windows
- **Y se han construido gracias a los scripts de Packer del mismo autor**
  - Herramienta que veremos a continuación

# Packer

## Construyendo imágenes a partir de ISOs y más

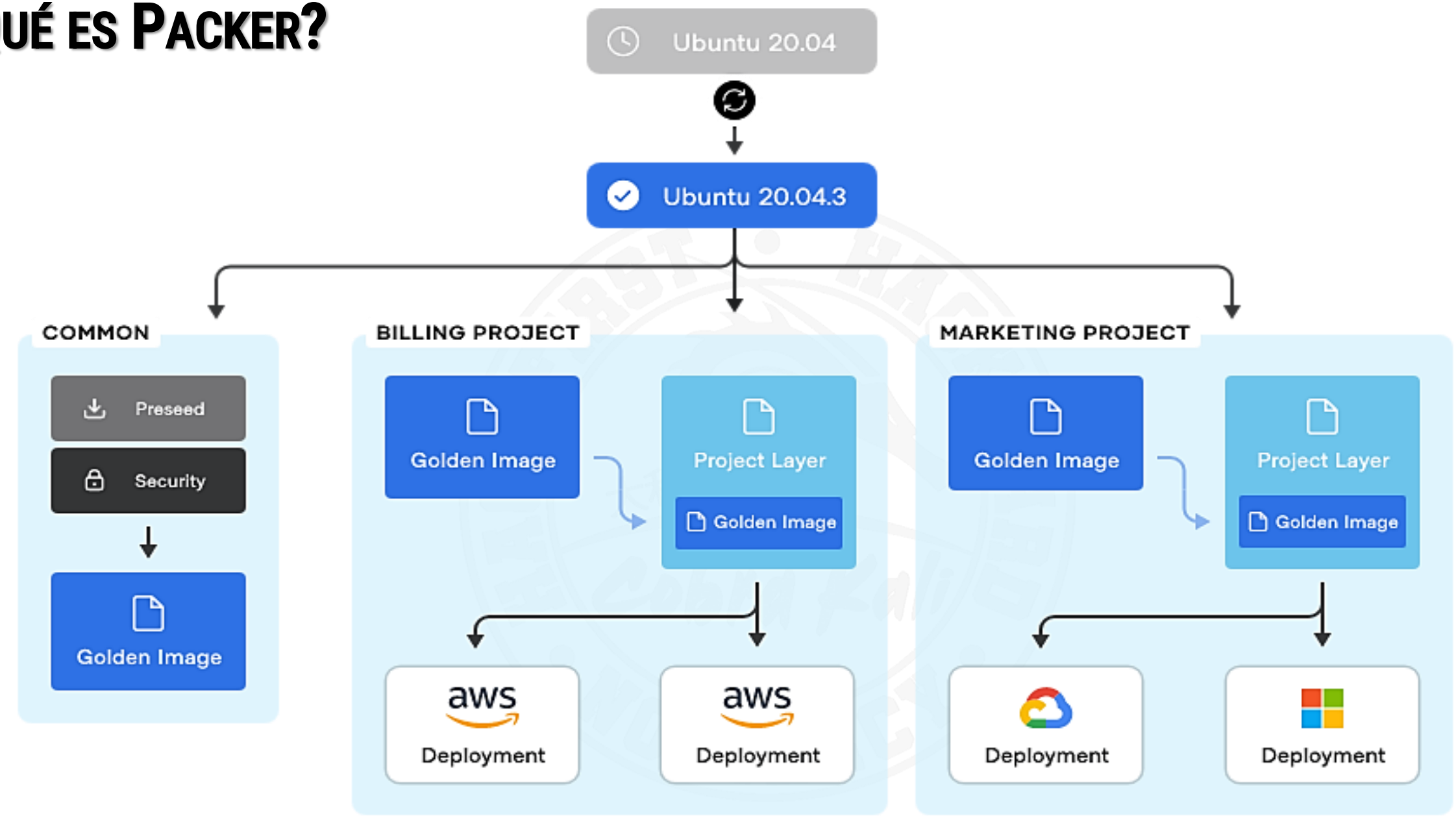
# ¿Qué es Packer?

José Manuel
Redondo López

- **Hemos visto que Vagrant levanta infraestructuras de máquinas virtuales gracias a un proveedor de virtualización**
- **El objetivo de Packer (https://www.packer.io/) es sin embargo crear plantillas / imágenes / AMI (Amazon Machine Image) idénticas para cualquier proveedor**
  - O cualquier artefacto que use un proveedor de virtualización soportado
- **Es decir, a partir de una única configuración es capaz de crear imágenes de máquinas para múltiples plataformas de forma automatizada**
  - Usando herramientas de provisionamiento como Ansible (que veremos luego), Chef, Puppet...para instalar el software que queramos dentro de la imagen
  - ¡Una verdadera capacidad multi-proveedor!
- **Con Packer se pueden crear boxes base de Vagrant**
  - Incluyendo instalar sistemas operativos automáticamente a partir de una ISO que tengamos

# ¿Qué es Packer?

José Manuel
Redondo López




**Fuente**: https://cloud.hashicorp.com/products/packer

# Ventajas de usar Packer

José Manuel
Redondo López

- **Implementación rápida de infraestructuras**
  - Las imágenes de Packer se pueden generar con todo el software necesario preinstalado (provisionamiento) en muy poco tiempo
- **Portabilidad de múltiples proveedores**
  - Packer crea imágenes idénticas para múltiples plataformas, lo que permite hacer cosas como esta garantizando la **máxima portabilidad**
    - Tener la imagen de desarrollo en un VirtualBox
    - Tener la imagen de pruebas/QA en una nube privada (Ej.: OpenStack)
    - Tener la imagen de producción en AWS
- **Detección temprana de errores**
  - Los errores de instalación del software o de la configuración se detectan cuando la máquina se construye, y no en funcionamiento
- **Mayor capacidad de prueba**
  - Después de construir una imagen se puede lanzar para probar que funciona correctamente

# Terminología de Packer

José Manuel
Redondo López

- **Artifacts** (artefactos): Los resultados de una compilación
  - Son un conjunto de IDs o archivos que representan una imagen de una máquina
  - Cada **builder** (ver luego) produce un solo artefacto. Ejs.:
    - El **builder** de **Amazon EC2** produce como artefacto un conjunto de ID de AMI (uno por región)
    - Para el de **VMware**, el artefacto es un directorio de archivos de la máquina virtual creada.
- **Templates**: Archivos JSON que definen una o más **builds** mediante la configuración de los diversos componentes de **Packer**
  - **Packer** lee plantillas para crear múltiples imágenes de máquinas en paralelo con esa información
  - Existen gran cantidad de **plantillas prefabricadas** que podemos usar para crear nuestras imágenes
    - https://github.com/StefanScherer/packer-windows
  - Tanto para despliegues **locales** (**Boxes** para nuestro **Vagrant**) como **en nube**
    - Ejemplo con AWS: https://medium.com/techno101/packer-a-complete-guide-with-example-cf062b7495eb
- **Builds**: Tareas que crean una imagen para una sola plataforma
  - Se pueden ejecutar varias en paralelo, una por cada **builder** que va a producir un **artifact**

# Terminología de Packer

José Manuel
Redondo López

- **Builders**: Componentes que crean imágenes de máquinas para una plataforma
  - Leen la configuración y la usan para ejecutar y generar una imagen de máquina
  - Se invocan como parte de una **build** para crear las imágenes
  - Se pueden añadir nuevos **builders** para otras plataformas a **Packer** como **plugins**
- **Commands**: Subcomandos parte de su interfaz en línea de comandos
  - Hacen distintas tareas (Ej.: `packer build`)
- **Provisioners**: Componentes que instalan y configuran software dentro de una máquina en ejecución
  - Antes de que esa máquina se convierta en una imagen estática distribuible
  - Se soportan varios tipos, como **scripts** de **shell**, **Ansible** (se ve en otro tema), **Chef**, **Puppet**, etc.
  - Es idéntico al concepto de **provisioner** que vimos en **Vagrant**
- **Post-processors**: Componentes que leen el resultado de un **builder** o de otro **post-processor** y lo procesan para crear un nuevo artefacto con algún cambio
  - Comprimir la imagen, cargarla en alguna parte, etc.

# Template de ejemplo de Packer

José Manuel
Redondo López

```
{
    // Distintos builders entre {}
    "builders": [
    {
      ...
      "type": "qemu", // Para que plataforma es este builder
      ....
    },
    {
      ....
      "type": "hyperv-iso",
      ....
    },
    {
      ....
      "type": "virtualbox-iso",
      ....
    }
  ],
  "post-processors": [
    {
      "keep_input_artifact": false,
      "output": "windows_10_{{.Provider}}.box",
      "type": "vagrant",
      "vagrantfile_template": "vagrantfile-windows_10.template"
    }
  ],
  "provisioners": [
    {
      "execute_command": "{{ .Vars }} cmd /c \"{{ .Path }}\"",
      "remote_path": "/tmp/script.bat",
      "scripts": [
        "./scripts/enable-rdp.bat"
      ],
      "type": "windows-shell"
    },
    {
      "scripts": [
        "./scripts/vm-guest-tools.ps1",
        "./scripts/debloat-windows.ps1"
      ],
      "type": "powershell"
    },
    {
      "restart_timeout": "{{user `restart_timeout`}}",
      "type": "windows-restart"
    },
    {
      "scripts": [
        "./scripts/set-powerplan.ps1",
        "./scripts/docker/disable-windows-defender.ps1"
      ],
      "type": "powershell"
    },
    ....
  ],
  "variables": {
    "autounattend": "./answer_files/10/Autounattend.xml",
    "disk_size": "61440",
    "disk_type_id": "1",
    "memory": "2048",
    "headless": "false",
    "iso_checksum": "sha256:026607e7aa7ff80441045d8830556bf8899062ca9b3c543702f112dd6ffe6078",
    "iso_url": "https://software-download.microsoft.com/download/sg/19043.928.210409-1212.21h1_r
    "restart_timeout": "5m",
    "vhv_enable": "false",
    "virtio_win_iso": "~/virtio-win.iso",
    "vm_name": "windows_10",
    "winrm_timeout": "6h",
    "vmx_version": "14"
  }
}
```

# ¿CREES QUE LO HAS ENTENDIDO TODO? ¡AUTOEVALÚATE!

José Manuel
Redondo López

?

- ***¿Comprendes que, si no encuentras lo que buscas, siempre puedes buscar a ver si existe un plugin para Vagrant que podría hacerlo?***
- **Optimizar boxes**
  - *¿Entiendes las opciones de VirtualBox que minimizan el uso de memoria y CPU de las MV?*
    - *¿Sabes que es una MV headless?*
    - *¿Sabes distinguir un linked clone de una MV de un clon normal?*
    - *¿Entiendes en qué consiste Page Fusion?*
  - *¿Comprendes porque basarse una imagen Alpine garantiza un tamaño mínimo, especialmente si lo comparamos con otras distribuciones de Linux?*
  - *¿Eres consciente de que el tamaño de tu MV dependerá del nº de paquetes instalados, y de por qué hay que insistir tanto en minimizarlos? ¿Sabes por qué esto es bueno para la seguridad?*
  - *¿Por qué crees que vaciar la caché y los log es una buena forma de minimizar el tamaño de una box Vagrant? ¿Perdemos información importante?*
    - *¿Tiene sentido esa información cuanto arranquemos máquinas a partir de esa box?*
  - *¿Entiendes por qué poner a cero todo el espacio libre de un HD puede disminuir enormemente el tamaño del .ova generado?*

# ¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

?

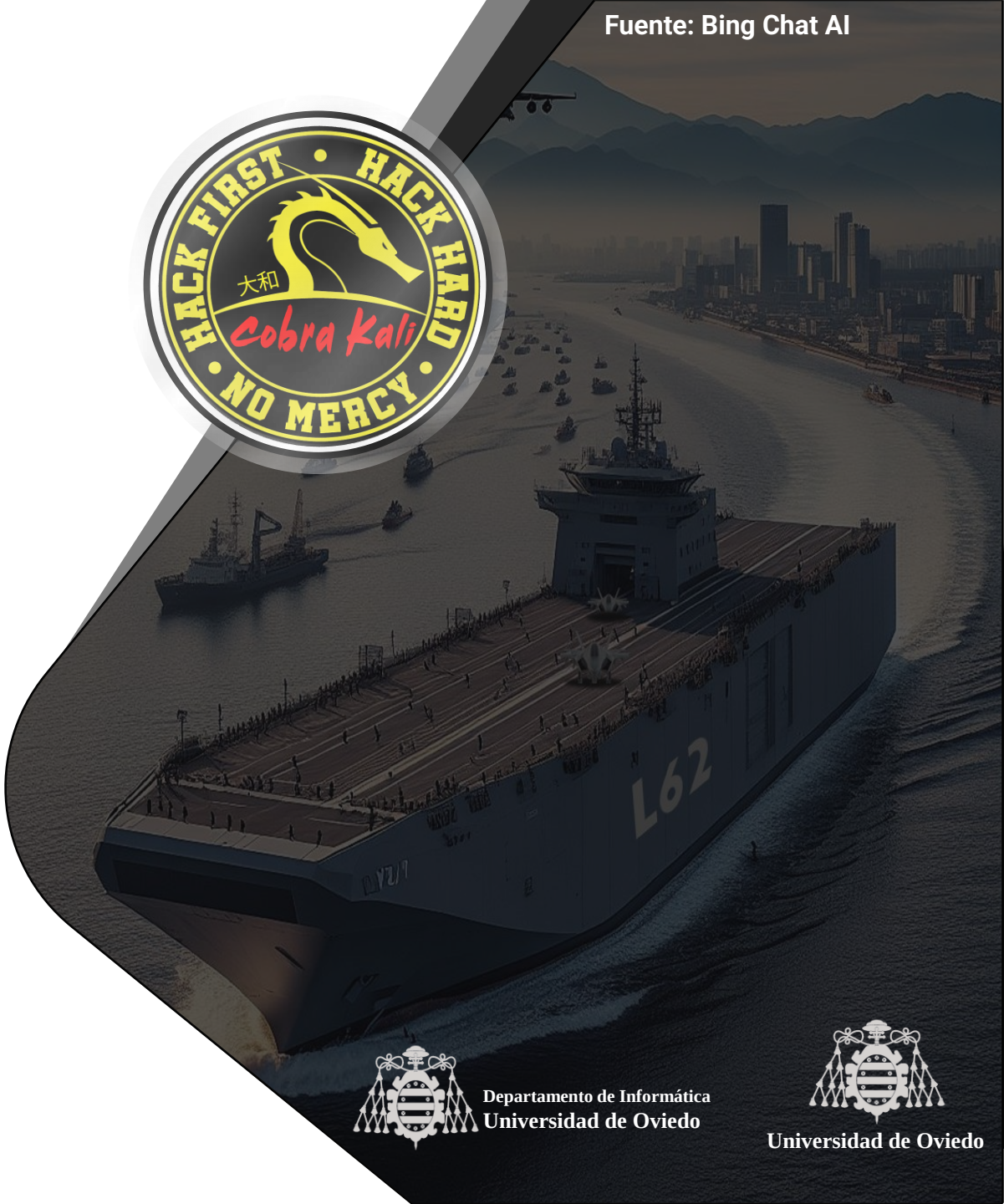
## Preparación de boxes

- *¿Recuerdas aproximadamente las operaciones generales que tienes que hacer para convertir una MV Linux que tengas en algo utilizable como base box de Vagrant?*
- *¿Sabes dónde encontrar las operaciones generales que tienes que hacer para convertir una MV Windows que tengas en algo utilizable como base box de Vagrant?*
- *¿Sabes dónde encontrar boxes Windows 10 relativamente actualizadas ideales para hacer pruebas puntuales con Windows?*

## Packer

- *¿Entiendes para qué sirve Packer?*
- *¿Eres consciente de que te puede ayudar a generar MVs adecuadas a partir de una configuración cuando tienes varios proveedores de virtualización a usar (locales o en nube)?*
- *¿Has entendido que Packer tiene un builder para cada tipo de imagen que produce destinada a un proveedor distinto?*
- *¿Sabes qué es una build y que usan para lanzarse y construir las imágenes?*
- *¿Sabes cuál es el propósito de un builder?*
- *¿Entiendes para que sirven los post-processors?*

# INTRODUCCIÓN A VAGRANT


Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo