



PROYECTO FINAL CFGS ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS Y REDES

Virtualización: código como infraestructura

| | |
|---------------|-------------------------|
| Alumno | José Pedro Campos Pérez |
| Tutor | Daniel Moreno Rosselló |
| Curso | 2018-2019 |

Resumen

Este proyecto tiene la finalidad de acercar todos los conocimientos necesarios para que los estudiantes puedan salvar las diferencias que se encuentran entre las técnicas de virtualización que reciben durante la formación profesional y las más actuales que son empleadas por las empresas. El proyecto iniciará con una presentación de las herramientas actuales de virtualización, una comparación entre estos sistemas y finalizará con unos ejercicios prácticos en los que se utilizarán las herramientas mostradas para desplegar aplicaciones virtuales desarrolladas en el curso por los mismos alumnos.

Palabras clave: virtualización, contenedor, Vagrant, Docker, Ansible, VirtualBox.

Resum

Aquest projecte té la finalitat d'apropar tots els coneixements necessaris per a que els alumnes puguin salvar les diferències que es troben entre les tècniques que reben durant la formació professional i les més actuals que són emprades per les empreses. El projecte s'iniciarà amb una presentació de les eines actuals de virtualització, una comparació entre aquestes i finalitzarà amb uns exercicis pràctics en els que es faran servir les eines mostrades per desplegar aplicacions virtuals desenvolupades en el curs per els mateixos alumnes.

Paraules clau: virtualització, contenidor, Vagrant, Docker, Ansible, VirtualBox.

Abstract

This project aims to bring all the necessary knowledge so that students can bridge the gap that are between the virtualization techniques they receive during their formation and the most current tools that are used by companies. The project will start with a presentation of the current virtualization tools, a comparison between these systems and end with practical exercises in which the tools shown to deploy virtual applications developed in the course by the students themselves will be used.

Key words: virtualization, container, Vagrant, Docker, Ansible, VirtualBox.

Índice

| | |
|--|----|
| Capítulo 1. Acerca de este proyecto | 5 |
| 1.1. Motivación | 5 |
| 1.2. Objetivos | 5 |
| 1.3. Metodología | 5 |
| Capítulo 2. Introducción | 6 |
| 2.1. Virtualización | 6 |
| 2.2. Máquinas virtuales | 7 |
| 2.3. Hipervisores | 7 |
| 2.4. Contenedores | 8 |
| 2.5. Contenedores y máquinas virtuales | 9 |
| 2.6. Beneficios de la virtualización | 10 |
| Capítulo 3. Herramientas | 13 |
| 3.1. VirtualBox | 13 |
| 3.2. VMware | 15 |
| 3.3. Vagrant | 16 |
| 3.3.1. Instalación de Vagrant | 16 |
| 3.3.2. Vagrant Boxes | 17 |
| 3.3.3. El fichero Vagrantfile | 19 |
| 3.3.4. Trabajando en máquinas con Vagrant | 23 |
| 3.3.5. Configuración de Hardware | 25 |
| 3.3.6. Configuración de red | 26 |
| 3.3.7. Configuración de hostname | 28 |
| 3.3.8. Compartir carpetas | 28 |
| 3.3.9. Aprovisionamiento | 29 |
| 3.4. Docker | 31 |
| 3.4.1. Instalación de Docker | 32 |
| 3.4.2. Docker Hub y Docker Search | 33 |
| 3.4.3. Trabajando con contenedores Docker | 35 |
| 3.4.4. Dockerfile | 38 |
| 3.4.5. Redes en Docker | 42 |
| 3.4.6. Volúmenes | 47 |
| 3.5. Comparación entre Vagrant y Docker | 48 |

| | |
|---|-----------|
| Capítulo 4. Sistemas operativos orientados a contenedores..... | 49 |
| 4.1. CoreOS..... | 49 |
| 4.2. Otros sistemas operativos..... | 50 |
| Capítulo 5. Automatización de la configuración..... | 51 |
| 5.1. Docker Compose..... | 51 |
| 5.1.1. El fichero docker-compose.yml..... | 52 |
| 5.2. Ansible..... | 55 |
| 5.2.1. Instalación de Ansible..... | 55 |
| 5.2.2. Inventarios de Ansible..... | 58 |
| 5.2.3. Módulos y comandos ad-hoc..... | 61 |
| 5.2.4. Ansible Playbooks..... | 64 |
| Capítulo 6. Orquestación de contenedores..... | 66 |
| 6.1. Fleet..... | 66 |
| 6.2. Otras herramientas para la orquestación de contenedores..... | 66 |
| Capítulo 7. Despliegamiento..... | 69 |
| 7.1. Vagrant..... | 69 |
| 7.2. Docker..... | 71 |
| 7.3. Docker Compose..... | 73 |
| Bibliografía..... | 74 |

Acceso al repositorio

https://github.com/jcamposp/TFinal_Virtualizacion

Capítulo 1.-Acerca de este proyecto

1.1.-Motivación

La motivación del siguiente proyecto nace con el propósito de servir como un módulo extraordinario que se podría desarrollar en los institutos de formación profesional para salvar las diferencias entre las herramientas de virtualización que se utilizan durante la formación y las herramientas actuales que se emplean en los sectores públicos y privados.

1.2.-Objetivos

El objetivo principal es la aplicación de distintas tecnologías de virtualización, contenedores y orquestación a una aplicación web para generar un entorno reproducible de desarrollo tanto en local como en la 'nube'.

Debido a la falta de tiempo o por la composición de la materia curricular de los ciclos, todavía no encontramos que estas herramientas se encuentren en los objetivos específicos para la formación de los estudiantes y consideramos que resultan muy importantes que las conozcan y se encuentren familiarizados con la materia, su entorno y aplicaciones y que hayan trabajado con ellas pudiendo investigar sus procesos.

1.3.-Metodología

La metodología seguida en este proyecto sigue una progresión pedagógica: presentación, características, instalación y ejemplo de utilización. Considerando sus beneficios y realizando una comparación con otras herramienta similares.

Por último, y como ejercicio práctico, se desplegará una aplicación PHP realizada en el módulo 'Implantación de aplicaciones Web' en un contenedor Docker sobre un servicio en la 'nube'.

Capítulo 2.-Introducción

2.1.-Virtualización

La virtualización consiste en la abstracción de los recursos de una computadora y su puesta en funcionamiento como máquina virtual en otra máquina física.

Al virtualizar un ordenador, se convierte en un paquete de software que puede ser ejecutado en otro ordenador o servidor. De esta forma, se puede tener un gran servidor que ejecuta varios de estos paquetes y conseguir tener varios sistemas informáticos virtualizados sobre un mismo servidor físico. Dentro de este entorno, se denomina “host” o “anfitrión” al servidor físico que alberga diversos “guests” o “invitados” que es la denominación que reciben las máquinas virtuales.



Figura 1: Servidor que ejecuta varios sistemas operativos

Un sistema de virtualización debe ser capaz de ofrecer una interfaz en la máquina anfitrión para poder interactuar con el sistema operativo de la máquina virtual. Además, la máquina anfitrión debe ofrecer una interfaz de sus recursos a la máquina virtual para que pueda utilizarlos. De estas interfaces de comunicación se encarga un software, hipervisor, que se instala en la máquina anfitriona para poder ejecutar las máquinas virtuales. Algunos de los tipos de virtualización son:

- **Virtualización de hardware:** creación de máquinas virtuales que se comportan como máquinas reales con su propio sistema operativo, funcionando en una máquina anfitrión que carece de sistema operativo. La máquina anfitriona solo ejecuta un software de virtualización para ejecutar las máquinas virtuales.
- **Virtualización de sistema operativo:** similar a la virtualización de hardware pero la máquina anfitriona ejecuta un sistema operativo y el software de virtualización interactúa con él.

- **Virtualización de almacenamiento:** unión de múltiples dispositivos de almacenamiento conectados en red, de tal manera que parezcan una única unidad. Volúmenes.
- **Virtualización de escritorio:** separación del entorno de escritorio de la máquina física, permitiendo que sea en una máquina servidor de escritorios donde se ejecuten los programas y se realicen los cálculos, mientras que en otra máquina remota se muestre la interfaz gráfica del escritorio con la que el usuario interactúa.

2.2.-Máquinas virtuales

Una máquina virtual es un software que simula un sistema de computación y puede ejecutar programas como si fuere una computadora real. Una característica esencial de las máquinas virtuales es que los procesos que ejecutan están limitados por los recursos y abstracciones proporcionados por ellos.

Las máquinas virtuales se pueden clasificar en dos grandes categorías según su funcionalidad y su grado de equivalencia a una verdadera máquina: máquinas virtuales de sistema y máquinas virtuales de proceso.

Las máquinas virtuales de sistema permiten a la máquina subyacente multiplicarse entre varias máquinas virtuales, cada una ejecutando su propio sistema operativo. A la capa de software que permite la virtualización se conoce como monitor de máquina virtual, o hipervisor. Un hipervisor puede ejecutarse o bien directamente sobre el hardware o bien sobre un sistema operativo.

Las máquinas virtuales de proceso se ejecutan como un proceso normal dentro de un sistema operativo sirviendo de enlace entre un lenguaje de programación y el sistema operativo, realizando una interpretación u otra técnica de enlace entre fuente y código máquina. El ejemplo más conocido de este tipo de máquina virtual es la máquina virtual de Java, que interpreta un código intermedio entre Java y código máquina.

2.3.-Hipervisores

Un hipervisor o monitor de máquina virtual es una plataforma que permite aplicar diversas técnicas de control de virtualización para utilizar diferentes sistemas operativos en una misma computadora. Los hipervisores pueden clasificarse en dos tipos:

- **Hipervisor tipo 1: Bare-metal**

Es software que se ejecuta directamente sobre el hardware para ofrecer la funcionalidad descrita. También reciben el nombre de 'nativo' o 'unhosted'. Algunos de los hipervisores tipo 1 más conocidos son:

| | | |
|------------------|--------------------------|------------------|
| Linux KVM | VMware ESXi | Xen |
| Citrix XenServer | Microsoft Hyper-V Server | Oracle VM Server |



Figura 2: Hipervisor tipo 1.

- **Hipervisor tipo 2: Hosted**

Es software que se ejecuta sobre un sistema operativo para ofrecer la funcionalidad descrita. Algunos de los hipervisores de este tipo más conocidos son:

| | | |
|------------|--------|----------------------|
| VirtualBox | VMware | Server |
| Player | QEMU | Microsoft Virtual PC |



Figura 3: Hipervisor tipo 2.

2.4.-Contenedores

La virtualización basada en contenedores es una aproximación a la virtualización en la cual la capa de virtualización se ejecuta como una aplicación en el sistema operativo. Mediante esta virtualización, el kernel del sistema operativo se ejecuta sobre el nodo de hardware con varias máquinas virtuales aisladas, que están instaladas sobre el mismo sistema operativo.

Con la virtualización basada en contenedores, no existe la sobrecarga asociada con tener a cada invitado ejecutando un sistema operativo completamente aislado. Este enfoque puede mejorar el rendimiento porque existe un solo sistema operativo encargándose de los avisos de hardware. En otras palabras, el contenedor opera como un proceso para el sistema operativo y almacena la aplicación que se ejecuta y todas sus dependencias. Esta aplicación solamente tiene visibilidad sobre el sistema de ficheros virtual del contenedor y utiliza indirectamente el kernel del sistema operativo principal para ejecutarse.

Los contenedores son máquinas virtuales mucho más portables y menos exigentes a nivel de recursos que las máquinas virtuales convencionales puesto que en lugar de albergar un sistema operativo completo lo que hacen es compartir los recursos propios del propio sistema operativo anfitrión sobre el que se ejecutan.



2.5.-Contenedores y máquinas virtuales

Los contenedores se diferencian de las máquinas virtuales en la ubicación de la capa de virtualización y en la forma en que utilizan los recursos del sistema operativo.

Las máquinas virtuales se basan en un hipervisor que se instala sobre el hardware del sistema o del sistema operativo. Posteriormente, las instancias virtuales se pueden aprovisionar a partir de los recursos disponibles en el sistema. Las máquinas virtuales están completamente aisladas unas de otras y se pueden migrar de un sistema virtualizado a otro sin tener en cuenta el hardware del sistema o los sistemas operativos.

El entorno de los contenedores está dispuesto de manera diferente. Los contenedores se instalan encima de un sistema operativo anfitrión. Éstos se pueden aprovisionar a partir de los recursos disponibles del sistema y se pueden implementar las aplicaciones necesarias. De esta manera, cada aplicación contenedora comparte el mismo sistema operativo subyacente, el núcleo del sistema operativo anfitrión e incluso las librerías.

Los contenedores se consideran más eficientes que las máquinas virtuales desde el punto de vista de los recursos puesto que no añaden recursos adicionales para cada sistema operativo. Las instancias resultantes son más pequeñas y más rápidas en su creación o migración y un único sistema puede albergar muchos más contenedores que máquinas virtuales.

Respecto al almacenamiento en disco, una máquina virtual puede ocupar varios gigas ya que tiene que contener un sistema operativo mientras que los contenedores solo contienen aquello que les diferencia del sistema operativo en las que se ejecutan.

Desde el punto de vista del consumo del procesador y de la memoria RAM, los contenedores hacen un uso eficiente del sistema anfitrión ya que comparten recursos con él. Así, los contenedores únicamente utilizan la memoria RAM y la capacidad de cómputo que estrictamente necesiten.

2.6.-Beneficios de la virtualización

Las tecnologías de virtualización proporcionan beneficios tanto a usuarios como a proveedores de servicios. A grandes rasgos, la virtualización permite a las compañías ahorrar en los siguientes campos:

- **Tiempo:** en instalación y administración de servidores.
- **Dinero:** la instalación de máquinas virtuales en hardware ya existente y aprovechar todo su potencial, considerando que la mayoría se pueden encontrar infrautilizados.
- **Energía:** la consolidación de servidores produce un ahorro de costes energéticos pues serán necesarios menos servidores y todo lo que se encuentra a su alrededor (sistemas de refrigeración, switches, routers, etc.).

La virtualización aporta grandes beneficios que hacen de la ella una de las tecnologías más ventajosas.

| Ventajas |
|--|
| Consolidación de servidores físicos: unificar varios servidores virtuales en un único servidor físico. |
| Flexibilidad: instalar, mover y eliminar recursos de forma rápida, sencilla y centralizada. |
| Escalabilidad: crecimiento fácil y rápido aumentando el número de máquinas virtuales. |
| Disponibilidad y balanceo de carga: varios servidores repartiendo la carga de trabajo y asegurando el servicio. |
| Fiabilidad: el fallo de una máquina virtual no afecta al resto de máquinas en el servidor. |
| Seguridad: se consigue mediante el aislamiento entre máquinas virtuales. |
| Reducción de costes: menos servidores físicos, menos espacio ocupado en el CPD, menos gastos de electricidad, refrigeración, etc. |
| Facilidad y rapidez en la recuperación ante desastres: rápida creación y clonación de máquinas virtuales. |
| Rápida respuesta frente a cambios: variar la arquitectura o la asignación de recursos de forma rápida. |
| Administración centralizada: administrar distintos dispositivos de forma centralizada y simple. |

Pero no todos son ventajas y también es valioso conocer y evaluar los inconvenientes de trabajar con entornos de virtualización para tomar las decisiones apropiadas.

| Inconvenientes |
|---|
| Limitaciones de hardware: todo el hardware disponible es repartido entre diferentes servidores virtuales. El software físico disponible debe ser compartido con más de una máquina virtual. |
| Peligro de pérdida de rendimiento: los recursos físicos son limitados. Es conveniente una buena gestión y planificación para que no se produzca una pérdida de rendimiento derivado de la escasez de recursos. |
| Disponibilidad de drivers: no suele ser un gran inconveniente pero puede suceder que el hardware y el software de virtualización todavía no disponga de los drivers adecuados para gestionarlos. |
| Actualización de nuevas tecnologías: la rápida evolución tecnológica puede producir una falla entre los sistemas físicos y virtuales. Es necesario un tiempo de adaptación de los sistemas que gestionan máquinas virtuales para adaptarlos a las nuevas tecnologías emergentes. |
| Hardware virtual obsoleto: es necesaria una amplia tarea de gestión y administración del software de virtualización para mantener actualizado y acorde a las tecnologías existentes en cada momento. |
| Riesgo de avería en los servidores físicos: una avería en un servidor físico que alberga varias máquinas virtuales producirá una caída de todos los servicios en él. |
| Imposibilidad de visualización del tráfico entre máquinas virtuales: puesto que cada máquina virtual y cada aplicación hacen pasar el tráfico de red a través del hipervisor y al formar éste de un sistema cerrado dentro del sistema virtual, muchos productos de seguridad tales como cortafuegos, sistemas de detección y prevención de intrusos no pueden observar el tráfico del hipervisor. |
| Máquinas virtuales inactivas: pueden sufrir más riesgos contra ataques y amenazas debido a que no suelen formar parte del ciclo de mantenimiento, actualizaciones y parches por lo que al ser conectadas pueden ser vulnerables a ataques que se podían evitar con la instalación de las actualizaciones. |
| Migraciones de máquinas virtuales: puesto que pueden trasladarse entre equipos físicos, es importante tener en cuenta distintos aspectos tales como la posible diferencia de los niveles de seguridad de los equipos físicos entre los que se traslada la máquina virtual así como la posible solución de seguridad. |

La gran facilidad que supone crear nuevas máquinas virtuales sobre servidores físicos lleva a algunos departamentos de informática a una sobre virtualización. Es decir, se crean demasiadas máquinas virtuales con poca justificación y se hace un mal uso de los recursos existentes.

Esta sobre explotación puede producir efectos contrarios a los buscados. Se pierde la flexibilidad y la escalabilidad hasta el punto de ser necesario adquirir nuevos servidores físicos para mantener toda la infraestructura desplegada, lo que impacta en el ahorro de costes que busca la virtualización.

La respuesta frente a cambios y la recuperación ante desastres se ve comprometida. Del mismo modo, la administración centralizada adquiere gran complejidad y este hecho puede impactar en el número cualificación del personal encargado de gestionar la infraestructura.

Por lo tanto, aunque la virtualización puede ofrecer grandes ventajas y un importante ahorro de costes, es necesaria una buena planificación y gestión de todo el entorno para poder aprovechar los beneficios minimizando los inconvenientes.

Capítulo 3.-Herramientas

3.1.-VirtualBox

Oracle VM VirtualBox es un software de virtualización Open Source desarrollado por la empresa alemana *Innotek GmbH*, aunque actualmente es desarrollado y mantenido por Oracle. Está programado en C++ y ensamblador x86. A pesar de ser un programa gratuito, algunos módulos o extensiones tales como el uso de USB 3.0, PXE o el cifrado de unidades virtuales requieren el pago de una licencia.



Sus principales ventajas son:

| Ventajas |
|---|
| Es compatible con Windows, Linux, Solaris y macOS. |
| Puede manejarse y gestionarse mediante la consola de comandos. |
| Se pueden compartir archivos entre distintas máquinas virtuales. |
| Se pueden crear instantáneas que permiten la restauración del sistema en ese punto. |
| Soporte de gráficos 3D, aunque limitado. |
| Compatible con máquinas VMware. |
| Soporta la captura de vídeo. |
| Permite el cifrado de las unidades virtuales. |
| Soporta tanto USB 2.0 como 3.0. |

3.2.-VMware

Software de virtualización desarrollado por la empresa *VMware Inc.* Proporciona distintas herramientas gratuitas y de pago de licencia según las necesidades del usuario. La herramienta gratuita, *VMware Workstation Player*, permite crear nuevas máquinas virtuales y solo puede ejecutar máquinas realizadas esta versión, por lo que no podrá ejecutar máquinas creadas por la versión de pago. La versión de pago, *VMware Workstation Pro*, permite lo mismo que la versión gratuita y añadiendo nuevas funcionalidades como la clonación o la realización de instantáneas



VMware posee las siguientes características, aunque la mayor parte de ellas necesitan la versión de pago para su utilización.

| Ventajas |
|---|
| Cuenta con gran cantidad de herramientas y módulos para entornos empresariales. |
| Permite compartir archivos entre el sistema anfitrión y la máquina virtual. |
| Compatible con lectores de tarjetas inteligentes. |
| Permite la creación de instantáneas. |
| Permite compartir máquinas virtuales. |
| Permite su integración con vSphere/ESXi y vCloud Air. |
| Soporta gráficos 3D compatibles con DirectX 10 y OpenGL 3.3. |
| Soporta tanto USB 2.0 como 3.0. |

Para la realización de este proyecto se ha escogido utilizar VirtualBox por haber trabajado con él durante estos dos años y por su gran documentación y ejemplos existentes. Además, puesto que VirtualBox es Open source pueden crearse y configurarse una gran cantidad de máquinas virtuales sin ningún tipo de restricción y aprovechar su gran cantidad de características y funcionalidades accesibles de forma gratuita que VMware no permitiría sin adquirir una licencia.

3.3.-Vagrant

Vagrant es una herramienta de código abierto creada en 2010 por Mitchell Hashimoto y escrito en lenguaje Ruby, pudiendo ser utilizado en proyectos escritos en otros lenguajes de programación. Vagrant permite crear y configurar entornos de desarrollo portátiles y virtualizados permitiendo definir los servicios a instalar así como también sus configuraciones, centralizando todo el proceso en la infraestructura del código y su automatización.

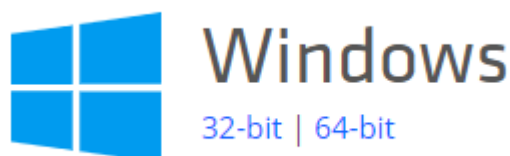


Vagrant permite crear y manejar máquinas virtuales con un mismo ecosistema de trabajo, definiendo determinadas características y componentes de la máquina, como puede ser la memoria RAM, la capacidad física, el número de procesadores, etc. Este mecanismo eficiente se consigue a través de un fichero de configuración, llamado *Vagrantfile*, donde se centraliza toda la configuración de la máquina virtual que se creará. Este fichero de configuración se puede utilizar para crear una máquina virtual exactamente igual cuantas veces se desee y debido a que es un fichero muy ligero permite ser incluido en repositorios o ser enviado por correo, o cualquier otro medio de comunicación, a otros usuarios para que lo ejecuten, disponiendo estos usuarios de la misma máquina virtual con exactamente el mismo ecosistema.

Para poder utilizar Vagrant será necesario disponer de un proveedor de máquinas virtuales, como VirtualBox, Hyper-V o VMware, aunque para este último proveedor será necesario comprar Vagrant para su utilización. Para el desarrollo de este trabajo, elegimos VirtualBox por ser la herramienta con la que nos hemos familiarizado, por ser multiplataforma y gratuita.

3.3.1.-Instalación de Vagrant

La instalación de Vagrant resulta muy sencilla: solo tendremos que dirigirnos a la página oficial de Vagrant, www.vagrantup.com, dirigirnos al apartado de descargas y seleccionar el ejecutable para nuestro sistema operativo. La versión de Vagrant en la que se realiza este trabajo es la 2.2.4 y para VirtualBox la 6.0.8 en Windows 10.



Podemos comprobar que Vagrant se encuentra instalado en nuestro sistema si al escribir Vagrant en la línea de comandos, interfaz con la que interactuamos con Vagrant, devuelve una salida como la siguiente:

```
C:\Users\jpcam>vagrant
Usage: vagrant [options] <command> [<args>]

    -v, --version          Print the version and exit.
    -h, --help             Print this help.

Common commands:
    box                    manages boxes: installation, removal, etc.
    cloud                  manages everything related to Vagrant Cloud
    destroy                stops and deletes all traces of the vagrant machine
```

Podemos comprobar la versión de Vagrant con el siguiente comando:

```
C:\Users\jpcam>vagrant -v
Vagrant 2.2.4
```

3.3.2.-Vagrant Boxes


Lo que hace posible que Vagrant cree máquinas virtuales sin necesidad de tener con nosotros imágenes ISO de los sistemas operativos y que las instale por nosotros son las *boxes*. Estas *boxes* son las imágenes que sirven de plantilla para crear nuestras máquinas virtuales y que se encuentran en servidores web listas para ser utilizadas por Vagrant y ser importadas. Estas *boxes* son unidades empaquetadas que contienen un sistema operativo con una determinada configuración.

Podemos consultar estas *boxes* dirigiéndonos a la página oficial de Vagrant y accediendo a la sección Boxes. Aquí, encontraremos todas las *boxes* disponibles pudiendo realizar búsquedas por sistema operativo, arquitectura, software, hipervisor, etc.

Search for boxes by operating system, included software, architecture and more

Provider any virtualbox vmware libvirt more

Sort by Downloads Recently Created Recently Updated



ubuntu/trusty64 20190429.0.1

Official Ubuntu Server 14.04 LTS (Trusty Tahr) builds


virtualbox

Downloads

30,382,078

Released

12 days ago



laravel/homestead 8.0.0-alpha1

Official Laravel local development box.

hyperv

parallels

virtualbox

vmware_desktop

Downloads

13,500,077

Released

about 17 hours ago

En el caso de la imagen que se muestra, descargar la primera *box* supondría descargar un paquete con el sistema operativo Ubuntu Server 14.04 LTS. La siguiente *box* que se muestra, laravel/homestead (8.0.0), es una máquina virtual que ya viene definida con las siguientes características:

Included Software

- | | | |
|--------------------|--------------------|-----------------------|
| • Ubuntu 18.04 | • snapshots | • Beanstalkd |
| • Git | • Sqlite3 | • Mailhog |
| • PHP 7.3 | • PostgreSQL | • avahi |
| • PHP 7.2 | • Composer | • ngrok |
| • PHP 7.1 | • Node (With Yarn, | • Xdebug |
| • Nginx | Bower, Grunt, and | • XHProf / Tideways / |
| • MySQL | Gulp) | XHGui |
| • Imm for MySQL or | • Redis | • wp-cli |
| MariaDB database | • Memcached | • Minio |

Optional Software

- | | | |
|-------------------|-------------|-----------------------|
| • Apache | • Go | • Ruby & Rails |
| • Crystal & Lucky | • MariaDB | • Webdriver & Laravel |
| Framework | • MongoDB | Dusk Utilities |
| • Dot Net Core | • Neo4j | • Zend Z-Ray |
| • Elasticsearch | • Oh My Zsh | |

Es decir, la descarga de esta última *box* supone disponer de una máquina virtual que ya tiene pre-instalado todo el software que necesitaríamos para trabajar, pudiendo trabajar con ella e incluso destruirla y ser descargada en cuestión de minutos y disponer de una nueva máquina lista para ser utilizada.

Los *boxes* del repositorio de Vagrant pueden ser descargados a través de la línea de comandos, aunque no será necesario disponer de ellos en nuestro sistema para utilizarlos, ya que si tratamos de levantar una máquina sin disponer de la *box* en nuestro sistema, Vagrant acudirá al repositorio para utilizar la *box* objetivo y realizará la creación de la máquina virtual en nuestro sistema.

De todas maneras, si queremos descargar *boxes* en nuestro sistema podemos hacerlo a través del comando siguiente, por ejemplo para un Ubuntu 16.04:

```
vagrant box add <nombre de la box>
```



ubuntu/xenial64 20190521.0.0
Official Ubuntu 16.04 LTS (Xenial Xerus) Daily Build

virtualbox

Downloads
3,244,669

Released
6 days ago

```
C:\Users\jpcam>vagrant box add ubuntu/xenial64
==> box: Loading metadata for box 'ubuntu/xenial64'
box: URL: https://vagrantcloud.com/ubuntu/xenial64
==> box: Adding box 'ubuntu/xenial64' (v20190521.0.0) for provider: virtualbox
box: Downloading: https://vagrantcloud.com/ubuntu/boxes/xenial64/versions/20190521.0.0/providers/virtualbox.box
box: Download redirected to host: cloud-images.ubuntu.com
box: Progress: 100% (Rate: 7263k/s, Estimated time remaining: --:--:--)
==> box: Successfully added box 'ubuntu/xenial64' (v20190521.0.0) for 'virtualbox'!
```

El nombre de la *box* es el nombre con el cual aparece en el repositorio de imágenes de la página oficial de Vagrant. Al ejecutar el comando, inmediatamente comienza la descarga de la *box*, y en cuestión de escasos minutos tendremos la *box* disponible en nuestro ordenador.

Una vez descargadas las *boxes*, podemos consultar cuáles tenemos en nuestro sistema a través del siguiente comando:

```
vagrant box list
```

```
C:\Users\jpcam>vagrant box list
laravel/homestead (virtualbox, 8.0.0-alpha1)
ubuntu/trusty64   (virtualbox, 20190429.0.1)
ubuntu/xenial64   (virtualbox, 20190521.0.0)
```

3.3.3.-El fichero Vagrantfile

El fichero Vagrantfile es un fichero necesario para configurar Vagrant por proyecto. Este fichero está escrito en lenguaje Ruby, pero el conocimiento de este lenguaje no es necesario para realizar modificaciones en el fichero, ya que en su mayoría solo se realizan asignaciones simples de variables. La función principal y necesaria de este fichero es la de describir las máquinas virtuales para un proyecto, así como la forma de configurar y aprovisionar estas máquinas. Será en este fichero en el cual se especifique qué *box* utilizar como plantilla y detallar las características de la máquina, tales como nombre del *hostname*, la cantidad de memoria RAM, CPUs, etc.; así como la forma de configurar y aprovisionar estas máquinas. En resumen, este fichero define:

- La *box* que servirá de plantilla de la máquina virtual.
- Las características de la máquina virtual.
- Conjunto de comandos que queremos que se ejecuten al crear la máquina. Como la instalación de aplicaciones y copias de datos desde carpetas compartidas en la máquina anfitrión a la máquina virtual..

Este fichero, de forma tradicional, se crea a través del siguiente comando. La ejecución del comando creará un fichero Vagrantfile en el directorio en el cual se ejecute. En el caso de que ya exista un fichero Vagrantfile en el directorio, la ejecución del comando devolverá un error.

```
vagrant init
```

```
C:\Users\jpcam\laboratorio>vagrant init
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

El paso siguiente será el de editar el fichero Vagrantfile para corresponder sus características con las deseadas. El fichero Vagrantfile está ampliamente comentado y tendremos que prestar especial atención en su edición para alcanzar las características que deseamos:

```
1  # -*- mode: ruby -*-
2  # vi: set ft=ruby :
3
4  # All Vagrant configuration is done below. The "2" in Vagrant.configure
5  # configures the configuration version (we support older styles for
6  # backwards compatibility). Please don't change it unless you know what
7  # you're doing.
8  Vagrant.configure("2") do |config|
9    # The most common configuration options are documented and commented below.
10   # For a complete reference, please see the online documentation at
11   # https://docs.vagrantup.com.
12
13   # Every Vagrant development environment requires a box. You can search for
14   # boxes at https://vagrantcloud.com/search.
15   config.vm.box = "base"
```

La característica a la que más especial atención hay que prestar es la que aparece subrayada en la anterior imagen, en la cual tendremos que definir la *box* que servirá como plantilla. En nuestro caso, hemos elegido la *box* de un Ubuntu 16.04.

```
1  # -*- mode: ruby -*-
2  # vi: set ft=ruby :
3
4  # All Vagrant configuration is done below. The "2" in Vagrant.configure
5  # configures the configuration version (we support older styles for
6  # backwards compatibility). Please don't change it unless you know what
7  # you're doing.
8  Vagrant.configure("2") do |config|
9    # The most common configuration options are documented and commented below.
10   # For a complete reference, please see the online documentation at
11   # https://docs.vagrantup.com.
12
13   # Every Vagrant development environment requires a box. You can search for
14   # boxes at https://vagrantcloud.com/search.
15   config.vm.box = "ubuntu/xenial64"
```

Solo nos quedará para levantar la máquina ejecutar un último comando. Hay que señalar que este comando tratará de levantar el fichero Vagrantfile que se encuentre en el directorio donde ejecutamos el comando por lo tanto tendremos que tener en cuenta cuál es el fichero objetivo del comando para dirigirnos a su localización. El comando es el siguiente:

```
vagrant up
```

```
C:\Users\jpcam\laboratorio>vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'ubuntu/xenial64'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'ubuntu/xenial64' version '20190521.0.0' is up to date...
==> default: Setting the name of the VM: laboratorio_default_1558903226841_13656

==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
default: The guest additions on this VM do not match the installed version of
default: VirtualBox! In most cases this is fine, but in rare cases it can
default: prevent things such as shared folders from working properly. If you see
default: shared folder errors, please make sure the guest additions within the
default: virtual machine match the version of VirtualBox you have installed on
default: your host and reload your VM.
default:
default: Guest Additions Version: 5.1.38
default: VirtualBox Version: 6.0
==> default: Mounting shared folders...
default: /vagrant => C:/Users/jpcam/laboratorio

C:\Users\jpcam\laboratorio>
```

Tras escasos un par de minutos tendremos lista la máquina. Si en este punto, abrimos nuestro proveedor de máquinas virtuales, VirtualBox, veremos que habrá aparecido una nueva máquina virtual que antes no estaba.



Como podemos ver en las imágenes, la máquina virtual creada ya se encuentra en marcha. Para acceder a la máquina bien lo podemos hacer desde el propio hipervisor o a través de Vagrant con el siguiente comando. Este comando, como el anterior, hay que ejecutarlo en el directorio desde donde se encuentra el fichero Vagrantfile pues depende de un directorio, '.vagrant' que se encuentra en esa localización. Lanzar este comando en otro directorio en el cual no se encuentre este directorio provocará un error. Por lo tanto, podemos acceder desde la localización y ejecutando el siguiente comando:

```
vagrant ssh
```

Las credenciales por defecto son 'vagrant' para usuario y 'vagrant' para la contraseña en caso de necesitarlas.

```
C:\Users\lcornejo\laboratorio>vagrant ssh
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-148-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.

New release '18.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Mon May 27 10:22:13 2019 from 10.0.2.2
vagrant@ubuntu-xenial:~$ ll
total 36
drwxr-xr-x 5 vagrant vagrant 4096 May 27 10:24 ./
drwxr-xr-x 4 root    root    4096 May 27 10:08 ../
-rw-r--r-- 1 vagrant vagrant  753 May 27 10:36 .bash_history
-rw-r--r-- 1 vagrant vagrant  220 May 21 15:05 .bash_logout
-rw-r--r-- 1 vagrant vagrant 3771 May 21 15:05 .bashrc
drwxr-xr-x 2 vagrant vagrant 4096 May 27 10:08 .cache/
drwxrwxr-x 2 vagrant vagrant 4096 May 27 10:24 .nano/
-rw-r--r-- 1 vagrant vagrant  655 May 21 15:05 .profile
drwxr-xr-x 2 vagrant vagrant 4096 May 27 10:08 .ssh/
```

El fichero Vagrantfile también lo podemos inicializar por defecto con la *box* directamente anexada combinando el comando 'init' con el nombre de la *box* como mostramos a continuación:

```
vagrant init ubuntu/trusty32
```

El fichero Vagrantfile resultante ya tendrá en el apartado de configuración de *box* la imagen que queremos utilizar y solo quedará levantar la máquina con el comando:

```
vagrant up
```

Otra forma de crear un Vagrantfile sería accediendo al repositorio de Vagrant en su página web y al acceder al *box* deseado veremos que el primer cuadro que aparece es como el siguiente:

```
Vagrantfile New
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
end
```

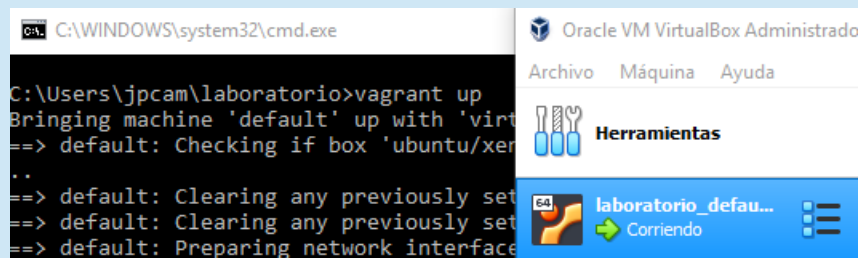

Este texto se corresponde al fichero Vagrantfile retirando todos los comentarios y ya tiene establecida como variable para la *box* la imagen de un Ubuntu Server 14.04. Solo tendremos que crear un fichero en blanco y copiar y pegar el texto, guardar el fichero como Vagrantfile, sin extensión, para después levantar la máquina

3.3.4.-Trabajando en máquinas Vagrant

Ya hemos visto que para acceder a una máquina creada por Vagrant podemos hacerlo a través del comando 'vagrant ssh' o a través del propio hipervisor, pero es de gran interés conocer otros comandos necesarios para trabajar con estas máquinas.

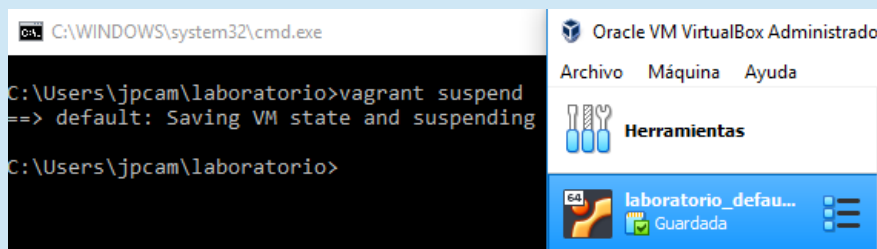
Encender una máquina apagada

- `vagrant up`



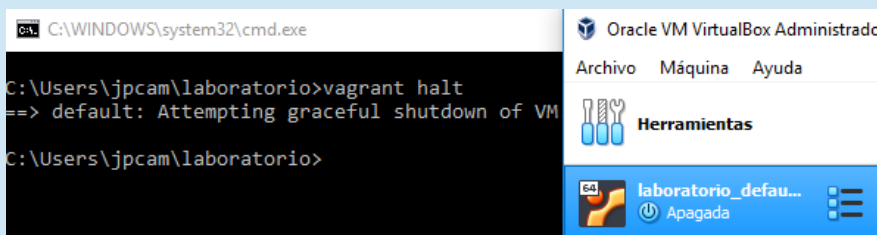
Suspender una máquina

- `vagrant suspend`



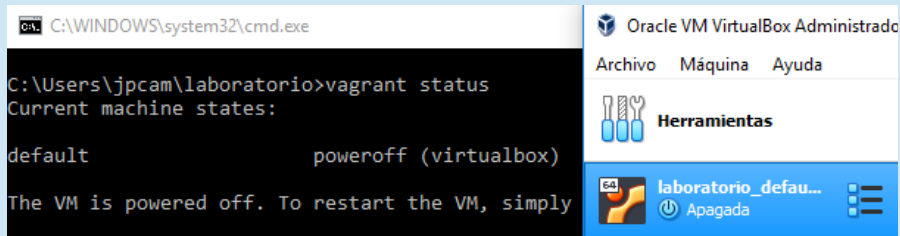
Apagar una máquina

- `vagrant halt`



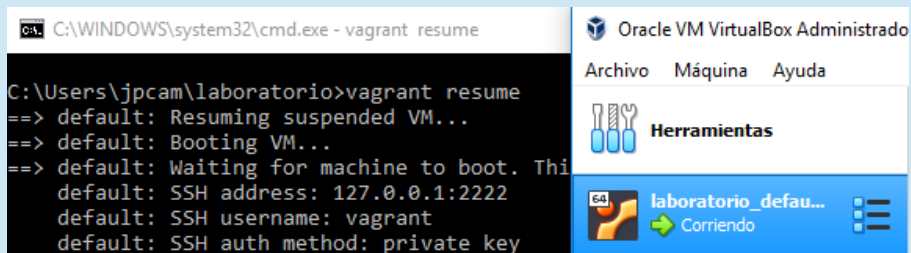
Comprobar el estado de la máquina

- `vagrant status`



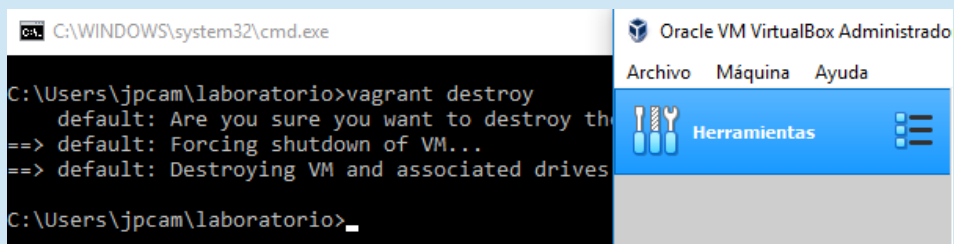
Iniciar una máquina suspendida

- `vagrant resume`



Destruir una máquina

- `vagrant destroy`



3.3.5.-Configuración de Hardware

Con Vagrant podemos indicar las características de los componentes de Hardware virtuales con los cuales queremos crear una máquina virtual, editando directamente el fichero Vagrantfile. Crear una nueva máquina virtual especificando estas características físicas requerirá la utilización del comando 'vagrant up', pero si queremos modificar alguna característica sobre una máquina ya creada tendremos que recurrir a un nuevo comando de Vagrant, con el cual se reiniciará la máquina y se establecerán las características pertinentes.

```
vagrant reload
```

Memoria RAM

Para modificar la memoria RAM de una máquina con Vagrant, accederemos a su fichero Vagrantfile y ubicaremos las siguientes líneas, retiraremos los comentarios y estableceremos la cantidad de RAM que necesitamos.

```
52  config.vm.provider "virtualbox" do |vb|
53    # # Display the VirtualBox GUI when booting the machine
54    # vb.gui = true
55    #
56    # # Customize the amount of memory on the VM:
57    vb.memory = "2048"
58  end
```

Procesadores

Para modificar el número de procesadores, tendremos que dirigirnos a la ubicación anterior pero tendremos que escribir el siguiente parámetro:

```
52  config.vm.provider "virtualbox" do |vb|
53    # # Display the VirtualBox GUI when booting the machine
54    # vb.gui = true
55    #
56    # # Customize the amount of memory on the VM:
57    vb.memory = "2048"
58    vb.cpus = "1"
59  end
```

La capacidad física de la unidad de almacenamiento dependerá de la *box* utilizada para crear la máquina virtual. Estamos hablando de que existen *boxes* que cuentan con unidades de 10GB, 20GB, 30GB, etc. Por lo que Vagrant no trae consigo ninguna herramienta para la modificación de esta capacidad.

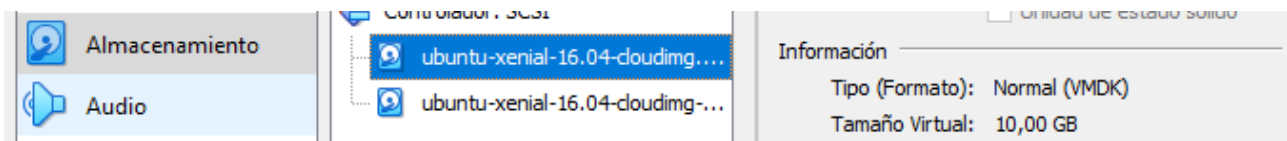
En cambio, existen posibilidades para realizar esta operación si la capacidad de la unidad de almacenamiento no nos convence de acuerdo a la *box* que estamos utilizando. Una de estas opciones es la utilización de un *plugin* que podemos instalar a partir de Vagrant y utilizar un nuevo parámetro para llevar a cabo el cambio de la capacidad de almacenamiento.

```
vagrant plugin install vagrant-disksize
```

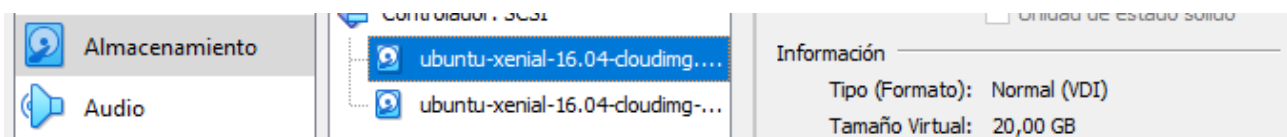
La utilización de este *plugin* es muy sencilla pero solamente puede ser utilizado teniendo en cuenta que tendremos que levantar una nueva máquina para que se lleve a cabo satisfactoriamente. Es decir, no podremos cambiar el tamaño pero sí crear una máquina con la capacidad de queremos. Solo tendremos que añadir el siguiente parámetro en el fichero Vagrantfile:

```
13 # Every Vagrant development environment requires a box. You can search for
14 # boxes at https://vagrantcloud.com/search.
15 config.vm.box = "ubuntu/xenial64"
16 config.disksize.size = "20GB"
```

En las siguientes imágenes podemos ver que nuestra *box*, ubuntu/xenial64, crea una unidad de 10GB por defecto.



Mientras que si levantamos una nueva máquina, obtendremos una unidad de almacenamiento definida por nosotros.



3.3.6.-Configuración de red

Podemos indicar a Vagrant de forma rápida mediante el fichero Vagrantfile la IP que queremos que la máquina utilice, tanto de forma pública, haciéndola accesible para todo el mundo, como de forma privada, formando parte de una subred privada que solamente sea accesible por los equipos de esa subred.

```
35 # Create a private network, which allows host-only access to the machine
36 # using a specific IP.
37 config.vm.network "private_network", ip: "192.168.100.1"
```

En cuanto al redireccionamiento de puertos, podemos hacer que nuestra máquina virtual sea accesible públicamente por el puerto que indicamos. Es decir, permitimos que al acceder a la máquina anfitrión se acceda a la máquina virtual a través del puerto que pongamos:

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

En el ejemplo anterior, cualquier petición a la máquina anfitrión por el puerto 8080, será trasladado a la máquina virtual por el puerto 80.

Podemos poner tantas reglas de redireccionamiento de puertos como requiramos copiando la misma línea e incluyendo diferentes opciones.

```
24 # Create a forwarded port mapping which allows access to a specific port
25 # within the machine from a port on the host machine. In the example below,
26 # accessing "localhost:8080" will access port 80 on the guest machine.
27 # NOTE: This will enable public access to the opened port
28 config.vm.network "forwarded_port", guest: 80, host: 8080
29 config.vm.network "forwarded_port", guest: 443, host: 8443
```

La configuración de red pública establece configurar la máquina virtual para crear un adaptador puente de red de modo que la máquina virtual pueda ser accesible desde la red pública a la que está conectada la máquina anfitrión. Para contar con esta característica, solo tendremos que retirar el comentario del Vagrantfile que le precede.

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

Ambas opciones crearán un adaptador de red en la máquina virtual disponible para ambas opciones.

```
enp0s8  Link encap:Ethernet  HWaddr 08:00:27:eb:c8:81
        inet addr:192.168.100.1  Bcast:192.168.100.255  Mask:255.255.255.0
        inet6 addr: fe80::a00:27ff:feeb:c881/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:0 (0.0 B)  TX bytes:1156 (1.1 KB)

enp0s9  Link encap:Ethernet  HWaddr 08:00:27:f9:e9:38
        inet addr:192.168.1.7  Bcast:192.168.1.255  Mask:255.255.255.0
        inet6 addr: fe80::a00:27ff:fe9:e938/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:9 errors:0 dropped:0 overruns:0 frame:0
        TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:1114 (1.1 KB)  TX bytes:1332 (1.3 KB)
```

3.3.7.-Configuración de hostname

Algo muy eficiente en los entornos de desarrollo es identificar todas las máquinas virtuales con nombres inequívocos y que no pongan en duda qué máquina es la que tenemos delante de nuestro terminal y lo podemos hacer a través de ponerle un nombre al hostname. Simplemente tendremos que añadir la siguiente línea:

```
14 # Every Vagrant development environment requires a box. You can search for
15 # boxes at https://vagrantcloud.com/search.
16 config.vm.box = "ubuntu/xenial64"
17 config.vm.hostname = "laboratorio1"
```

Tendremos que lanzar el comando 'vagrant reload' y al acceder a la máquina ya tendremos el nombre del hostname cambiado:

```
Last login: Sat Jun  1 21:11:30 2019 from 10.0.2.2
vagrant@laboratorio1:~$
```

Este cambio también se puede realizar desde dentro a partir de la modificación del fichero '/etc/hostname' pero cualquier modificación que suponga lanzar de nuevo el comando 'vagrant reload' sin haber modificado el nombre de *hostname* volverá a cambiarse al valor definido en el Vagrantfile.

3.3.8.-Compartir carpetas


De forma rápida podemos indicar a través del fichero Vagrantfile la posibilidad de compartir una carpeta entre la máquina virtual y la máquina anfitrión. Simplemente tenemos que dirigirnos al fichero Vagrantfile y buscar las siguientes líneas:

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

El primer argumento que hay que pasar es la ruta al directorio que queremos sincronizar con la máquina virtual en nuestra máquina anfitrión y el segundo argumento será el directorio dentro de la máquina virtual. Estos dos directorios se encontrarán en total sincronización.

Tenemos que señalar que el directorio en la máquina anfitrión tendremos que crearlo manualmente nosotros mientras que el directorio en la máquina virtual será creado de manera automática a través del fichero Vagrantfile.

Para demostrar el funcionamiento, crearemos en nuestra máquina anfitrión un fichero de texto en el directorio '/data' que se encuentra dentro del directorio en el cual se encuentra las dependencias de la máquina virtual y lo consultaremos desde la máquina virtual en el directorio que expusimos.



 prueba.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

Fichero de texto de prueba

```
vagrant@laboratorio1:~$ ll
total 32
drwxr-xr-x 5 vagrant vagrant 4096 Jun  2 08:29 ./
drwxr-xr-x 4 root     root     4096 Jun  1 17:58 ../
-rw----- 1 vagrant vagrant  256 Jun  2 08:27 .bash_history
-rw-r--r-- 1 vagrant vagrant  220 May 21 15:05 .bash_logout
-rw-r--r-- 1 vagrant vagrant 3771 May 21 15:05 .bashrc
drwx----- 2 vagrant vagrant 4096 Jun  1 17:58 .cache/
drwxrwxrwx 1 vagrant vagrant   0 Jun  2 07:42 data/
-rw-r--r-- 1 vagrant vagrant  655 May 21 15:05 .profile
drwx----- 2 vagrant vagrant 4096 Jun  1 17:58 .ssh/
vagrant@laboratorio1:~$ cd data/
vagrant@laboratorio1:~/data$ cat prueba.txt
Fichero de texto de pruebavagrant@laboratorio1:~/data$
```

De forma similar, todo cambio producido en el directorio sincronizado, será reflejado en el directorio de la máquina anfitrión, incluyendo cualquier tipo de fichero y directorio.

 nuevo directorio
 prueba.txt

```
vagrant@laboratorio1:~/data$ ll
total 5
drwxrwxrwx 1 vagrant vagrant   0 Jun  2 08:34 ./
drwxr-xr-x 5 vagrant vagrant 4096 Jun  2 08:33 ../
drwxrwxrwx 1 vagrant vagrant   0 Jun  2 08:34 nuevo_directorio/
-rwxrwxrwx 1 vagrant vagrant  53 Jun  2 08:33 prueba.txt*
```

3.3.9.-Aprovisionamiento

Podemos automatizar el proceso de configuración del entorno de la máquina virtual a través de la ejecución de una serie de comandos que se deberán ejecutar automáticamente durante el arranque de nuestra máquina en un *script*. Este proceso automático es lo que se conoce como aprovisionamiento.

Para aprovisionar a partir del fichero Vagrantfile, nos tendremos que dirigir al final del fichero donde encontraremos el siguiente apartado:

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

Básicamente, aprovisionar a partir de la utilización de *shell*, solo tendremos que retirar las líneas comentadas e insertar los comandos que queremos que se ejecuten una vez que la máquina se encuentre preparada y nosotros al ingresar a la máquina ya se encuentre lista toda su configuración realizada. Por ejemplo, uno de los comandos más básicos sería el de actualización del software o la instalación de algún paquete determinado tal y como muestra los comentarios del Vagrantfile.

Si retiramos los comentarios y lo dejamos tal como se muestra en la siguiente imagen, Vagrant se encargará de actualizar la máquina y de instalar Apache una vez se levante la máquina. Hay que destacar que los comandos que incluyamos en este apartado deben ser declarativos, es decir, que cualquier interacción que sea necesaria con el usuario para confirmarla debe ser automáticamente salvada, como podría ser la petición que realiza la instalación de un paquete al ocupar espacio en la unidad de almacenamiento. Es por esto mismo, que el comando para la instalación de Apache incluye el parámetro '-y' para indicar 'Yes' en su instalación.

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

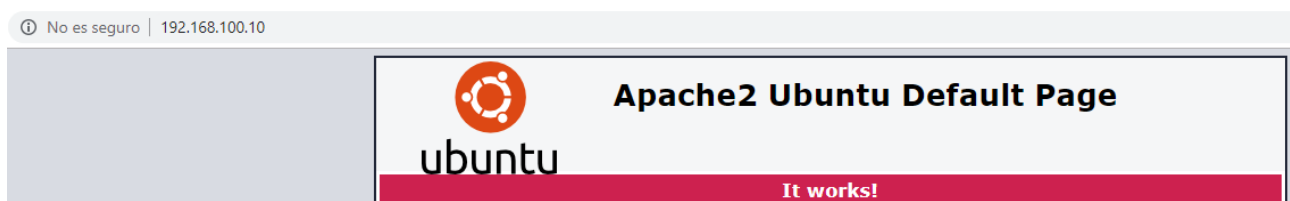
Para lanzar el aprovisionamiento, tendremos que valernos del comando siguiente.

```
vagrant reload --provision
```

Podemos comprobar que la instalación de Apache se ha llevado a través de la máquina virtual.

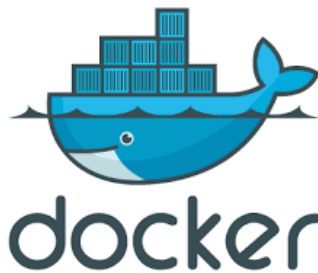
```
vagrant@laboratorio1:~$ sudo /etc/init.d/apache2 status
• apache2.service - LSB: Apache2 web server
  Loaded: loaded (/etc/init.d/apache2; bad; vendor preset: enabled)
  Drop-In: /lib/systemd/system/apache2.service.d
           └─apache2-systemd.conf
  Active: active (running) since Sun 2019-06-02 09:01:37 UTC; 1min 40s ago
```

O también podemos utilizar un navegador en la máquina anfitrión y escribir en la barra de navegación la IP de la máquina virtual para que aparezca la página principal de Apache.



3.4.-Docker

Docker es un gestor de contenedores Open Source programado en Go que fue lanzado por la empresa Docker Inc bajo la licencia Apache License 2.0. Surgió como un proyector interno de la empresa dotCloud, por Salomón Hykes, y pasó a ser de código abierto en marzo de 2013. A diferencia de Vagrant, no emula un sistema operativo completo, creando únicamente un contenedor en el que instala las librerías y herramientas necesarias para, aprovechando los recursos de la máquina anfitrión, ejecutar una o varias aplicaciones.



Docker permite ejecutar aplicaciones en entornos aislados y proporciona las mismas o más ventajas que a la hora de ejecutar nuestras aplicaciones en una máquina virtual. Docker permite tener el mismo entorno, evitando posibles inconsistencias entre entornos de desarrollo y producción, además de poder trabajar con los proyectos de otros desarrolladores sin tener que instalar dependencias y configuraciones.

Aunque pueda parecer lo mismo que una máquina virtual, Docker proporciona todas las ventajas y funcionalidades sin tener que estar gestionando y configurando máquinas virtuales gracias a los contenedores. Hay que señalar que en una máquina virtual se orquestan diferentes servicios y procesos mientras que en la filosofía de Docker en cada contenedor hay una aplicación, en cada contenedor hay un servicio y en cada contenedor hay un proceso. Tener únicamente un proceso por contenedor, la complejidad en las máquinas virtuales se reduce considerablemente. Levantar un contenedor Docker asegura tener la misma configuración, poseyendo una configuración determinada y que se comparte con todo el mundo.

La principal función de un contenedor Docker es aislar, permitiendo trabajar con una aplicación trabajando en el contenedor ejecutándose de forma aislada del resto de aplicaciones y pudiendo ejecutarse en cualquier máquina. Esta aplicación estará aislada y viajará en un contenedor, pudiendo ejecutarse en cualquier entorno.

3.4.1.-Instalación de Docker

Docker es multiplataforma y puede instalarse en cualquier sistema operativo, Linux, Windows o macOS. Puede ser descargado en su página oficial y, como hemos dicho, se encuentra preparado para ser instalado en cualquier plataforma, describiendo la instalación para cada sistema operativo de forma documentada.

Para la instalación de Docker en Windows 10 será necesario que la licencia sea Pro o Enterprise debido a que Docker utiliza actualmente Hyper-V para los contenedores, siendo antes utilizados por VirtualBox. En caso de no disponer tal licencia, podremos descargar la versión Docker Toolbox.

Para la documentación de Docker, utilizaremos el sistema operativo Ubuntu 17.10. El uso de Docker en los sistemas operativos es el mismo por lo que no hay de por qué preocuparse.

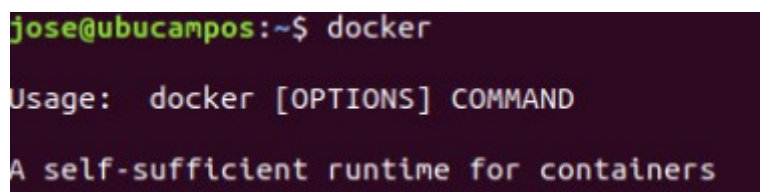
Para la instalación en Ubuntu, tendremos que lanzar los siguientes comandos:

```
sudo apt-get update  
sudo apt-get install docker.io
```

Para contar con Docker al iniciar el equipo, ingresaremos los siguientes comandos

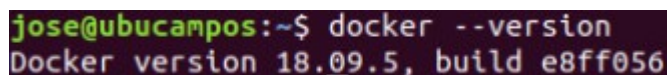
```
sudo systemctl start docker  
sudo systemctl enable docker
```

Podemos comprobar que Docker se encuentra efectivamente instalado si lanzamos el comando siguiente y devuelve una salida reconociendo el comando.



```
jose@ubucampos:~$ docker  
Usage:  docker [OPTIONS] COMMAND  
A self-sufficient runtime for containers
```

Pudiendo comprobar la versión de Docker con el siguiente comando:

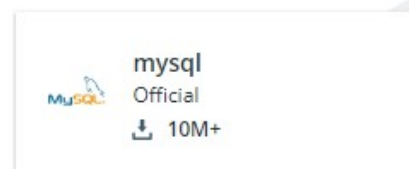


```
jose@ubucampos:~$ docker --version  
Docker version 18.09.5, build e8ff056
```


3.4.2.-Docker Hub y Docker Search

De forma similar a Vagrant, con Docker también encontramos repositorios desde los cuales podemos descargar imágenes ya creadas con las cuales trabajar de inmediato. La dirección es <https://hub.docker.com>. En la página, podemos realizar búsquedas para encontrar las imágenes que deseemos, pudiendo filtrar por imágenes oficiales o de un publicador verificado, arquitecturas, sistemas operativos, etc.

Official Images



Al acceder a una de las imágenes disponibles en el repositorio, accederemos a las instrucciones para realizar su instalación, *tags*, variantes, etc.; categorías que veremos más adelante.

Otra forma de acceder a estos repositorios es a través del comando siguiente con el cual pasando un argumento, podremos acceder a los nombres de las imágenes de Docker disponibles filtradas por el argumentos que hemos pasado.

```
jose@ubucampos:~$ sudo docker search apache
[sudo] contraseña para jose:
NAME                DESCRIPTION                STARS     OFFICIAL
httpd               The Apache HTTP Server Project  2498     [OK]
tomcat              Apache Tomcat is an open source implementati...  2412     [OK]
cassandra           Apache Cassandra is an open-source distribut...  984      [OK]
```

La salida del comando será la misma que al acceder al repositorio de Docker, apareciendo los contenedores de mayor a menor puntuación de los usuarios, apareciendo opciones como si el contenedor es oficial, etc.

Podemos realizar la descarga de imágenes disponibles mediante el comando. El comando buscará la imagen en el repositorio y la descargará en nuestro equipo.

```
docker pull <contenedor>
```

```
jose@ubucampos:~$ sudo docker pull ubuntu
[sudo] contraseña para jose:
Using default tag: latest
latest: Pulling from library/ubuntu
6abc03819f3e: Pull complete
05731e63f211: Pull complete
0bd67c50d6be: Pull complete
Digest: sha256:f08638ec7ddc90065187e7eabdfac3c96e5ff0f6b2f1762cf31a4f49b53000a5
Status: Downloaded newer image for ubuntu:latest
```

Para esta documentación, utilizaremos la descarga de un contenedor de Ubuntu. Podemos ver que en la siguiente imagen aparece lo que antes hemos mencionado como *tag*. Esta etiqueta hace referencia a la última versión del sistema operativo, situándose en la última disponible. Es decir, el comando siempre descargará la última versión disponible del contenedor a no ser que especifiquemos la versión. Si accedemos al repositorio, podemos observar las siguientes *tags* disponibles:

Supported tags and respective Dockerfile links

- 18.04 , bionic-20190515 , bionic , latest ([bionic/Dockerfile](#))
- 18.10 , cosmic-20190515 , cosmic ([cosmic/Dockerfile](#))

Por ejemplo, para descargar un contenedor de Ubuntu en su versión 16.04 tendremos que correr el comando anterior añadiendo dos puntos (:) y a continuación el número de la versión:

```
jose@ubucampos:~$ sudo docker pull ubuntu:16.04
16.04: Pulling from library/ubuntu
9ff7e2e5f967: Pull complete
59856638ac9f: Pull complete
6f317d6d954b: Pull complete
a9dde5e2a643: Pull complete
Digest: sha256:cad5e101ab30bb7f7698b277dd49090f520fe063335643990ce8fbd15ff920ef
Status: Downloaded newer image for ubuntu:16.04
```

Hay que destacar que las imágenes de Docker vienen provistas con lo necesario para ejecutar la aplicación, programa o sistema operativo que viene anexo a ellas, por lo que cualquier modificación que se haga en un contenedor, por ejemplo una actualización o la creación de un fichero, al levantar un nuevo contenedor utilizando la misma imagen, los cambios que se produjeron en el anterior no existirán en el nuevo. Si queremos tener una imagen que cumpla con ciertos criterios para nuestro entorno y levantar los contenedores siempre con esa estructura, tendremos que crear nuestra propia imagen. Simplemente, tendremos que ejecutar un comando apuntando al contenedor objetivo y asignarle un nombre:

```
docker commit <contenedor> <nombre>
```

Así, por ejemplo tenemos un contenedor con Ubuntu en el cual hemos instalado 'Vim', lo que ha requerido que el contenedor se actualizase y luego llevar a cabo la instalación. Levantar cualquier contenedor con la imagen de Ubuntu tendría la configuración básica con la cual hemos descargado la imagen y no la configuración que hemos realizado.

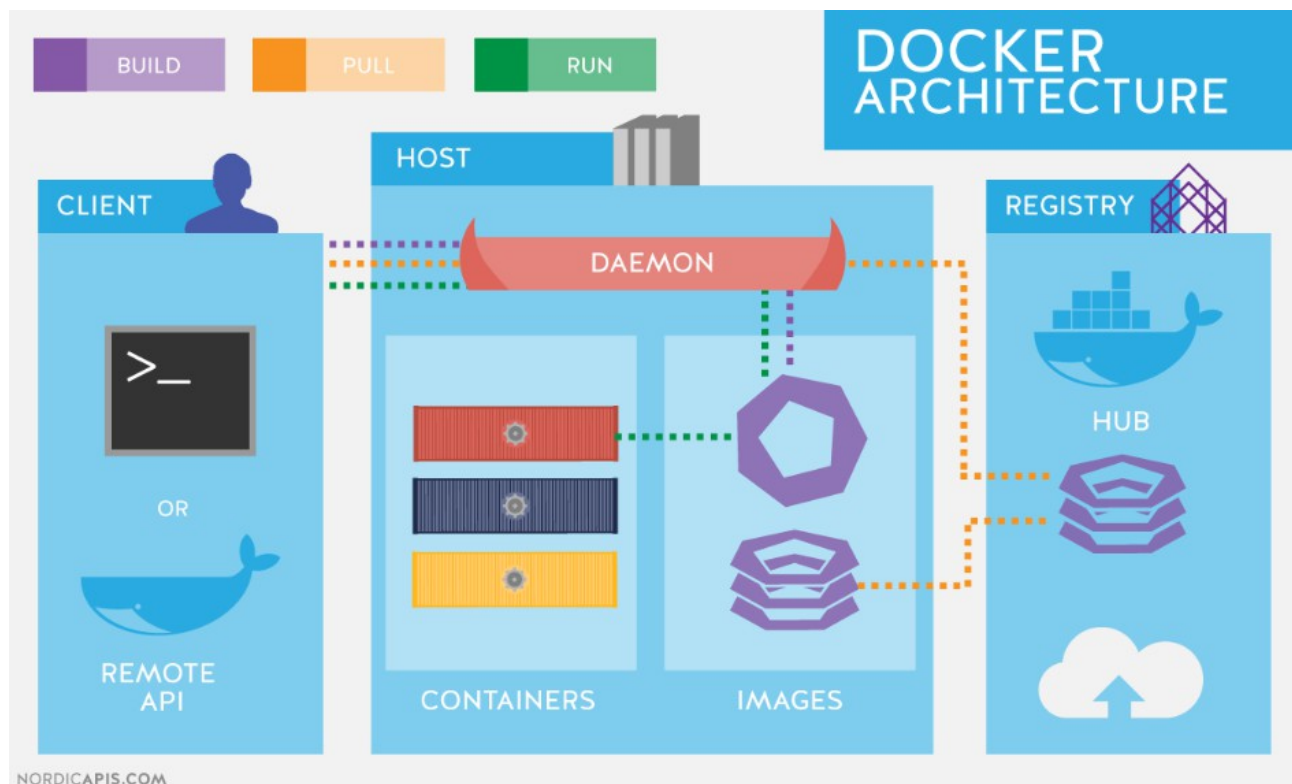
```
jose@ubucampos:~$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                NAMES
f1e53882d732   ubuntu    "bash"                  59 seconds ago Up 56 seconds        0.0.0.0:8080->80/tcp   ubuntu_vim
ba3cd1fc5c02   php:7.3-apache "docker-php-entrypoi..." About an hour ago Up About an hour      0.0.0.0:8080->80/tcp   phpapache

jose@ubucampos:~$ docker commit ubuntu_vim mi_ubuntu:1.0
sha256:7083def502a5420df0c337f5c04b1f34419e9941dbe6a24f4f6d73ed1732d8c9

jose@ubucampos:~$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
mi_ubuntu     1.0       7083def502a5   5 seconds ago 155MB
```

Al crear un nuevo contenedor utilizando esta imagen, ya tendrá por defecto la configuración que hemos realizado, que ha sido elemental.

3.4.3.-Trabajando con contenedores Docker



Representación visual de la arquitectura de Docker.

Descargar una imagen

- `docker pull <imagen>`

```
jose@ubucampos:~$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
Digest: sha256:f08638ec7ddc90065187e7eabdfac3c96e5ff0f6b2f1762cf31a4f49b53000a5
Status: Downloaded newer image for ubuntu:latest
```

Crear un contenedor

- `docker run <imagen>`

De no encontrar la imagen de forma local, Docker accederá a su repositorio y creará el contenedor con la imagen de mismo nombre.

```
jose@ubucampos:~$ docker run ubuntu
```

Listar imágenes en el repositorio local

- `docker images`

```
jose@ubucampos:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
php                  7.3-apache         a128d5060269       8 days ago         378MB
ubuntu               latest             7698f282e524       3 weeks ago        69.9MB
```

Mostrar los contenedores activos

- `docker ps`

```
jose@ubucampos:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
ba3cd1fc5c02       php:7.3-apache     "docker-php-entryp... 6 hours ago
```

Mostrar todos los contenedores, incluidos los parados

- `docker ps -a`

```
jose@ubucampos:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
ba81a48f7f81       ubuntu             "/bin/bash"        6 minutes ago
```

Crear contenedor interactivo

- `docker run -it <imagen>`

Permite acceder a un contenedor de manera interactiva si lo permite.

```
jose@ubucampos:~$ docker run -it ubuntu
root@ba81a48f7f81:/#
```

Acceder a contenedor parado

- `docker start -i <ID_contenedor>`

```
jose@ubucampos:~$ docker start -i ba81
root@ba81a48f7f81:/#
```

Crear contenedor en segundo plano

- `docker run -d <imagen>`

```
jose@ubucampos:~$ docker run -d nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
743f2d6c1f65: Already exists
d6c2f01b1dae: Pull complete
d4da6ff1b555: Pull complete
Digest: sha256:12db363acf5b2d2f9f5fed240e228a04692bdac68402430fbd2f720c3a967d01
Status: Downloaded newer image for nginx:latest
83d6ce14efedfe65a8169b4109d1baa4dbfa81df370f2f542547f7bcaa5dd195
jose@ubucampos:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
83d6ce14efed        nginx              "nginx -g 'daemon of..." 28 seconds ago
```

Crear contenedor con nombre

- `docker run -it --name <nombre> <imagen>`

```
jose@ubucampos:~$ docker run -d --name mi_nginx nginx
e7f089e78f32e5016d0b2fd1c4a910a0199253251f4f8702edb28d0888bb826b
```

Acceder a contenedor en ejecución en segundo plano

- `docker exec -it [<ID_contenedor>],<nombre_contenedor> bash`

```
jose@ubucampos:~$ docker exec -it mi_nginx bash
root@e7f089e78f32:/#
```

Salir de un contenedor pero no detenerlo

Habrá que ejecutar la siguiente combinación de teclas: Control + P y luego Control + Q. Al ejecutar satisfactoriamente la combinación de teclas, es posible que aparezca el mensaje por terminal 'read escape sequence'. Si comprobamos los contenedores en ejecución encontraremos que el contenedor del que hemos salido todavía sigue en marcha. Podemos ver que por el ID se trata del mismo contenedor.

```
jose@ubucampos:~$ docker exec -it mi_nginx bash
root@e7f089e78f32:/# read escape sequence
jose@ubucampos:~$ docker ps
CONTAINER ID        IMAGE               COMMAND
e7f089e78f32        nginx              "nginx -g 'daemon of..."
```

3.4.4.-Dockerfile

Con el fichero Dockerfile podemos automatizar el proceso de creación de imágenes sin tener que acceder a los contenedores y manualmente configurarlos paso a paso, de forma similar como se hizo con el Vagrantfile. El fichero Dockerfile tiene una gran cantidad de opciones y vamos a ver las opciones más importantes.

Como hicimos para trabajar con Vagrant, lo más cómodo es trabajar en un entorno de directorios, porque solo podremos tener un Dockerfile por directorio. En un directorio crearemos un fichero llamado Dockerfile sin extensión y lo editaremos.

Partiremos siempre de una imagen como base, que va anexada al elemento 'FROM', y luego indicaremos mediante 'RUN' los comandos que queremos que se ejecuten dentro del contenedor. Estos comandos deben ser declarativos y no pueden interactuar con el usuario, por lo tanto tendremos que salvar esas excepciones, como cuando el instalador de algún paquete requiere la confirmación por parte del usuario. Así, para la imagen del punto anterior en la cual instalamos 'Vim' tendremos un Dockerfile como el siguiente.

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y vim
```

Cuando tengamos lista el fichero, estaremos listos para lanzar el comando para crear la imagen a partir del Dockerfile.

```
docker build <directorio_al_fichero_Dockerfile>
```



```
jose@ubucampos:~/laboratorio/mis_imagenes$ docker build -t ubuntu_vim .  
Sending build context to Docker daemon 2.56kB  
Step 1/3 : FROM ubuntu  
--> 7698f282e524  
Step 2/3 : RUN apt-get update
```

Podemos comprobar que la imagen la tenemos disponible en nuestro catálogo de imágenes para crear nuevos contenedores y que efectivamente podemos lanzar 'Vim' sin necesidad de realizar una instalación porque partimos de una imagen que ya está configurada como tal.

```
jose@ubucampos:~/laboratorio/mis_imagenes$ docker images  
REPOSITORY          TAG             IMAGE ID         CREATED          SIZE  
ubuntu_vim          latest          095d043ad363    About a minute ago 155MB
```

```
jose@ubucampos:~/laboratorio/mis_imagenes$ docker run -it --name vim_docker ubuntu_vim bash  
root@a7b2e13e6f8c:/# vi file  
root@a7b2e13e6f8c:/#
```

Evidentemente, el fichero Dockerfile se puede volver muy complejo, pero básicamente su función se basa en agregar comandos y opciones a parámetros que podemos conocer a partir de la documentación.

- **CMD:** directiva que indica el comando por defecto del contenedor. Una vez terminado de ser arrancado el contenedor, esta directiva indica qué comando se va a ejecutar al arrancar el contenedor.

```
FROM ubuntu  
  
RUN apt-get update  
RUN apt-get install -y vim  
  
CMD echo "Bienvenido al contenedor Vim"
```

```
jose@ubucampos:~/laboratorio/mis_imagenes/ubuntu_vim$ docker build -t imagev1:1.0 .  
Sending build context to Docker daemon 2.048kB
```

```
jose@ubucampos:~/laboratorio/mis_imagenes/ubuntu_vim$ docker run -it --name vim imagev1:1.0  
Bienvenido al contenedor Vim
```

- **ENTRYPOINT:** directiva que permite lanzar un comando al iniciar un contenedor de forma blindada, de forma que siempre se ejecute y no puede ser evitado. La diferencia con 'CMD' es que con 'ENTRYPOINT' no puede ser alterado.
- **WORKDIR:** permite definir el directorio de trabajo del Dockerfile a partir de este punto. Los comandos que se ejecuten tras definir un 'WORKDIR' se realizarán en ese ámbito.

- **COPY:** copia elementos de la máquina anfitrión al contenedor de forma relativa desde el contexto donde se encuentra el fichero Dockerfile.
- **ADD:** copia elementos de la máquina anfitrión al contenedor. La directiva 'ADD' permite copiar ficheros de extensión .tar y desempaquetarlos mientras que 'COPY' no permite realizar tal operación.

```
FROM ubuntu

RUN apt-get update
RUN apt-get install -y vim

RUN mkdir /datos
WORKDIR /datos
RUN touch fichero1.txt

COPY index.html .

ADD f.tar .
```

1. Usar imagen base de 'ubuntu'.
2. Actualizar el contenedor, habilitando la opción de instalación de paquetes.
3. Instalación de 'Vim'.
4. Crear el directorio '/datos'.
5. Movernos al directorio '/datos'.
6. Crear allí el fichero 'fichero1.txt'.
7. Copiar el fichero 'index.html' del directorio donde se encuentra el Dockerfile en la máquina anfitrión al directorio actual del contenedor, '/datos'.
8. Copiar el fichero 'f.tar' del directorio donde se encuentra el Dockerfile en la máquina anfitrión al directorio actual del contenedor, '/datos'.

- **ENV:** permite definir variables de entorno.
- **ARG:** permite definir variables de entorno como argumentos en el momento de construcción del contenedor. Si existe algún elemento 'ARG' en el Dockerfile y no se ha completado en el momento del 'docker build', se producirá un error. Se trata de una variable que espera recibir un valor definido por el usuario con '--build-arg'.

```
ENV dir=/data dir1=/data1
RUN mkdir $dir && mkdir $dir1

ARG dir2
RUN mkdir $dir2
```

1. Se definen las variables \$dir y \$dir1.
2. Se ejecuta el comando 'mkdir' con las variables, creando dos directorios.
3. ARG solicitará en el momento de la construcción del contenedor una variable como argumento.
4. Se crea un directorio en la localización que se pasa.

```
jose@ubucampos:~/laboratorio/mis_imagenes/ubuntu_vim$ docker build -t image:1.0 --rm --build-arg dir2=/data2 .
Sending build context to Docker daemon 2.048kB
```

- **EXPOSE:** permite exponer puertos para ser utilizados. Esta opción no excluye al parámetro '-p' en el momento de la construcción del contenedor. Es una herramienta para que el usuario final conozca cuál es el puerto expuesto.
- **VOLUME:** crea un volumen de forma automática que podrá ser enlazado por otros contenedores posteriormente.

Ahora bien, si el contenedor lo queremos compartir con nuestro entorno de trabajo o compañeros, podemos subirlo al propio repositorio de Docker a través del comando 'login' e introduciendo posteriormente nuestras credenciales.

```
docker login
```

```
jose@ubucampos:~$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't
have a Docker ID, head over to https://hub.docker.com to create one.
Username: camposperezjosep
Password:
Login Succeeded
```

Para subir nuestra imagen al repositorio de Docker y no ser una autoridad oficial, tendremos que valernos del siguiente comando para que nuestra imagen cumpla con los requisitos de nombre para que se encuentre en el repositorio de Docker.

```
docker image tag <imagen>:<tag> <nombre_usuario>/<nombre_de_la_imagen>
```

```
jose@ubucampos:~$ docker image tag image:3.0 camposperezjosep/1er_imagen
jose@ubucampos:~$ docker images
```

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|-----------------------------|--------|--------------|----------------|-------|
| camposperezjosep/1er_imagen | latest | 53304d2dc82a | 30 minutes ago | 192MB |

Una vez tengamos la imagen lista con nuestro nombre de usuario como parte del nombre de la imagen, ya estaremos listos para realizar la operación de subida al repositorio mediante:

```
docker push <imagen>
```

```
jose@ubucampos:~$ docker push camposperezjosep/1er_imagen
The push refers to repository [docker.io/camposperezjosep/1er_imagen]
577881a6bc9b: Preparing
```

Una vez el comando haya acabado, podemos comprobar que está subido al repositorio de Docker.



camposperezjosep/1er_imagen

By [camposperezjosep](#) • Updated 3 minutes ago

Container

```
docker pull camposperezjosep/1er_imagen
```



3.4.5.-Redes en Docker

Ya sabemos que los contenedores de Docker se encuentran totalmente aislados, pero en muchas ocasiones tendremos que disponer de contenedores que sean accesibles, ya sea porque la aplicación o proceso que ejecutan necesitan conexión con el exterior. Por defecto, los puertos en Docker son totalmente privados, por lo que no pueden ser accedidos desde fuera. Por lo tanto, tendremos que exponer los puertos y a la vez, ser mapeados con un puerto de la máquina anfitrión donde se está ejecutando el contenedor.

Básicamente, lo que tendremos que hacer será una redirección de puertos de la máquina anfitrión al contenedor. Para ello, podremos valernos del siguiente comando:

```
docker run -d -P <imagen>
```

La opción '-P' supone que todos los puertos del contenedor se vuelvan públicos. De forma predefinida, las imágenes de Docker que descargamos pueden tener algún puerto ya redireccionado con la máquina. Este puerto lo podemos conocer accediendo a la información de la imagen y más tarde inspeccionándola. En la siguiente imagen podemos ver que la imagen de Apache ya tiene, por defecto, abierto el puerto 32768 para los contenedores que se creen con esta imagen.

```
jose@ubucampos:~/laboratorio/apache$ docker ps
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS |
|--------------|-------|--------------------|----------------|---------------|-----------------------|
| 5381d95d4e0b | httpd | "httpd-foreground" | 36 seconds ago | Up 33 seconds | 0.0.0.0:32768->80/tcp |



It works!

En el caso de que queramos definir un puerto específico, tendremos que realizar el siguiente comando. Con el parámetro '-p' podremos especificar el juego de puertos que queremos definir, siendo el primero el del contenedor y el segundo el de la máquina anfitrión.

```
docker run -d -p XX:YY <imagen>
```

De esta forma ya podremos definir el puerto por el cual va a escuchar el contenedor y con el cual desde nuestra máquina acceder al contenedor.

```
jose@ubucampos:~/laboratorio/apache$ docker run -dp 8080:80 httpd  
f108cd688ec21a92d6adaceb72c2a46e8162077d1e13173df3a75d9e6398383c
```



It works!

Todo esto se basa en que Docker mantiene una arquitectura de red que permite gestionar las distintas redes que podemos utilizar dentro de Docker para que estas redes se conecten a cada contenedor. En este punto, debemos conocer el comando:

```
docker network
```

Con el cual podemos recuperar información de las conexiones disponibles, conocer detalles, crear, etc.

```
jose@ubucampos:~/laboratorio/apache$ docker network

Usage:  docker network COMMAND

Manage networks

Commands:
  connect    Connect a container to a network
  create     Create a network
  disconnect Disconnect a container from a network
  inspect    Display detailed information on one or more networks
  ls         List networks
  prune      Remove all unused networks
  rm         Remove one or more networks
```

Por ejemplo, para listar las conexiones disponibles que tenemos en nuestro sistema, podemos usar comando 'ls' para obtener un resultado parecido al siguiente:

```
jose@ubucampos:~/laboratorio/apache$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
746cf931d418        bridge             bridge              local
bbd965d6ea57        host               host                local
a5bebe4b2c15        none              null                local
```

Existen tres tipos de redes:

- **Bridge:** los contenedores se van a comunicar hacia el exterior a través de la red física que utiliza la máquina anfitrión. Es la red predefinida por los contenedores.
- **Host:** los contenedores van a poder conectarse únicamente con la máquina anfitrión. Contenedores autónomos e independientes.
- **None:** los contenedores se encuentran totalmente aislados.

Podemos comprobar la información de alguna de estas redes a través del siguiente comando. Con la salida podemos ver la red a la que pertenece la red así como los objetos asociados a la red. Hay que destacar que solamente aparecerán los contenedores que se encuentren activos en el momento de la ejecución. Cualquier contenedor apagado no aparecerá en la lista.

```
docker network inspect <nombre_de_la_red>
```

```
jose@ubucampos:~/laboratorio$ docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "268d52a29173af1a05a3ebbd039496f5d7a6beb176556a26ab339911df5887f1",
    "Created": "2019-06-08T22:06:07.949646105+02:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    }
  }
],
{
  "Containers": {
    "5381d95d4e0b7da8377fb72a02f70b507cda79b9f58c0e48fbd6fa888bb1d110": {
      "Name": "pedantic_hopper",
      "EndpointID": "e50559860248fd9ffdf59be2c82a62b227a23de46dd5199f7537bc788dbd0c16",
      "MacAddress": "02:42:ac:11:00:03",
      "IPv4Address": "172.17.0.3/16",
      "IPv6Address": ""
    },
    "f108cd688ec21a92d6adaceb72c2a46e8162077d1e13173df3a75d9e6398383c": {
      "Name": "tender_noyce",
      "EndpointID": "6e996147970abc5f647f3144ca902c7fe087bd071994acf59fc19aee84812c30",
      "MacAddress": "02:42:ac:11:00:02",
      "IPv4Address": "172.17.0.2/16",
      "IPv6Address": ""
    }
  }
},
]
```

En el apartado donde aparecen los contenedores, podemos ver que Docker ya asigna unas IPs a los contenedores dentro de la red que maneja Docker y que efectivamente tenemos conexión con la máquina anfitrión a través de un *ping*.

```
jose@ubucampos:~/laboratorio$ ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.111 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.167 ms
^C
--- 172.17.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1022ms
```

Opcionalmente, podemos crear una nueva red para nuestros contenedores. Hay que destacar que esta red nueva que creemos tendrá muchas más bondades que las redes que existen por defecto porque por la propia naturaleza de Docker, se encuentran muy limitadas para hacer de su sistema del aislamiento un punto muy notable.

Para crear una red de forma sencilla, sin muchos argumentos y definiciones, podemos correr el siguiente comando.

```
docker network create <nombre_de_la_red>
```

```
jose@ubucampos:~/laboratorio$ docker network create red1
9803bce2b2afdbe47a3870ac9df8b45d1a37b1688fc48694f65270fde19fece7
jose@ubucampos:~/laboratorio$ docker network ls
```

| NETWORK ID | NAME | DRIVER | SCOPE |
|--------------|--------|--------|-------|
| 268d52a29173 | bridge | bridge | local |
| bbd965d6ea57 | host | host | local |
| a5bebe4b2c15 | none | null | local |
| 9803bce2b2af | red1 | bridge | local |

Si inspeccionamos esta red, podremos ver que Docker ha asignado un valor consecutivo a las redes que ya se encuentran en su sistema.

```
"Subnet": "172.18.0.0/16",
"Gateway": "172.18.0.1"
```

Para enlazar un contenedor a una red concreta, sencillamente al crear el contenedor definiremos la red a la cual debe realizar la conexión utilizando el parámetro '--network'. Hay que destacar que Docker genera un DNS interno cuando utilizamos la posibilidad de poner nombres a los contenedores, de forma que realizar un *ping* a otro contenedor utilizando su nombre lo resolverá internamente.

```
jose@ubucampos:~/laboratorio$ docker run -it --name ubuntu bash --network red1 ubuntu
root@3ad4fcb602c5:/#
```

También podemos realizar la conexión “en caliente” de un contenedor a una red, usando el siguiente comando, agregando la IP a su configuración, pudiendo trabajar con contenedores de una red u otra.

```
docker network connect <red> <contenedor>
```

De la misma forma, podríamos desconectar un contenedor de una red a través de:

```
docker network disconnect <red> <contenedor>
```

En este punto, ya sabemos crear redes, conectar contenedores de forma dinámica en caliente o al crear un contenedor, y ahora es el momento de conocer cómo hacer para enlazar estos contenedores y se puede realizar por dos medios: utilizando las redes que ya trae consigo Docker (bridge, host, none) o con las personalizadas que creamos nosotros mismos.

Para enlazar dos contenedores, lo podemos hacer a través de el siguiente comando:

```
docker run -it --link <contenedor>:<alias> <imagen>
```

```
jose@ubucampos:~/laboratorio$ docker run -it --name b3 --link b1:contenedor1 busybox
/ # ping b1
PING b1 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.227 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.218 ms
```

La resolución del enlace es unidireccional, por lo que si hemos establecido un enlace entre A y B, A puede resolver su dirección pero B contra A no puede.

```
/ # jose@ubucampos:~/laboratorio$ docker attach b1
/ # ping b3
ping: bad address 'b3'
```

En el caso de querer enlazar contenedores con redes que hemos creado nosotros mismos, los contenedores que formen parte de la red ya han configurado sus tablas para que se puedan reconocer, de modo que se puede establecer la conexión entre ellas por su nombre o alias.

Para demostrar su uso, crearemos un contenedor que actúe de MySQL que actúe como servidor y otro contenedor que actúe como cliente. Básicamente, en el contenedor servidor se encontrarán las bases de datos y desde el otro cliente se establecerá la conexión y para ello es necesario que exista un enlace entre ellos. El punto más importante en este punto es que el parámetro de conexión de red coincidan, los demás parámetros son los propios de MySQL.

```
jose@ubucampos:~$ docker run -d --name mysql_server --network red1 -e MYSQL_ROOT_PASSWORD=secret mysql
e3b753c0b72bb15dbe639ecb831804fe154e64e7a86aa1cc02b0f73c2a31017e
jose@ubucampos:~$ docker exec -it mysql_server bash
root@e3b753c0b72b:/# mysql -u root -p
Enter password:
```

```
jose@ubucampos:~$ docker run -it --name mysql_client --network red1 mysql bash
root@903d0a435db5:/# mysql -h mysql_server -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
```

En la segunda imagen podemos ver que al establecer la conexión, indicamos el parámetro '-h' haciendo referencia al primer contenedor que sirve como contenedor, resolviendo el nombre y estableciendo la conexión correctamente.

3.4.6.-Volúmenes

Los volúmenes es el mecanismo por el cual podemos hacer persistir la información de los contenedores y sus datos cuando estos se desactiven. Es un componente que administra completamente Docker y no los gestiona el sistema operativo, por lo que son más fáciles de respaldar o migrar, son multiplataforma, se pueden compartir de forma segura entre varios contenedores, almacenar en hosts remotos o en entornos cloud, cifrar el contenido, etc. En otras palabras, los ficheros y datos que tenemos en el contenedor provienen de un elemento externo al contenedor y todo cambio y edición en los datos en ese elemento externo repercute en la información del contenedor.

```
jose@ubucampos:~$ docker volume

Usage:  docker volume COMMAND

Manage volumes

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused local volumes
  rm          Remove one or more volumes
```

Podemos ver los volúmenes creados y a qué contenedores se encuentran asociados. Los volúmenes se encuentran en el directorio de Docker '/var/lib/docker/volumes' y se encuentran identificados por un ID y asociados a un directorio externo que se encuentran totalmente sincronizados.

Para demostrar su uso, vamos a compartir un recurso entre la máquina anfitrión y el contenedor con un directorio de la máquina anfitrión concreto. Tendremos que usar el parámetro '-v' para indicar primero el directorio de la máquina anfitrión, dos puntos y el directorio en el contenedor. De no existir tales directorios se crearán si se cumplen los requisitos de permisos. Si probamos de crear ficheros en uno de los lados, podremos ver que aparece en el otro lado y viceversa.

```
jose@ubucampos:~/laboratorio/dir1$ docker run -it -v /home/jose/laboratorio/dir1:/dir1 --name ubuntu_volumen ubuntu
root@515b548d0251:/# exit
```

```
root@515b548d0251:/dir1# touch fichero1
```

```
jose@ubucampos:~/laboratorio/dir1$ ll
total 8
drwxr-xr-x 2 jose jose 4096 jun  9 10:52 ./
drwxr-xr-x 5 jose jose 4096 jun  9 10:49 ../
-rw-r--r-- 1 root root   0 jun  9 10:52 fichero1
```

Podemos entonces decir, que se monta un enlace entre el directorio de la máquina anfitrión y el directorio del contenedor.

Un paso más allá resultaría de querer compartir volúmenes entre contenedores de Docker. Básicamente, tendremos un contenedor principal que utiliza un directorio de la máquina anfitrión como volumen, y tantos otros contenedores como queramos que apuntarán al volumen del contenedor principal. Todo ello, se resume en utilizar el parámetro '-volumes-from <contenedor>'.

```
jose@ubucampos:~$ docker run -it --name ub2 --volumes-from ub1 ubuntu bash
root@7f48edff05ff:/# cd d
datos/ dev/
root@7f48edff05ff:/# cd datos/
root@7f48edff05ff:/datos# touch fichero1.txt
```

```
root@2a6e3231c0da:/datos# ls
fichero1.txt
```

Sobre los volúmenes hay que señalar que cuando un contenedor queda huérfano, ya no funciona. Nos referimos a que no exista ningún contenedor que apunte al volumen, persistiendo el volumen pero inutilizado, debido a que no puede ser asociado a ningún contenedor porque no existe ninguno que apunte a él. En otras palabras, el contenedor sigue existiendo, pero ya no podría ser asociado a ningún contenedor. La solución a este tipo de problemas sería la de acceder al volumen que ha quedado obsoleto, tomar su contenido y al montar un nuevo contenedor con un nuevo volumen, incluir los ficheros que hemos tomado del anterior contenedor al nuevo.

3.5.-Comparación entre Vagrant y Docker

Mientras que en Vagrant se tiene un aislamiento total puesto que se pueden gestionar todos y cada uno de los aspectos de dicha máquina como si fuese un sistema real como por ejemplo el tamaño del disco, memoria, procesadores o dirección IP, en Docker debido al uso de contenedores, el aislamiento es bastante menor puesto que comparte el *kernel* con la máquina nativa. Cabe señalar que la cantidad de aislamiento se traduce en una mayor seguridad aunque tiene diversas desventajas como los puntos que ahora indicaremos.

Aunque tanto las máquinas virtuales como los contenedores utilizan los recursos Hardware de la máquina anfitrión, las máquinas virtuales utilizan muchos más recursos debido a que ejecutan sistemas virtualizados completos mientras que en el caso de los contenedores, únicamente se utilizan aquellos recursos necesarios para ejecutar las aplicaciones.

El tiempo necesario tanto para crear como para ejecutar máquinas virtuales es mucho más elevado que en el caso de los contenedores, debido también a que son sistemas completos. Además, el tamaño virtualizado es muy superior en el caso de las máquinas debido al mismo motivo.

Capítulo 4. Sistemas operativos orientados a contenedores

Tras el éxito y popularización de los contenedores han surgido sistemas operativos modernos diseñados para funcionar de manera óptima con un ecosistema de contenedores. Estos sistemas proporcionan la mínima funcionalidad requerida para implementar aplicaciones, junto con propiedades de actualización y restablecimiento.

Los sistemas operativos orientados a contenedores han evolucionado el modelo tradicional. Aquí, las aplicaciones son binarios autónomos que se pueden mover en su entorno de contenedor. Esta tecnología se conoce también como virtualización del sistema operativo en el que su núcleo permita la existencia de múltiples instancias o contenedores de espacio de usuario aisladas, en lugar de una única.

4.1.-CoreOS

Es un sistema operativo ligero de código abierto de la compañía de mismo nombre basada en el núcleo de Linux y lanzado en 2013. Proporciona las características necesarias para ejecutar contenedores con un enfoque práctico para las actualizaciones del sistema operativo.



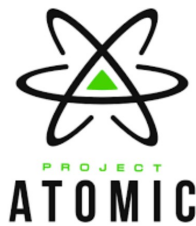
Está diseñado para implementar una aplicación distribuida en un clúster de nodos. De esta forma, se encarga de proporcionar la infraestructura necesaria para los despliegues en clúster, automatización, seguridad, fiabilidad y escalabilidad. Ofrece las funcionalidades mínimas necesarias para la implementación de aplicaciones dentro de contenedores de Software, junto con mecanismos incorporados para el descubrimiento de servicios y el intercambio de configuración.

CoreOS no proporciona un gestor de paquetes, por lo que necesita que todas las aplicaciones se ejecuten dentro de sus contenedores. Para ello utiliza Docker y contenedores Linux subyacentes de virtualización para la ejecución de múltiples sistemas aislados en un único sistema anfitrión de control, que sería la instancia CoreOS.

4.2.-Otros sistemas operativos

Otros sistemas orientados a contenedores son Red Hat Enterprise Linux, Project Atomic y Snappy Ubuntu Core.

- **Project Atomic:** facilita la arquitectura centralizada en la aplicación proporcionando una solución para desplegar aplicaciones basadas en contenedores de forma rápida y confiable. La actualización atómica y la reversión de aplicaciones y anfitriones permiten la implementación de pequeñas mejoras frecuentes.



- **Snappy Ubuntu Core:** usa una imagen de servidor mínima con las mismas bibliotecas que el sistema operativo Ubuntu actual. Su enfoque ágil es más rápido, más fiable y permite ofrecer mayores garantías de seguridad para aplicaciones y usuarios. Las aplicaciones se pueden actualizar de forma atómica y revertirse si es necesario, pensando en contenedores.

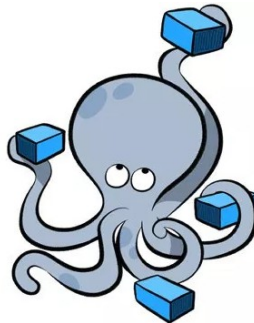


- **Red Hat Enterprise Linux:** también conocido por sus siglas RHEL, es una distribución comercial de GNU/Linux desarrollada por Red Hat. Es la versión comercial basada en Fedora que a su vez está basada en el anterior Red Hat Linux.

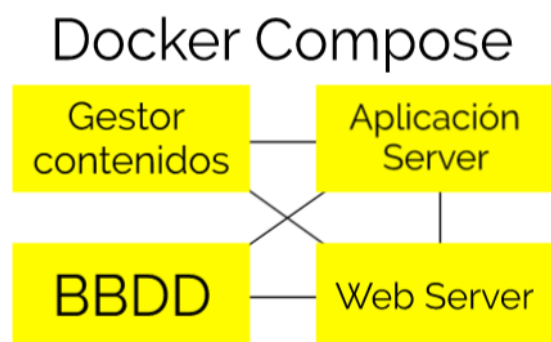
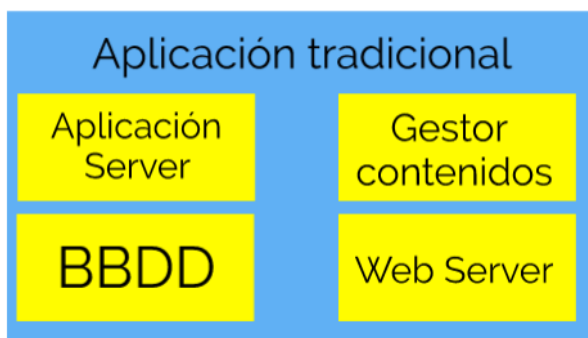


Capítulo 5.-Automatización de la configuración

5.1.-Docker Compose



Como vimos en el apartado anterior, Docker permite el despliegue en micro servicios. Por lo tanto, una aplicación tradicional que está funcionando a partir de diversas aplicaciones, con Docker lo podemos transformar en un entorno de microservicios donde existen varios contenedores que ofrecen las mismas funcionalidades en conjunto que la primera aplicación.



Cuanto más contenedores tenemos y más flexibles queremos que sean, para establecer las conexiones entre tales, volúmenes, etc.; se hace cada vez más tediosa la configuración de tales contenedores y de ello nació Docker Compose. Se trata de un programa que orquesta servicios o componentes de contenedores a través de un fichero YAML que contiene las instrucciones para establecer las relaciones y enlaces entre los contenedores. En resumen, Docker Compose es un frontal contra Docker que permite gestionar este tipo de arquitectura de forma más simple.

YAML es un formato de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl; así como el formato para correos electrónicos.

5.1.1.-El fichero docker-compose.yml

Podemos decir que este fichero es el Dockerfile o el Vagrantfile para Docker Compose. Se trata de un fichero escrito en un lenguaje de marcado escrito en YAML que permite definir las características de nuestro entorno de forma muy predecible.

```
1  version: '3'
2  services:
3    web:
4      build: .
5      ports:
6        - "5000:5000"
7      volumes:
8        - ./code
9        - logvolume01:/var/log
10     links:
11       - redis
12     redis:
13       image: redis
14   volumes:
15     logvolume01: {}
```

Un fichero docker-compose.yml tiene el siguiente aspecto: se define la versión, que tiene que corresponderse con la versión que estemos utilizando en Docker y que puede consultarse en la documentación de Docker. A continuación, le sigue cada uno de los contenedores: tenemos el contenedor 'web' que se va elaborar a partir de un Dockerfile que se encuentra en el mismo directorio del docker-compose.yml (de ahí el punto .), y se definen los puertos y volúmenes, y el enlace con el otro contenedor, 'redis' que va a ser construido utilizando la imagen 'redis' que bien puede encontrarse de manera local o accediendo al repositorio de Docker.

A modo de introducción y antes de enlazar contenedores, crearemos un único contenedor a partir de un docker-compose.yml. Valga decir que al tener que utilizar un fichero que debe tener un nombre único tendremos que valernos de un entorno de directorios. Tenemos el siguiente fichero docker-compose.yml, en el cual decimos que vamos a construir un contenedor utilizando la imagen de 'nginx' y que el puerto 8080 del *host* redirige al puerto 80 del contenedor. Para levantar el contenedor, utilizamos el comando siguiente, utilizando el parámetro '-d', como en Docker, para que permanezca en segundo plano, o -it para modo interactivo.

```
jose@ubucampos:~/dockercompose$ cat docker-compose.yml
version '3'
services:
  nginx:
    image: nginx
    ports:
      - "8080:80"
```

```
docker-compose up
```

La ejecución del comando podemos extraer información que ya conocemos, como la creación de una red personalizada y el nombre que ha dado Docker Compose al contenedor, que se realiza de acuerdo al directorio donde se lance el comando, por lo tanto es una buena práctica que el directorio tenga un nombre relacionado con la orquestación que vamos a realizar.

```
jose@ubucampos:~/dockercompose/nginx$ docker-compose up
Creating network "nginx_default" with the default driver
```

```
jose@ubucampos:~/dockercompose/nginx$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                    NAMES
c9ab9943ed75   nginx    "nginx -g 'daemon of..." 6 minutes ago  Up 45 seconds  0.0.0.0:9000->80/tcp      nginx_nginx_1
```

Conociendo las opciones que tiene Docker, muchas de ellas son válidas para Docker Compose:

```
jose@ubucampos:~/dockercompose/nginx$ docker-compose ps
Name                Command             State              Ports
-----
nginx_nginx_1       nginx -g daemon off; Up                  0.0.0.0:9000->80/tcp
```

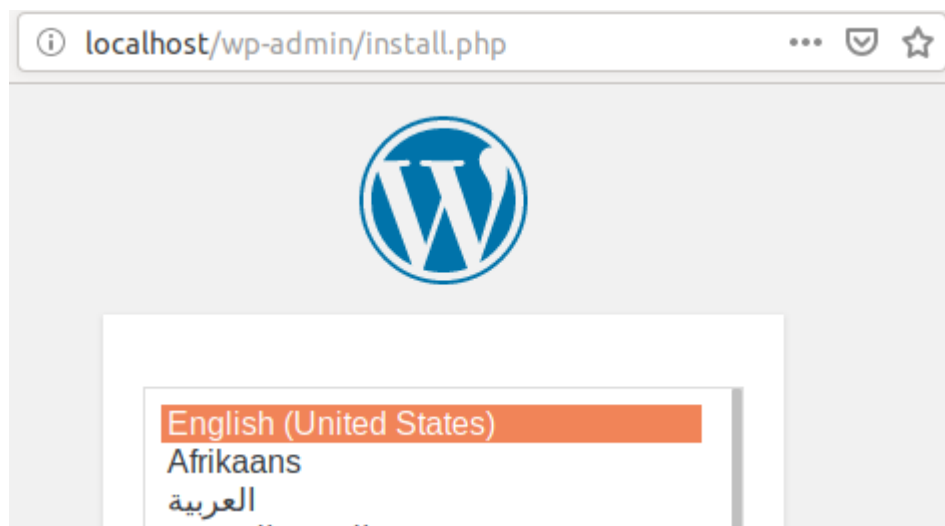
De la misma forma, muchas opciones que hemos visto con Docker son muy similares a las opciones que tenemos con Docker Compose y no vamos a abundar en ellas por ser muy similares, siempre teniendo en cuenta que trabajamos en la jerarquía de directorios y con el fichero docker-compose.yml.

A modo de ejemplo, tenemos el siguiente fichero docker-compose.yml que establecerá un juego de contenedores orquestados para levantar un servicio de Wordpress junto con MySQL (se podrá encontrar en el repositorio de GitHub del trabajo).

```
1  version: '3'
2  services:
3    wordpress:
4      image: wordpress
5      environment:
6        WORDPRESS_DB_HOST: dbserver:3306
7        WORDPRESS_DB_PASSWORD: secretpwd
8      ports:
9        - 80:80
10     depends_on:
11       - dbserver
12   dbserver:
13     image: mysql:5.7
14     environment:
15       MYSQL_ROOT_PASSWORD: secretpwd
16     ports:
17       - 3306:3306
```

Levantando este servicio se obtendrán dos contenedores: el primero que utilizará la imagen de Wordpress del repositorio de Docker estableciendo como variables de entorno el nombre del host y la contraseña, comunicando con la máquina anfitrión a través del puerto 80 y luego la cláusula 'depends_on'. Esta cláusula determina que como requisito, este contenedor depende del que se menciona, y ya se encarga Docker Compose de preparar primero el contenedor definido antes que levantar el de Wordpress, por lo tanto el orden en el que aparecen los contenedores definidos no es una garantía del orden de levantamiento de los contenedores. Y si seguimos el orden del fichero, se creará un nuevo contenedor con MySQL con la imagen del repositorio de Docker y utilizando la variable de entorno que Wordpress y el puerto típico de MySQL, 3306.

```
jose@ubucampos:~/dockercompose/wordpress$ docker-compose up -d
Starting wordpress_dbserver_1 ...
Starting wordpress_dbserver_1 ... done
Starting wordpress_wordpress_1 ...
Starting wordpress_wordpress_1 ... done
```



También es viable utilizar ficheros Dockerfile junto con Docker Compose. Para utilizar esta simbiosis, en el apartado 'image' del contenedor tendremos que sustituir por la cláusula 'build' y poniendo a continuación la ruta al fichero Dockerfile, de tal forma que utilizará la configuración ya realizada de un contenedor sin tener que abundar y repetirla en Docker Compose.

5.2.-Ansible

Ansible es un software que automatiza el aprovisionamiento de software, la gestión de configuraciones y el despliegue de aplicaciones y clasificado como una herramienta de orquestación. Permite gestionar los servidores, configuraciones y aplicaciones de forma robusta, sencilla y paralela. Ansible gestiona sus diferentes nodos con SSH. Solo requiere Python en el servidor remoto para ser utilizado, y YAML para describir las acciones a realizar y las configuraciones que se deben propagar.



En cuanto a la arquitectura de Ansible tenemos el servidor desde el cual comienza la orquestación y el nodo que es manejado por el controlador por SSH. Lo más normal es que exista un solo controlador y diversos nodos, como un nodo de bases de datos, nodo web, etc.

5.2.1.-Instalación de Ansible

Ansible está disponible para mac OS y Linux pero no para Windows.

Para esta documentación utilizaremos los sistemas virtualizados para utilizar Ansible. Partiremos de una máquina virtual que servirá como controlador que tendrá el control sobre otras máquinas. Puesto que ya sabemos utilizar Vagrant, lo utilizaremos para crear el Vagrantfile quedando tal como el siguiente para nuestro entorno:

```
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "bento/ubuntu-16.04"
  config.vm.network "private_network", ip: "192.168.30.10"
  config.vm.synced_folder "shared", "/home/vagrant/projects"
  config.vm.hostname = "controller"
  config.vm.provider "virtualbox" do |v|
    v.memory = 1024
    v.cpus = 1
  end
end
```

Básicamente, crearemos una máquina virtual con las siguientes características:

- Utilización de una imagen Ubuntu 16.04
- Utilizará una red privada y tendrá la IP 192.168.30.10
- Existe un directorio donde se encuentra el Vagrantfile llamado 'shared' que se mantiene sincronizado con el directorio de la máquina virtual '/home/vagrant/projects'. Este directorio se crea si no existe.
- Establecemos el nombre de hostname a 'controller' para saber qué máquina es el controlador.
- Establecer características físicas: 1GB de memoria RAM y 1 procesador.

Una vez creada la máquina, nos conectaremos a ella a través de 'vagrant ssh' o a partir del comando ssh. Recordemos que las credenciales de Vagrant son: usuario: vagrant; contraseña: vagrant).

```
ssh vagrant@<ip>
```

```
jose@ubucampos:~/ansible$ ssh vagrant@192.168.30.10
vagrant@192.168.30.10's password:
Permission denied, please try again.
vagrant@192.168.30.10's password:
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-131-generic x86_64)
```

Ya dentro de la máquina virtual, para la instalación de Ansible lo primero que tendremos que hacer será agregar el repositorio de Ansible a nuestro sistema para después descargarlo.

```
apt-add-repository ppa:ansible/ansible
```

```
jose@ubucampos:~$ sudo apt-add-repository ppa:ansible/ansible
[sudo] contraseña para jose:
```

```
apt-get install ansible
```

```
jose@ubucampos:~$ sudo apt-get install ansible
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
```

Listo, ya tenemos nuestra máquina virtual controladora. Ahora pasaremos a crear una máquina virtual que tomará el rol de nodo. Para no complicar la configuración de la máquina, emplearemos el mismo Vagrantfile de antes pero cambiaremos la IP por la 192.168.30.12, el hostname por 'nodo1' y no compartiremos ninguna carpeta con la máquina anfitrión.


```
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "bento/ubuntu-16.04"
  config.vm.network "private_network", ip: "192.168.30.12"
  config.vm.hostname = "nodo1"
  config.vm.provider "virtualbox" do |v|
    v.memory = 1024
    v.cpus = 1
  end
end
```

Esta máquina nodo la vamos a ir configurando desde la máquina controlador. Como hemos mencionado antes, Ansible realiza la conexión por SSH y debemos realizar la configuración para que sea posible realizarla como súper usuario. Así pues, desde la máquina controlador tendremos que establecer una conexión SSH con el nodo.

```
vagrant@controller:~$ ssh vagrant@192.168.30.12
The authenticity of host '192.168.30.12 (192.168.30.12)' can't be established.
ECDSA key fingerprint is SHA256:1Z76nTl7aEpVhcnFIanMDmBiVXrhL9SXkGVxN7LcLD0.
Are you sure you want to continue connecting (yes/no)? yes
```

Una vez en ella, nos identificamos como súper usuario con 'sudo su' , modificamos la contraseña a la que queramos con el comando 'passwd' y, por último, modificamos el fichero '/etc/ssh/sshd_config', modificando la variable de autenticación para permitir la conexión como *root*.

```
# Authentication:
LoginGraceTime 120
PermitRootLogin yes
StrictModes yes
```

Luego, ejecutamos el comando 'service ssh restart' y a partir de este momento, podremos iniciar sesión al nodo desde el controlador como *root*.

```
vagrant@controller:~$ ssh root@192.168.30.12
root@192.168.30.12's password:
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-131-generic x86_64)
```

Ahora bien, si queremos que el acceso a los nodos se realice sin petición de contraseña, tendremos que generar una clave y llevarla anexada acceso a la máquina nodo empleando la contraseña del usuario *root* del nodo.

```
ssh-keygen -t rsa
ssh-copy-id root@<ip_nodo>
```

```
vagrant@controller:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vagrant/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vagrant/.ssh/id_rsa.
```

```
vagrant@controller:~$ ssh-copy-id root@192.168.30.12
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/vagrant/.ssh/id_rsa.pub"
```

Al acabar, el terminal nos pedirá que probemos de realizar la conexión SSH con el nodo para comprobar que se ha efectuado satisfactoriamente la operación y ya en el nodo no tenemos que autenticarnos.

```
vagrant@controller:~$ ssh root@192.168.30.12
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-131-generic x86_64)
```

5.2.2.-Inventarios de Ansible

Existen diferentes maneras de decirle a Ansible qué nodos va a gestionar, siendo la más fácil y común la de añadir al inventario las máquinas que se mantiene en el servidor controlador de Ansible. Este inventario se localiza en '/etc/ansible/hosts'.

```
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
# - Comments begin with the '#' character
# - Blank lines are ignored
# - Groups of hosts are delimited by [header] elements
# - You can enter hostnames or ip addresses
# - A hostname/ip can be a member of multiple groups
```

El fichero destaca por ser un fichero completamente comentado en el que se explican formas y maneras de indicar direcciones IP, rangos, etc. Como hemos dicho, la forma más fácil es utilizar el inventario de Ansible por este fichero ya que fácilmente para controlar diversas máquinas, solo hace falta indicar las direcciones IPs al final del documento.

```
#db-[99:101]-node.example.com
192.168.30.13
192.168.30.14
```

Con tan solo esta fácil edición, ahora Ansible sabe cuáles son los nodos donde va a tener que ejecutar los comandos que ejecutaremos.

Pero está claro que Ansible-Playbook no solo vamos a poder lanzar tareas para todos los hosts, podemos especificar qué tareas ejecutar en qué hosts, comandos, etc.; orquestando nosotros como usuarios nuestra colección de nodos.



Dentro de un inventario, como el de '/etc/ansible/hosts', vamos a tener diferentes grupos y hosts. Podemos agrupar los hosts en grupos mediante la utilización de una palabra clave encerrada entre corchetes, que será utilizado por un Play para lanzar tareas para un grupo específico. Hay que tener en cuenta que no todos los hosts escuchan SSH por el puerto 22, pero podemos definir el puerto SSH del host al final de cada uno (192.168.30.14:4422).

Por ejemplo, podemos tener un inventario tal como el que sigue y un Playbook. Podemos ver que en el Playbook se define que se debe tener instalado Apache en todos los nodos de nuestro inventario pero solo en aquellos que se encuentran agrupados bajo el grupo 'databases' se va llevar a cabo la instalación de MariaDB

```
[databases]
192.168.30.13

[web]
192.168.30.14
192.168.30.15
```

```
- hosts: all
  remote_user: root
  tasks:
    - name: Ensure Apache is at the latest version
      apt: name=apache2 state=latest

- hosts: databases
  tasks:
    - name: Ensure MariaDB is at the latest version
      apt: name=mariadb state=latest
```

También es cierto que podemos aprovechar ficheros ansible-playbook.yml ya creados y disponibles para anexarlos en nuevos ficheros ansible-playbook.yml, pero no de la forma de copiar el código en el nuevo fichero, sino que sea el propio Ansible el que acuda al fichero en cuestión y traiga consigo esa configuración.

Para realizar esta operación, partiremos de un fichero ansible-playbook.yml al que llamamos apache-playbook.yml. Y solicitaremos a Ansible que incluya esta configuración en el nuevo fichero ansible-playbook.yml.

```
--
- name: Ensure Apache is at the latest version
  apt: name=apache2 state=latest
- name: Ensure Apache is running
  service: name=apache2 state=started enabled=yes
...
```

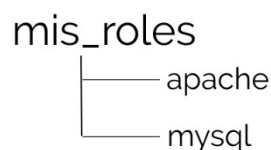
```
- hosts: all
  remote_user: root
  tasks:
    - include: apache-playbook.yml
```

```
vagrant@controller:~/projects$ ansible-playbook ansible-playbook.yml

PLAY *****

TASK [setup] *****
ok: [192.168.30.14]
ok: [192.168.30.13]
```

Cuando nuestra aplicación crece y hacemos 'include' dentro de otro, etc.; puede volverse caótico y tener una partitura ilegible. Para ello existen los roles, paquetes de configuración bien organizados que pueden utilizarse en cualquier entorno y de forma estructura. Los roles funcionan siguiendo una estructura de directorios, por el cual cada uno de los directorios contendrá todas las tareas pertinentes para realizar la consecución con éxito del objetivo.



Dentro de cada uno de los roles deberíamos encontrar:

- **Directorio 'files':** almacenar todos los ficheros de configuración que queremos mover de un controlador a un nodo.
- **Directorio 'templates':** plantillas que se utilizan para configurar roles.
- **Directorio 'tasks':** reúne todas las tareas que realiza un rol.
- **Directorio 'handlers':** manejadores.
- **Directorio 'vars':** variables.
- **Directorio 'defaults':** contiene las variables por defecto de un rol.
- **Directorio 'meta':** información sobre el rol.

Al final, un fichero ansible-playbook.yml utilizando el funcionamiento de roles, quedaría así de sencillo, aunque por detrás es por donde ocurre la magia.

```
- hosts: databases
  mis_roles:
    - apache
```

En la página web <https://galaxy.ansible.com>, tenemos una gran cantidad de roles que podemos descargar ya preparados para su utilización fácil y próspera, a los cuales podemos acceder a los repositorios de GitHub de manera que ya se encuentran ordenados y solo tendremos que descargarlos y situarlos en nuestro ecosistema.

The screenshot shows the GitHub repository for 'geerlingguy / ansible-role-php-mysql'. At the top, it displays the repository name and statistics: 6 Watchers, 20 Stars, and 23 Forks. Below this, there are tabs for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Security, and Insights. The repository is described as 'Ansible Role - PHP MySQL support' with a link to the Galaxy page. It includes tags for 'ansible', 'role', 'php', 'mysql', 'database', 'php-mysql', 'extension', and 'pdo'. Statistics show 55 commits, 1 branch, 13 releases, and 2 contributors under the MIT license. A 'Clone or download' button is visible. A list of files and their commit history is shown, including 'defaults', 'meta', 'molecule/default', 'tasks', 'vars', '.gitignore', '.travis.yml', 'LICENSE', and 'README.md'.

5.2.3.-Módulos y comandos ad-hoc

La forma más sencilla de utilizar Ansible es ejecutar comandos 'ad-hoc' los cuales utilizan módulos siendo su formato como el siguiente:

```
ansible <grupo_hosts> -i <fichero_de_inventario> -m <módulo> [-a <argumento1>]. ...]
```

Por ejemplo, la ejecución del comando *ping* a todos los módulos lo podemos realizar a través del siguiente ejemplo:

```
vagrant@controller:~$ ansible all -i /etc/ansible/hosts -m ping -u root
192.168.30.13 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
192.168.30.14 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

Con este ejemplo, vemos la funcionalidad paralela de Ansible, ejecutando de forma sincronizada en las dos máquinas virtuales haciendo mucho más eficiente el trabajo del administrador. Podemos también lanzar comandos sobre los nodos utilizando el parámetro '-a'. Si el fichero de inventario es '/etc/ansible/hosts' no hará falta definirlo en el lanzamiento del comando.

```
vagrant@controller:~$ ansible all -a "hostname" -u root
192.168.30.14 | SUCCESS | rc=0 >>
nododos
192.168.30.13 | SUCCESS | rc=0 >>
nodouno
```

Se puede deducir por los ejemplo que los comandos 'ad-hoc' resultan muy eficientes para lanzar tan solo una vez un comando y que se ejecute N veces en cada nodo y al mismo tiempo.

Para algunos comandos será necesario acudir a módulos y a su estructura para llevarlos a cabo, como bien podría ser a la hora de instalar, que quedaría como en la imagen que sigue. Ansible buscará en cada máquina que se encuentre el paquete 'ntp' instalado y en caso de que no esté en ese estado, lo descargará. Si quisiéramos desinstalar un paquete, deberemos cambiar el estado a 'absent'.

```
vagrant@controller:~$ ansible all -m apt -a "name=ntp state=installed" -u root
```

Con el paquete ya instalado, solo faltará iniciarlo y claro está lo vamos a hacer con Ansible:

```
vagrant@controller:~$ ansible all -m service -a "name=ntp state=started" -u root
192.168.30.14 | SUCCESS => {
  "changed": false,
  "name": "ntp",
  "state": "started"
}
```

Comprobamos que el servicio está en marcha en cualquiera de los nodos, o bien desde el propio controlador Ansible.

```
root@nodouno:~# sudo /etc/init.d/ntp status
● ntp.service - LSB: Start NTP daemon
   Loaded: loaded (/etc/init.d/ntp; bad; vendor preset: enabled)
   Active: active (running) since Tue 2019-06-11 20:35:37 UTC; 6min ago
     Docs: man:systemd-sysv-generator(8)
    CGroup: /system.slice/ntp.service
            └─13700 /usr/sbin/ntpd -p /var/run/ntpd.pid -g -u 112:117
```

Veremos a continuación unos ejemplos de los módulos y su utilización:

Crear un grupo

- `ansible all -m group -a "name=admin state=present" -u root`

```
vagrant@controller:~$ ansible all -m group -a "name=admin state=present" -u root
192.168.30.13 | SUCCESS => {
```

```
root@nododos:/home/vagrant# cat /etc/group | grep admin
lpadmin:x:115:vagrant
admin:x:1001:
```

Crear un usuario

- `ansible all -m user -a "name=jose group=admin createhome=yes" -u root`

```
vagrant@controller:~$ ansible all -m user -a "name=jose group=admin createhome=yes" -u root
192.168.30.14 | SUCCESS => {
```

```
root@nododos:/home/vagrant# cat /etc/passwd | grep jose
jose:x:1001:1001::/home/jose:
```

Copiar un fichero

- `ansible all -m copy -a "src=fichero.txt dest="/tmp/fichero.txt" -u root`

```
vagrant@controller:~$ ansible all -m copy -a "src=fichero.txt dest=/tmp/fichero.txt" -u root
192.168.30.13 | SUCCESS => {
```

```
root@nododos:/home/vagrant# ls /tmp
fichero.txt
```

Obtener información de un fichero

- `ansible all -m stat -a "path=/etc/hosts" -u root`

```
vagrant@controller:~$ ansible all -m stat -a "path=/etc/hosts" -u root
192.168.30.14 | SUCCESS => {
  "changed": false,
  "stat": {
    "atime": 1560281186.1530378,
```


Descargar un fichero

- `ansible all -u root -m fetch -a "src=/tmp/fichero1.txt dest=/tmp"`

```
vagrant@controller:~$ ansible all -u root -m fetch -a "src=/tmp/fichero1.txt dest=/tmp"
192.168.30.14 | SUCCESS => {
  "changed": true,
```

Crear directorio o ficheros con permisos concretos

- `ansible all -m file -a "dest=/tmp/test mode=664 state=directory" -u root`

```
vagrant@controller:~$ ansible all -m file -a "dest=/tmp/test mode=664 state=directory" -u root
192.168.30.13 | SUCCESS => {
```

```
root@nododos:/home/vagrant# ls -l /tmp
total 8
-rw-r--r-- 1 root root  26 Jun 11 21:05 fichero.txt
-rw-r--r-- 1 root root   0 Jun 11 21:11 fichero1.txt
drw-rw-r-- 2 root root 4096 Jun 11 21:15 test
```

5.3.3.-Ansible Playbooks

Los Playbooks nos proporcionan una manera totalmente diferente de utilizar Ansible. Mientras que podemos ejecutar comandos ad-hoc con Ansible, los Playbooks nos permiten tener un control de versiones como Git. Pueden ejecutar tareas tal y como nosotros queramos haciendo uso de los inventarios, tags, roles, etc.

Un Playbook se compone de varios Plays. El objetivo de un Play es mapear un grupo de hosts con unos determinados roles, los cuales son representados por tareas. Tendremos la posibilidad de orquestar muchas máquinas al mismo tiempo con los Plays de Ansible gracias a la agrupación de los nodos en los inventarios de Ansible.

Los Playbooks están escritos en YAML y con el siguiente estaríamos ejecutando lo siguiente:

1. Para todos los hosts, instalar Apache y llevarlo a la última versión.
2. Asegurarse de que Apache se encuentra corriendo y habilitado.

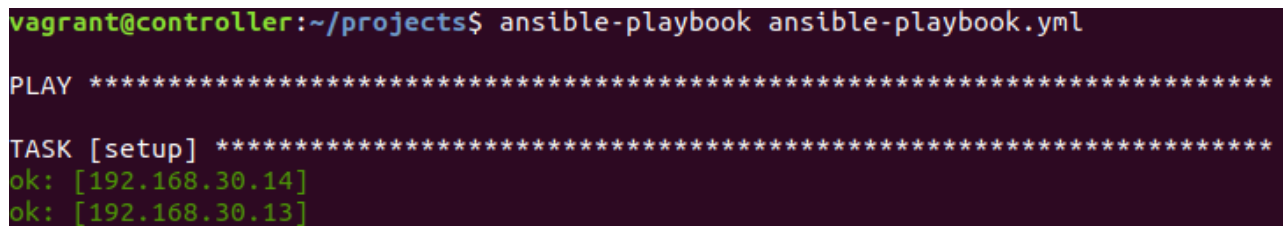
```
- hosts: all
  remote_user: root
  tasks:
    - name: Ensure Apache is at the latest version
      apt: name=apache2 state=latest
    - name: Ensure Apache is running
      service: name=apache2 state=started enabled=yes
```


Cada Play contiene una lista de tareas las cuales se ejecutan en orden y solo una al mismo tiempo en cada máquina, aunque de forma paralela en cada máquina. Si falla una tarea, se puede volver a ejecutar ya que el resultado de lanzar el mismo comando tiene como resultado la misma salida.

Todas las tareas tienen un 'name', el cual se incluye en la salida del comando al ejecutar el Playbook. La accesibilidad de YAML para ser un lenguaje 'human readable' hace del fichero `playbook.yml` de fácil lectura y comprensión para su ejecución.

Para ejecutar un fichero `ansible-playbook.yml` tendremos que ejecutar el comando siguiente desde el controlador.

```
ansible-playbook ansible-playbook.yml
```



```
vagrant@controller:~/projects$ ansible-playbook ansible-playbook.yml  
PLAY *****  
TASK [setup] *****  
ok: [192.168.30.14]  
ok: [192.168.30.13]
```

Capítulo 6. Orquestación de contenedores

Un sistema de orquestación de contenedores trata el hardware dispar de la infraestructura como una colección y lo representa para la aplicación como un único recurso. Programa los contenedores basándose en las restricciones de los usuarios y utiliza la infraestructura de la manera más eficiente posible, escalando los contenedores dinámicamente y manteniendo los servicios en alta disponibilidad.

6.1.-Fleet

Fleet utiliza el modelo maestro-esclavo donde el motor Fleet desempeña el papel de maestro y el agente Fleet el de esclavo. El motor responsable de programar las unidades Fleet y el agente de ejecutarlas y divulgar su estado al motor. Las unidades Fleet también constan de metadatos para controlar dónde se ejecuta la unidad con respecto a la propiedad del nodo, así como la base de otros servicios que se ejecutan en ese nodo en particular. Si un nodo muere, se elige un nuevo motor y las unidades de programa en ese nodo se reprograman en un nuevo nodo. Fleet se utiliza principalmente para la orquestación de servicios críticos del sistema.

6.2.-Otras herramientas para la orquestación de contenedores

Otras herramientas para la orquestación muy populares en la actualidad son Kubernetes, Docker Swarm y Apache Mesos.

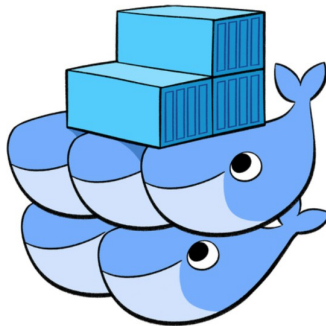
- **Kubernetes:** plataforma de código abierto para la orquestación de contenedores iniciada por Google. La unidad más pequeña en Kubernetes es un *pod*. Los *pods* son un conjunto de contenedores que se encuentran juntos en un único nodo. Cada *pod* tiene una dirección IP y todos los contenedores de ese *pod* la comparten.



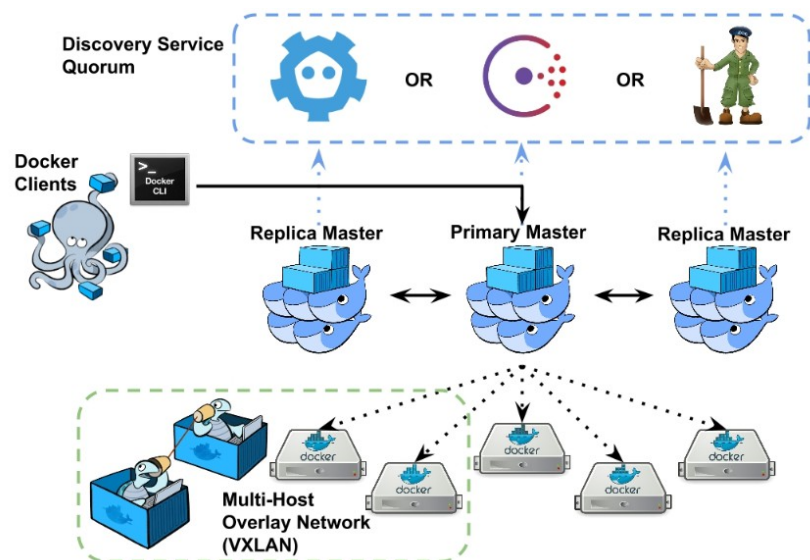
Se compone de diferentes servicios. Kubelet es el responsable del estado de ejecución de cada nodo. Se ocupa de iniciar, detener y mantener los contenedores de aplicaciones. Kube-proxy se ocupa de la redirección de servicios y el balanceo de carga del tráfico en los *pods*. El servidor API sirve a la API estándar utilizando

JSON sobre HTTP, proporcionando tanto la interfaz interna como externa a Kubernetes. Así, el servidor API procesa y valida las peticiones REST y actualiza el estado de los objetos API en etcd, permitiendo a los clientes configurar cargas de trabajo y contenedores. El planificador es el componente conectable que selecciona en qué nodo debería ejecutarse un *pod* no programado basado en la disponibilidad de recursos y las cargas de trabajo existente.

- **Docker Swarm:** es la solución de orquestación nativa de Docker.



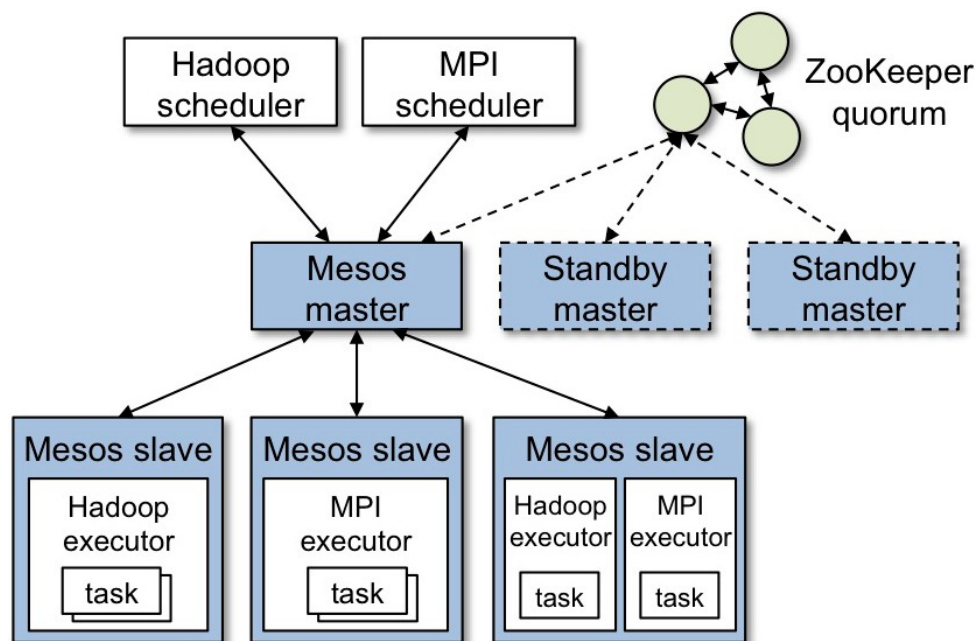
Con su uso se administra el clúster como entidad en lugar de administrar nodos individuales. Esta herramienta tiene un planificador integrado que decide la ubicación de los contenedores en el clúster y utiliza restricciones y afinidades específicas del usuario para decidir esa ubicación. En su arquitectura existe un maestro que se encarga de la planificación de contenedores Docker basado en el algoritmo planificador, restricciones y afinidades. Para proporcionar alta disponibilidad se pueden ejecutar múltiples maestros en paralelo, distribuyendo las cargas de trabajo uniformemente. Los agentes se ejecutan en cada nodo y se comunican con el maestro a partir del descubrimiento, necesario porque se ejecutan en diferentes nodos no iniciados por él.



- **Apache Mesos:** desarrollado por la Universidad de California Berkeley, combina el sistema operativo con el administrador de clústeres.



Mesos consiste en un demonio maestro que gestiona los demonios agentes que se ejecutan en cada nodo del clúster y los marcos de Mesos que ejecutan tareas en estos agentes. El maestro permite un uso compartido de recursos a través de marcos. Cada oferta de recursos contiene una lista de agentes. El maestro decide cuántos recursos ofrecer a cada marco de acuerdo con una determinada política de organización. Para soportar un conjunto diverso de políticas, el maestro emplea una arquitectura modular que facilita la adición de nuevos módulos de asignación a través de un mecanismo de complemento. Un *framework* que se ejecuta en la parte superior de Mesos consta de dos componentes: un programador que se registra con el maestro y un proceso ejecutor que se ejecuta en los nodos del agente para ejecutar las tareas del marco. Mientras que el maestro determina cuántos recursos se ofrecen a cada marco, los planificadores de los marcos seleccionan cuál de los recursos ofrecidos utilizar.



Capítulo 7.-Desplegamiento

Para mostrar el funcionamiento de las diversas plataformas que se han presentado durante la documentación, partiremos de tres ejercicios prácticos:

1. Automatizar la creación de una máquina virtual con Ubuntu 16.04 que tenga instalado Apache2.
2. Automatizar la creación de un contenedor con Apache que muestre una página web descargada de GitHub.
3. Orquestación de contenedores Docker mediante Docker Compose para una aplicación Web que utiliza PHP, phpmyadmin y MySQL.

7.1.-Vagrant

Partiremos de un fichero Vagrantfile en el cual, de momento, solo tendremos que definir la *box* que vamos a utilizar. Para ello, a modo de recordatorio, acudimos al repositorio de Vagrant y hacemos una búsqueda de una *box* que nos sirva como base. Así, encontramos la imagen 'ubuntu/xenial64' que se corresponde a un Ubuntu 16.04, y ejecutamos el comando siguiente para generar ya un fichero Vagrantfile con la *box* ya definida.

```
vagrant init ubuntu/xenial64
```



ubuntu/xenial64 20190613.1.0

Official Ubuntu 16.04 LTS (Xenial Xerus) Daily Build

Nuestra máquina virtual va a estar destinada a ofrecer una página web y para ello vamos a tener que instalar Apache. Para ello, accedemos a nuestro fichero Vagrantfile y en el apartado de aprovisionamiento, vamos a indicar los comandos necesarios para realizar tal instalación. Hay que destacar que previamente tendremos que actualizar la máquina virtual para contar con el comando de instalación.

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

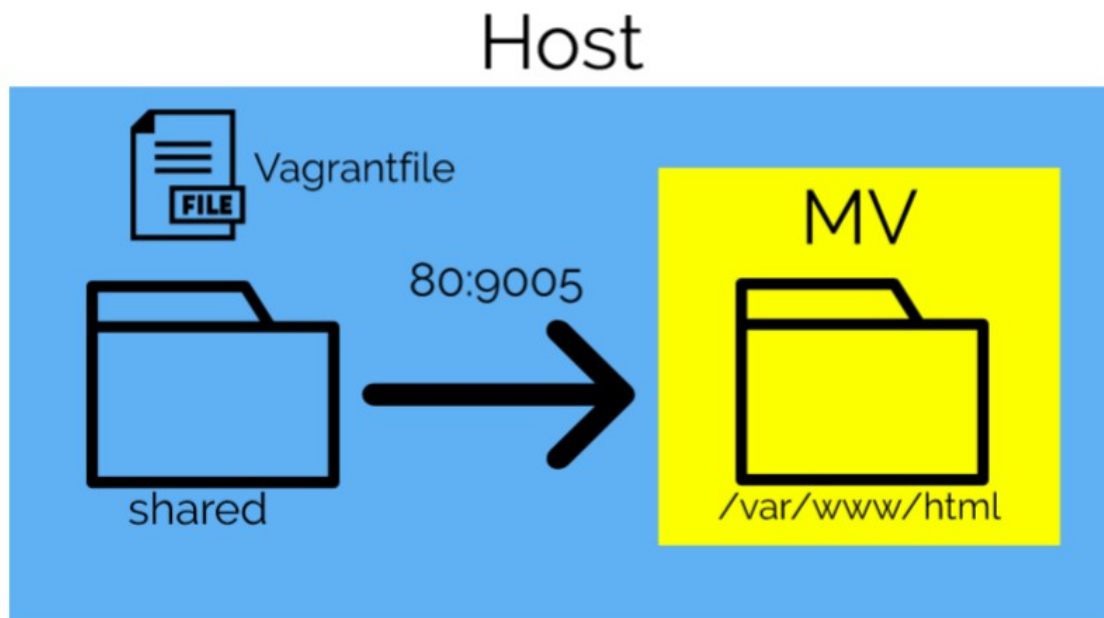
Para esta máquina vamos a utilizar dos directrices definidas en el Vagrantfile que van a ser el de sincronización de carpeta y el redireccionamiento de puertos.

```
jose@ubucampos:~/tf/vagrant$ ls -l
total 8
drwxr-xr-x 9 jose jose 4096 jun 13 21:13 shared
-rw-r--r-- 1 jose jose 3016 jun 13 22:11 Vagrantfile
jose@ubucampos:~/tf/vagrant$ ls shared/
css          index.html  package.json  startbootstrap-freelancer-gh-pages
gh-pages.zip js          package-lock.json  vendor
gulpfile.js  LICENSE    README.md
img          mail       scss
```

Para la primera directriz, definimos un nuevo directorio que se va a encontrar en la misma localización que el Vagrantfile. Enfrentaremos este directorio con el directorio de la máquina virtual 'var/www/html' que es por defecto el DocumentRoot de Apache, por lo que al acceder al servicio se mostrará el contenido de este directorio.

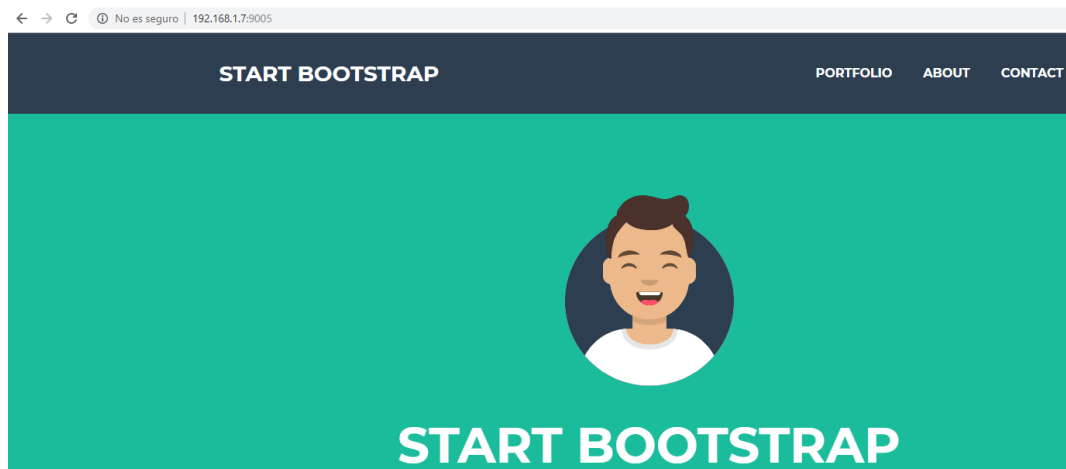
```
# Share an additional folder to the guest VM. The first argument is  
# the path on the host to the actual folder. The second argument is  
# the path on the guest to mount the folder. And the optional third  
# argument is a set of non-required options.  
config.vm.synced_folder "shared", "/var/www/html"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

La segunda directriz definiremos el puerto de redireccionamiento de la máquina anfitrión a la máquina virtual. De tal forma, que al acceder a la máquina virtual por su IP e indicando el puerto, accederemos directamente a la máquina virtual.



```
# 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: 9005
```

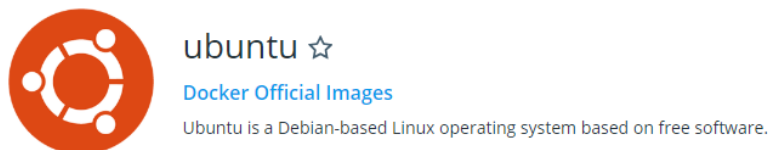
Solo quedará levantar la máquina virtual con el comando 'vagrant up' para tener la máquina lista.



El fichero Vagrantfile de esta máquina se podrá encontrar completo en el repositorio de GitHub.

7.2.-Docker

En el caso de Docker, tendremos que elaborar un fichero Dockerfile que sirva para generar una imagen a partir de la cual construir posteriormente nuestro contenedor.



En este fichero Dockerfile vamos a indicar la automatización de la descarga de una página web disponible de GitHub, pero para ello tendremos que instalar todos los requisitos para realizar tal operación, como la instalación del paquete 'unzip', para descomprimir el fichero objetivo de GitHub, así como la herramienta 'wget' para poder llevar a cabo la instalación. Con la página web disponible en la imagen del contenedor, moveremos los ficheros obtenidos al directorio '/var/www/html' que es el DocumentRoot de Apache.

```
FROM ubuntu:16.04
RUN apt-get update
RUN apt-get install -y apache2
RUN apt-get install -y unzip
RUN apt-get install -y wget
RUN wget https://github.com/BlackrockDigital/startbootstrap-freelancer/archive/gh-pages.zip
RUN unzip gh-pages.zip
RUN cp -a startbootstrap-freelancer-gh-pages/* /var/www/html
EXPOSE 9010
CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```


Una vez elaborado el Dockerfile, ejecutamos el comando siguiente para elaborar la imagen. El parámetro '-t' permite ponerle un nombre a la imagen, recomendando utilizar un nombre que guarde relación con la imagen para aliviar posibles futuros problemas. El comando elaborará una imagen llamada 'docker_bootstrap' a partir del fichero Dockerfile que se encuentra en el mismo directorio desde el cual ejecutamos el comando, de ahí el punto (.).

```
docker build -t docker_bootstrap .
```

Podemos consultar las imágenes disponibles y encontrar entre ellas la que acabamos de elaborar:

```
jose@ubucampos:~/tf/docker$ docker images
```

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|------------------|--------|--------------|---------------|-------|
| docker_bootstrap | latest | 9c1313981df7 | 5 minutes ago | 262MB |

Ya solo quedará elaborar el contenedor a partir del siguiente comando, en el que definimos cuál va a ser el puerto a través del cual la máquina anfitrión redirigirá hacia el contenedor mediante el parámetro '-p' y utilizaremos también el parámetro '-d' para que se ejecute en segundo plano y deje el terminal libre para ser utilizado.

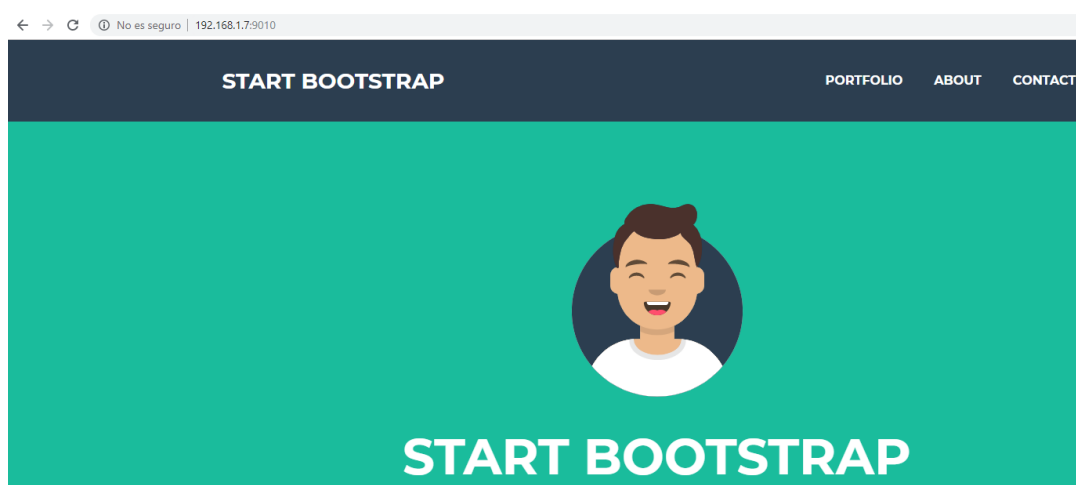
```
docker run -d -p 9010:80 docker_bootstrap
```

Para verificar que se encuentra en funcionamiento, podemos lanzar el comando 'docker ps' y mostrar todos los contenedores que se encuentran operativos.

```
jose@ubucampos:~/tf/docker$ docker ps
```

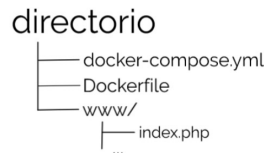
| CONTAINER ID | IMAGE | COMMAND NAMES | CREATED | STATUS |
|--------------|------------------|---|---------------|--------------|
| 0e68e7cd04c7 | docker_bootstrap | "/usr/sbin/apache2ct..." stoic_lederberg | 8 minutes ago | Up 8 minutes |

Y, efectivamente, accediendo a la máquina por el puerto 9010 accedemos al contenedor.

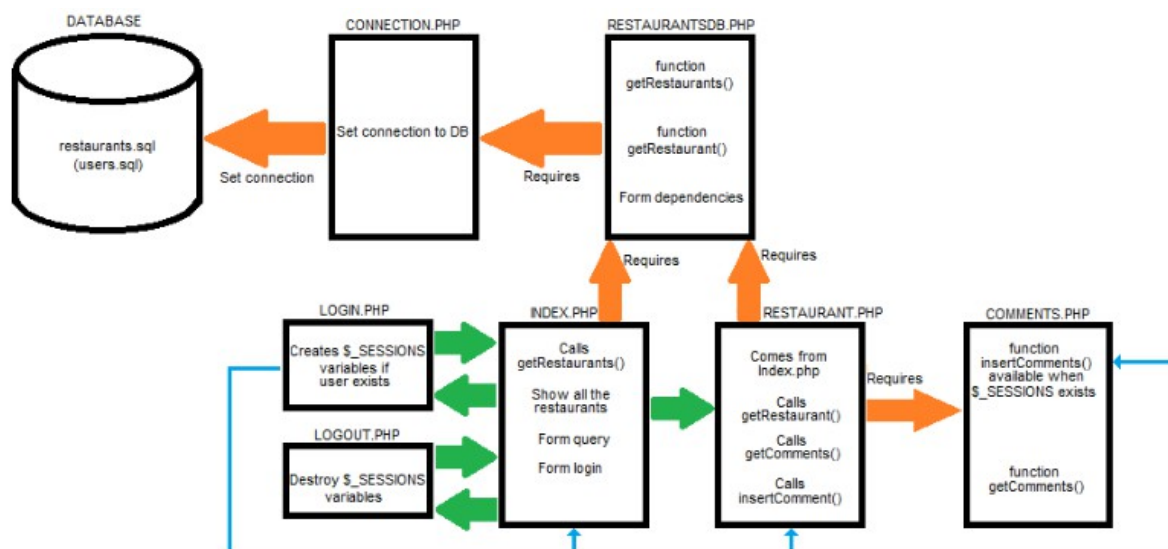


7.3.-Docker Compose

Para Docker Compose partiremos de un directorio en el cual tendremos los siguientes ficheros y directorios:



La opción de utilizar un fichero Dockerfile radica en que tendremos que utilizar una imagen personalizada en la que se encuentren instaladas las extensiones para *mysqli*, y así también aprovechamos para construir un contenedor utilizando la cláusula 'build'. En el directorio 'www/' encontramos todos los ficheros para la aplicación PHP: connection.php, login.php, index.php, etc.; que se relacionan siguiendo el siguiente esquema:



Así pues, estamos hablando de que tendremos 3 contenedores:

- Contenedor web con PHP y Apache.
- Contenedor MySQL.
- Contenedor phpmyadmin.

El fichero Dockerfile tiene el siguiente contenido, simplemente para instalar una imagen que viene con PHP y Apache y habilitar la extensión 'mysqli'

```
FROM php:7.3-apache

RUN docker-php-ext-install mysqli
RUN docker-php-ext-enable mysqli
```

Continuamos y vamos a ver en detalle el fichero docker-compose.yml:

```
version: '3'

services:
  web:
    build: .
    ports:
      - "9015:80"
    volumes:
      - ./www:/var/www/html

  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: secret

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    environment:
      MYSQL_ROOT_PASSWORD: secret
    ports:
      - "8085:80"
```

1. Definimos la versión y pasamos a exponer los servicios.
2. Contenedor web: construido a partir del Dockerfile situado en el propio directorio donde se encuentra docker-compose.yml.
3. Redirige por el puerto 9015 de la máquina anfitrión al puerto 80 del contenedor.
4. El volumen que se sincroniza es './www' en el host contra '/var/www/html' en el contenedor.
5. Contenedor db: utiliza la imagen 'mysql:5.7' del repositorio de Docker y se crea la variable de entorno 'secret' como contraseña del usuario 'root'.
6. Contenedor phpmyadmin: utiliza la imagen 'phpmyadmin/phpmyadmin' y utiliza la misma variable de entorno, escuchando por el puerto 80 redirigido desde la máquina anfitrión a través del 8085.

Hecho esto, podremos acceder a la dirección del contenedor de phpmyadmin con el usuario root y la contraseña que hemos definido en el docker-compose.yml e importar la base de datos que tenemos creada.



Lo último que nos queda es modificar las variables del fichero 'connection.php' para establecer los valores de las variables que se utilizan para establecer la conexión con la base de datos.

```
<?php
define ("DB_HOST","ita_db_1");
define ("DB_USER","root");
define ("DB_PASS","secret");
define ("DB_NAME","restaurants");
define ("DB_PORT","3306");

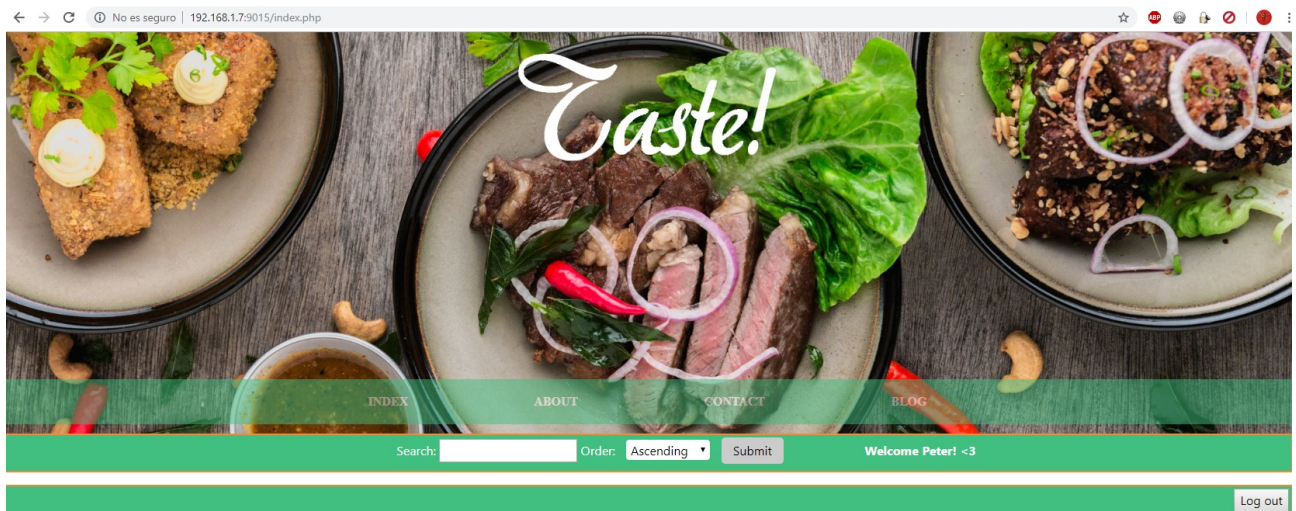
$connection = mysqli_connect(DB_HOST,DB_USER,DB_PASS,DB_NAME,DB_PORT);

if(!$connection) {
    exit ("Can not access to the database. ".mysqli_error($connection));
}

#echo "Yes"; Connection established
?>
```

- **DB_HOST:** nombre del contenedor Docker. También se podría utilizar su ID. El nombre se crea automáticamente a partir del nombre del directorio donde se encuentra el fichero docker-compose.yml seguido del nombre del contenedor definido en el docker-compose.yml.
- **DB_USER:** usuario para establecer la conexión.
- **DB_PASS:** contraseña definida como variable de entorno.
- **DB_NAME:** nombre que hemos puesto a la base de datos a la hora de importar la base de datos a phpmyadmin.
- **DB_PORT:** parámetro opcional, por defecto el 3306.

Con todo ya parametrizado correctamente, podremos acceder a la aplicación PHP perfectamente y disfrutar de ella.



Acceso al repositorio

https://github.com/jcamposp/TFinal_Virtualizacion

Bibliografía

- Anderson, Charles. "Docker [software engineering]." *IEEE Software* 32.3 (2015): 102-c3.
- Ares Martín, José Luis. "Virtualización y Cloud Computing en la PYME." (2012).
- Geerling, Jeff. "Ansible for DevOps." (2014).
- Pérez Ramírez, Jesús. "Citisocial 2.0: Redes sociales en la ciudad inteligente." (2017). Santana Martel, Carolina. "Despliegue de una aplicación Ruby on Rails utilizando las tecnologías de virtualización Docker y Coreos en la nube pública de Amazon Web Services." (2017)
- Pons Martínez, Aglaya. "Uso de contenedores (Docker) en aplicación web ASP. NET CORE 2." (2018).