UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA EN TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESPLIEGUE AUTOMATIZADO DE ESCENARIOS DE RED VIRTUALIZADOS

SAMUEL GARCÍA SÁNCHEZ

2022

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA EN TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESPLIEGUE AUTOMATIZADO DE ESCENARIOS DE RED VIRTUALIZADOS

Autor
SAMUEL GARCÍA SÁNCHEZ

Tutor
MARIO SANZ RODRIGO

2022

Resumen

Ejemplo resumen **Nombre:** Tecnilógica Ecosistemas SAU **Accenture** Rebus sic stantibus [1]

[1] os la estoy metiendo doblada

Abstract

Ejemplo de código en bash:

```
#!/bin/bash
#Enumeracion de los puertos abiertos de una maquina
  for port in $(seq 1 65535); do
    timeout 1 bash -c "echo > /dev/tcp/10.10.10.52/$port" > /dev/null
 2>&1 && echo "$port/tcp" &
done; wait
```

Código 1: Bash example

Referencia a la figura 2.1 de la página 3

Índice general

Índice de figuras

Capítulo 1

Introducción

El código empleado en este trabajo está disponible en el siguiente repositorio de GitHub:

Dirección URL

https://github.com/samugs13/DAERV

Capítulo 2

Estado del arte

2.1. Tecnologías de virtualización

Se podría decir que la virtualización es ya uno de los pilares fundamentales del mundo IT debido a las grandes ventajas que proporciona. Previo al desarrollo de las tecnologías y tipos de virtualización disponibles, es conveniente explicar en qué consiste la virtualización, que no es más que una representación mediante software de un entorno físico o recurso tecnológico, como pueden ser aplicaciones, servidores o almacenamiento. [1]

Gracias a esta tecnología, es posible contar con varios ordenadores virtuales en el mismo hardware, donde cada uno de ellos puede interactuar de forma independiente y ejecutar sistemas operativos o aplicaciones diferentes mientras comparten los recursos de una sola máquina host. Al crear varios recursos a partir de un único equipo o servidor, la virtualización mejora la escalabilidad y las cargas de trabajo, al tiempo que permite usar menos servidores y reducir el consumo de energía, los costos de infraestructura y el mantenimiento.

En función del sistema a simular, podemos encontrar diferentes categorías [2], un ejemplo es la virtualización de red, que consiste en crear redes virtuales sobre redes físicas o reproducir completamente redes físicas en software. Otro ejemplo sería la virtualización de almacenamiento, que combina varios dispositivos de almacenamiento en red, con la apariencia de una única unidad o dispositivo de almacenamiento, accesible por varios usuarios. Podríamos enumerar más tipos de virtualización, pero en lo que a este trabajo respecta vamos a centrarnos en la virtualización de software, que separa las aplicaciones del hardware y el sistema operativo, y en la que distinguimos dos subtipos: virtualización mediante hipervisor y virtualización en contenedores.

2.1.1. Virtualización mediante hipervisor

Una máquina virtual es un software que ejecuta programas o procesos como si fuera la máquina física. Es decir, se abstrae el hardware y se representa con una capa de software que proporciona una interfaz igual que el hardware, de forma que sobre ella podemos instalar uno o varios sistema operativos invitados o *guests* distintos. Esta capa de software

también se encarga de repartir y aislar los recursos del host entre las VM[1], de manera que el host queda protegido si falla una VM, y las VM estén protegidas entre ellas. Pues bien, cuando hablamos de esta capa de software estamos hablando de lo que se conoce como hipervisor.

Como ya se ha mencionado, un hipervisor es una capa intermedia de software que permite al ordenador anfitrión prestar soporte a varias máquinas virtuales mediante el uso compartido de sus recursos. Cuando se ejecuta una instrucción en el SO[2] invitado, el hipervisor la coje y la ejecuta en el SO anfitrión. En este proceso, el SO no diferencia entre ejecutar procesos en la máquina virtual o en la física, lo que representa plenamente el concepto de virtualización.

Dentro de los hipervisores [3], podemos distinguir dos tipos. El primero es el Tipo 1, conocido también como hipervisor nativo o *bare-metal*. Este hipervisor se ejecuta directamente sobre el hardware en lugar de un SO clásico. Todos los hipervisores necesitan algunos elementos del sistema operativo (por ejemplo, el administrador de memoria, el programador de procesos, la pila de entrada o salida [E/S], los controladores de dispositivos, entre otros) para ejecutar las máquinas virtuales. Por tanto, este hipervisor es equivalente a un SO con un poco de información adicional que le permite gestionar los SO invitados. Es muy común encontrarlos en centros de datos, por la eficiencia que supone el ahorrar una capa de software.

Los hipervisores de Tipo 2 se ejecutan sobre el SO anfitrión como una capa de software o aplicación. Están orientados a usuarios individuales que buscan ejecutar varios SO en el mismo ordenador. La ejecución de una VM sobre un hipervisor de este tipo es más lenta que en un hipervisor de Tipo 1.

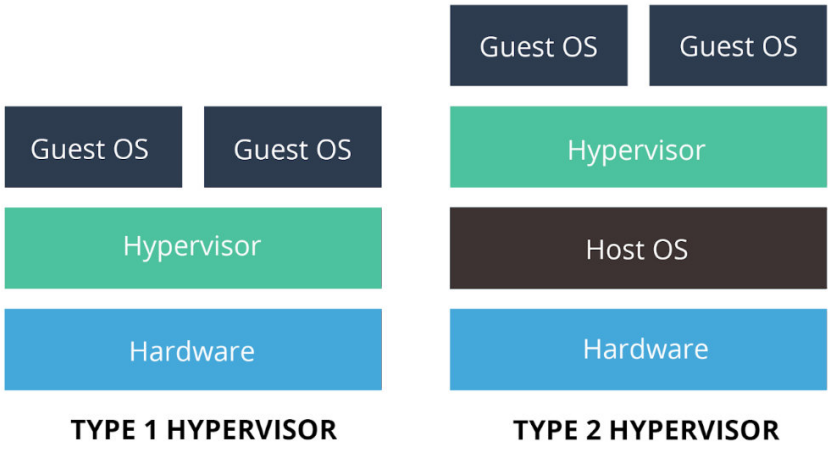

Figura 2.1: Tipos de hipervisor

A continuación se presentan algunas tecnologías que emplean este tipo de virtualización.

[1]Virtual Machine
[2]Sistema Operativo

KVM

KVM (Kernel Virtual Machine) [4] es una tecnología de virtualización open source que convierte el kernel de Linux en un hipervisor de Tipo 1 que se puede usar para la virtualización. Las KVM tienen todos los elementos necesarios de un SO porque forman parte del kernel de Linux. Cada máquina virtual se implementa como un proceso habitual de Linux. Al ser un hipervisor de Tipo 1 ofrece un mejor rendimiento.

Figura 2.2: Logo de KVM

La configuración de la máquina virtual creada se almacena internamente en un fichero XML, el cual es posible editar manualmente a posteriori si se quiere hacer algún cambio. KVM nos permite disfrutar de las ventajas del software open source: no habrá restricciones en cuanto a integración, como sí puede haberlas si se usa un software propietario como VMware; y es independiente de proveedores. Es posible instalar una GUI[3] como virt-manager, que se apoya en la biblioteca libvirt (API de virtualización estándar de Linux), para facilitar su uso.

VirtualBox

VirtualBox [5] es desarrollado por Oracle, aunque es gratuito y open source al igual que KVM. Es un hipervisor de Tipo 2, por lo que ofrece un rendimiento inferior comparado con un Tipo 1. VirtualBox permite crear y cargar máquinas virtuales de una forma muy sencilla, lo que hace que sea la alternativa elegida por muchos usuarios. Su asistente ofrece algunos valores sugeridos para tipos específicos de máquinas virtuales durante la creación de estas, pero su gestión final se produce en una configuración posterior.

Figura 2.3: Logo de VirtualBox

Una ventaja de VirtualBox son las instantáneas, que permiten tomar una imagen de la máquina virtual en un momento dado. La imagen conserva la máquina virtual, lo que permite volver a ese momento específico.

[3]Graphical User Interface

VirtualBox ofrece un soporte muy completo. Tiene versiones para Windows, Linux, Macintosh y Solaris, y puede ejecutar un amplísimo número de sistemas operativos invitados, incluidos Windows, macOS, Linux, DOS, Solaris u OpenBSD.

VMware

VMware [6] es un software comercial, lo que significa que si queremos aprovechar al máximo todas sus herramientas y configuraciones, debemos pagar por la licencia de uso. Aquí vamos a hablar de VMware Workstation Player, que es el producto gratuito de VMware para virtualización de máquinas, orientado para uso personal, doméstico y sistema educativo.

Figura 2.4: Logo de VMware

En comparación con VirtualBox, VMware Workstation Player es una experiencia más fluida y ágil, y ofrece mejor soporte y estabilidad para una amplia gama de hardware. Está disponible para Windows y Linux y también admite todo tipo de sistemas invitados. [7]

Además, VMware Workstation Player sí permite personalizar toda la configuración durante el proceso de creación de la máquina virtual. La diferencia no es mucha, pero significa que la máquina virtual está lista para ejecutarse después de finalizar el asistente, en lugar de tener que realizar más configuraciones una vez que se completa.

Por el contrario, no admite instantáneas o puntos de control. Puede suspender temporalmente el sistema operativo invitado para reanudar desde un punto específico, pero no es tan completo como la creación de un historial de imágenes para la máquina virtual.

2.1.2. Virtualización en contenedores

La virtualización basada en contenedores, también llamada virtualización del sistema operativo, es una aproximación a la virtualización en la cual la capa de virtualización se ejecuta como una aplicación en el sistema operativo.

Un contenedor [8] es un conjunto de uno o más procesos aislados del resto del sistema, que acceden sólo a los recursos que se indican. El contenedor encapsula el programa específico y las librerías, mientras que utiliza el sistema operativo del host. Podemos distinguir entre dos tipos de contenedores: [9]

- **A nivel de sistema operativo:** un sistema operativo completo se ejecuta en un espacio aislado dentro de la máquina host, compartiendo el mismo kernel.
- **A nivel de aplicación:** una aplicación o servicio, y los procesos mínimos requeridos por esa aplicación, se ejecutan en un espacio aislado dentro de la máquina host.

Al compartir el mismo kernel del sistema operativo host, un contenedor sólo puede ejecutar procesos en ese sistema operativo. Es decir, un contenedor que se ejecuta en un servidor de Linux, por ejemplo, solo puede ejecutar un sistema operativo Linux, mientras que tal y como habíamos comentado, un hipervisor emula el hardware, lo que permite que varios sistemas operativos (Windows o Linux) se ejecuten simultáneamente en un solo sistema. Además, esta compartición del núcleo hace que el nivel de aislamiento sea menor comparado con las máquinas virtuales, ya que al acceder todos los contenedores al mismo núcleo, si se explota una vulnerabilidad en el núcleo ésta afectaría a todo el sistema, incluidos todos los contenedores.

A cambio, con la virtualización basada en contenedores, no existe la sobrecarga asociada con tener a cada huésped ejecutando un sistema operativo completamente instalado. Este enfoque también puede mejorar el rendimiento porque hay un solo sistema operativo encargándose de los avisos de hardware.

Tal y como se ha mencionado previamente, los contenedores hacen uso del kernel del sistema operativo, del que cabe destacar las siguientes características necesarias para el correcto funcionamiento de este tipo de virtualización: [10]

- **Grupos de control o cgroups.** Se encargan de gestionar los recursos del ordenador asignados a un proceso o conjunto de procesos, como pueden ser el número de *slices* de tiempo de CPU asignadas a cada proceso, el límite de memoria a usar por proceso, los dispositivos de bloques de E/S... Es posible estructurar los recursos en jerarquía.
- **Namespaces.** Permiten aislar procesos entre sí. Los procesos de un contenedor se asignan a un namespace, y el sistema operativo aísla los recursos entre ese namespace y el resto. Hay diferentes tipos de namespace: de PID[4], que permiten mantener los mismos PIDs en diferentes contenedores; mount, para separar sistemas de ficheros; de red para aislar controladores de red...

Los contenedores, por tanto, suponen una virtualización más ligera: usan menos recursos que las máquinas virtuales, además de proporcionar mayor flexibilidad y rapidez de arranque y despliegue. Esto último se debe a que, al usar el mismo kernel que el host, sólo

[4]Process Identifier

están iniciando procesos, algo que es casi instantáneo y que hace muy fluida la ejecución de diferentes máquinas a la vez.

Para contextualizar con el apartado 2.1.1, se muestra una imagen comparativa de la estructura de un contenedor respecto a una máquina virtual:

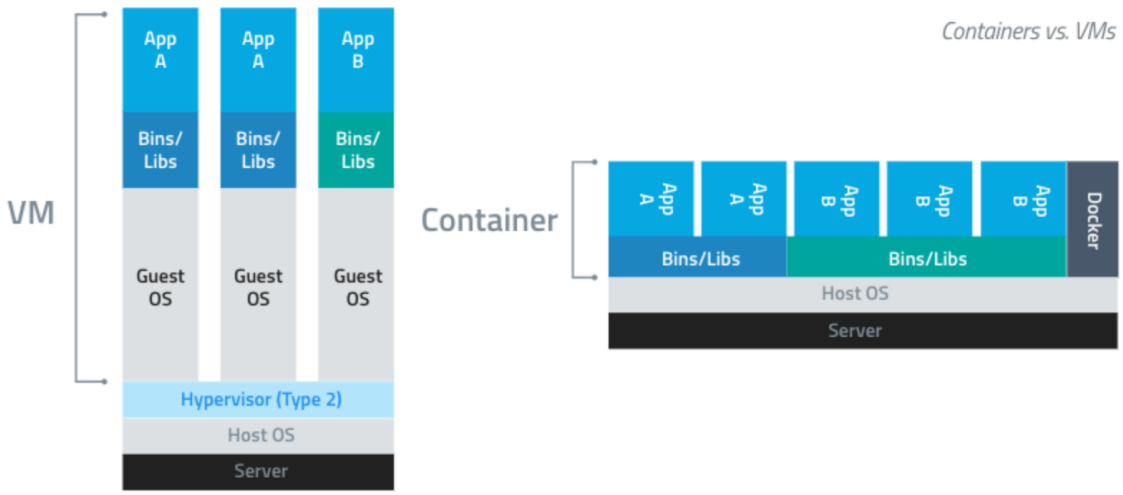

Figura 2.5: Estructura de un contenedor vs VM

LXC

LXC (Linux Containers) [11] es una plataforma de código abierto de contenedores a nivel de sistema operativo. Este tipo de contenedores hacen que un único host Linux actúe como varios host Linux. Esto es debido a que los contenedores LXC incluyen un sistema Linux prácticamente completo, similar a una VM, con su propio sistema de ficheros, espacio de red y aplicaciones. Su objetivo es recrear un entorno lo más parecido posible a una instalación de Linux, pero sin la necesidad de un kernel independiente.

Figura 2.6: Logo de LXC

Chroot es un comando UNIX que permite ejecutar un proceso bajo un directorio raíz simulado, de manera que el proceso no puede acceder a archivos fuera de ese directorio. LXC es similar a un chroot, pero ofrece mucho más aislamiento.

Para crear diferentes contenedores de sistema operativo, se emplean plantillas o templates. Las plantillas proporcionadas en LXC son scripts específicos de un sistema operativo.

LXC se suele usar junto a LXD. LXD ofrece una interfaz para gestionar contenedores LXC como si fueran máquinas virtuales, proporcionando snapshots y control de imágenes, además de otras funcionalidades que incrementan el potencial de LXC. Una de las ventajas principales de LXC es que es una tecnología sencilla de manejar.

Docker

Docker [12] es uno de los proyectos más conocidos y utilizados en este tipo de virtualización. Lejos de ser un sistema operativo como tal, esta plataforma de código abierto hace uso de las funciones de aislamiento de recursos del kernel de Linux para dar lugar a contenedores independientes. Está basada en Linux, pero en los últimos años se ha producido un desarrollo y actualmente también es posible su uso en Windows.

Los contenedores que proporciona Docker son a nivel de aplicación (puede ejecutar aplicaciones normales sin incluir un sistema operativo completo), y su implementación está basada en imágenes, lo que permite compartir una aplicación o un conjunto de servicios, con todas sus dependencias, en varios entornos.

Docker utiliza una arquitectura cliente-servidor. En este tipo de arquitectura, las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y quienes demandan esos recursos o servicios, que son los clientes. Un sistema de contenedores Docker se compone principalmente de 5 elementos: [13]

- **Demonio:** es el proceso principal de la plataforma.
- **Cliente:** binario que constituye la interfaz y que permite al usuario interactuar con el Demonio mediante CLI.
- **Imagen:** plantilla utilizada para crear el contenedor para la aplicación que queremos ejecutar.
- **Registros:** directorios donde se almacenan las imágenes, tanto de acceso público como privado. El más común es Docker Hub.
- **Contenedores:** carpetas donde se almacena todo lo necesario (librerías, dependencias, binarios, etc) para que la aplicación pueda ejecutarse de forma aislada.

Docker Engine es la aplicación cliente-servidor responsable de iniciar y parar los contenidos de una manera similar a como lo hace el hipervisor en una máquina virtual. A continuación se muestra una figura donde se muestran de forma gráfica las interacciones entre los ya mencionados componentes:

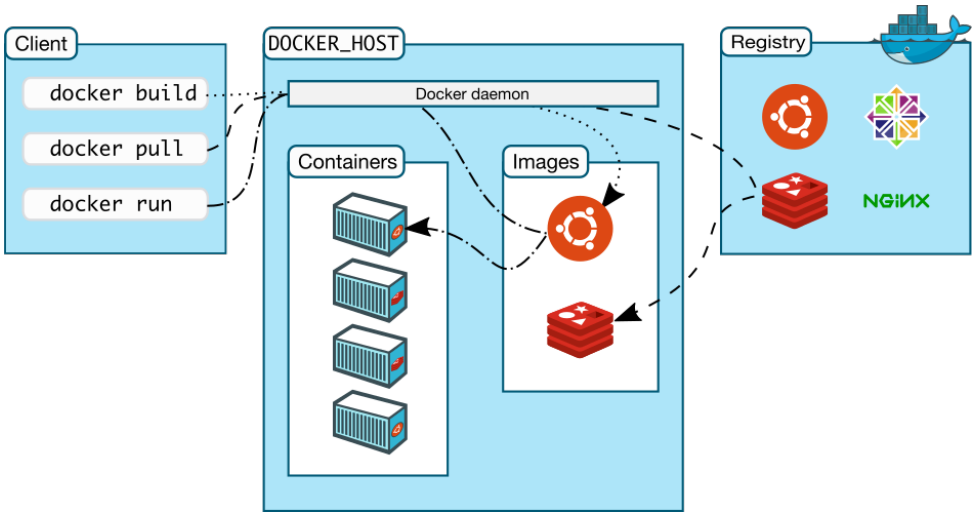

Figura 2.7: Arquitectura del sistema de contenedores Docker

En el ciclo de vida de un contenedor Docker podemos distinguir 5 estados principales:

- **Created:** hace referencia a un contenedor que ha sido creado pero no arrancado.
- **Running/Started:** contenedor corriendo con todos sus procesos.
- **Paused:** contenedor cuyos procesos se han pausado usando la señal SIGSTOP de cgroups.
- **Stopped/Exited:** contenedor cuyos procesos se han parado. La diferencia con *paused* es que se libera la memoria que estaban usando los procesos que han sido detenidos, usando la señal SIGKILL de cgroups. El sistema de ficheros se mantiene tal y como estaba en el momento que se detuvo el contenedor.
- **Deleted/Dead:** contenedor eliminado. Es posible recuperarlo durante un periodo de tiempo determinado.

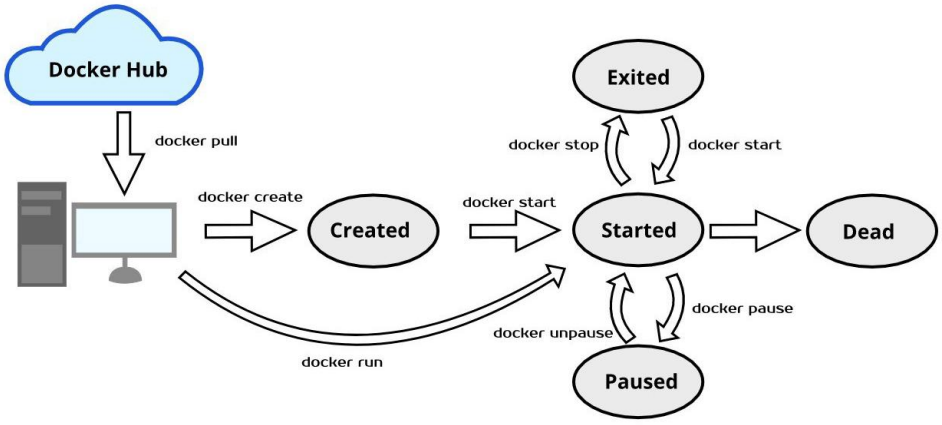

Figura 2.8: Ciclo de vida de los contenedores Docker

Como se puede apreciar en la figura, Docker permite gestionar el estado de los contenedores de forma sencilla mediante la CLI[5], algunas de sus órdenes más destacadas son:

```
$ docker pull #permite descargar una imagen de un repositorio
$ docker create #crea un nuevo contenedor, pero no lo arranca
$ docker start #arranca contenedores parados
$ docker run #es equivalente a create + start
$ docker stop #detiene los procesos corriendo en un contenedor
$ docker pause #pausa los procesos corriendo en un contenedor
$ docker unpause #reanuda la ejecucion de los procesos pausados
$ docker rm #elimina los procesos y el contenedor donde corrian
```

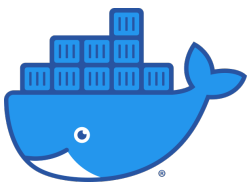

Figura 2.9: Logo de Docker

[5]Command Line Interface

Diferencias entre LXC y Docker

Con todo esto, podemos sacar varias conclusiones importantes acerca de estos dos tipos de contenerización, que condicionarán la elección de uno u otro en función de su uso. La principal sería que mientras que con LXC se virtualiza un sistema operativo completo, con Docker se virtualizan aplicaciones. Además, los contenedores LXC sólo permiten virtualizar entornos Linux y no se pueden portar entre máquinas, mientras que Docker permite portar entre máquinas e incluso plataformas. Esto último es algo relevante puesto que si en algún momento fuese necesario migrar a otro entorno, gracias a la portabilidad de Docker nos ahorraremos el tener que instalar en este nuevo entorno todas aquellas aplicaciones que normalmente usemos.

Finalmente, también cabe recalcar que LXC proporciona un menor aislamiento del sistema operativo respecto a Docker. Al virtualizar sistemas operativos completos y aislados dentro del mismo host, debe haber usuario root y llamadas al propio sistema operativo (si no fuese así, estaríamos ante un sistema «recortado» porque no se podrían hacer determinadas cosas). Docker sirve para virtualizar aplicaciones dentro de un mismo host, por lo que el nivel de acceso root sí puede estar más limitado y controlado, lo que lo hace más seguro.

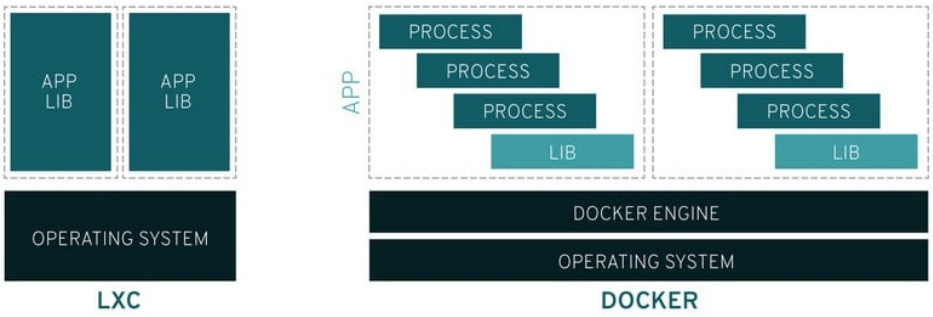

Figura 2.10: Estructura de Docker vs LXC

2.2. Tecnologías de aprovisionamiento

Antes de entrar en materia, se va a presentar la infraestructura como código, un concepto fundamental para el desarrollo de este trabajo.

La infraestructura como código (IaC) permite gestionar y preparar la infraestructura de un entorno con código, en lugar de hacerlo mediante procesos manuales. Con este tipo de infraestructura, se crean archivos de configuración que contienen las especificaciones que esta necesita, lo cual facilita la edición y la distribución de las configuraciones. Asimismo, garantiza que siempre se despliegue el mismo entorno. [14]

La IaC se puede abordar mediante dos enfoques: uno declarativo o uno imperativo:

- **Enfoque declarativo:** define el estado deseado del sistema, incluidas las propiedades que debe tener y los recursos necesarios, y la herramienta de IaC se encarga de configurarlo.
- **Enfoque imperativo:** define los comandos específicos para lograr la configuración deseada, los cuales se deben ejecutar en el orden correcto.

Las tecnologías de virtualización presentadas en el apartado anterior recibirán órdenes del orquestador de escenario (herramienta IaC que veremos más adelante) para desplegar las máquinas necesarias para formar un escenario de red determinado.

Por tanto, podemos decir que el aprovisionamiento consiste en la instalación y la configuración del software (incluido el sistema operativo y las aplicaciones) necesario para que dichas máquinas puedan desempeñar su función en el escenario de red. No es lo mismo que la configuración, aunque ambos son pasos en el proceso de implementación.

Dentro del aprovisionamiento podríamos diferenciar dos tipos, que están muy ligados a los distintos enfoques de la IaC. El primero sería un aprovisionamiento "en frío" o estático, en el que se abastece la máquina antes de arrancarla y levantar el escenario. Por otro lado tendríamos el aprovisionamiento "en caliente" o dinámico, cuya idea principal reside en lanzar órdenes, comandos o depositar archivos en una máquina ya levantada y arrancada.

2.2.1. Aprovisionamiento estático

Este tipo de aprovisionamiento tiene un enfoque imperativo, y, como ya se ha mencionado, se basa en crear de manera offline la máquina con todo lo necesario ya instalado, de forma que cuando se despliegue sólo requiera algunas pequeñas modificaciones adicionales, como podrían ser configuraciones de red. A continuación se va a detallar el uso de Docker y Vagrant para el aprovisionamiento, pero hay otras alternativas a tener en cuenta como las ISO/OVA de VirtualBox.

Docker

Habíamos visto, en el apartado de virtualización basada en contenedores, que una imagen Docker es una instantánea de la aplicación o servicio que corre en un contenedor y de su configuración y las dependencias. En otras palabras, una imagen es un archivo, compuesto por múltiples capas, que constituye una representación estática del estado de un contenedor.

Estas imágenes son las plantillas base desde la que partimos ya sea para crear una nueva imagen o crear nuevos contenedores para ejecutar las aplicaciones. Las imágenes pueden ser locales si se almacenan en el host, o remotas si se suben a repositorios públicos como DockerHub, donde cualquier usuario que lo desee puede hacer uso de ellas. En los repositorios se pueden encontrar tanto imágenes oficiales desde las que partir como imágenes de terceros ya modificadas.

Para crear nuestras propias imágenes, donde dotemos a nuestros contenedores del software así como de su configuración necesaria, Docker nos proporciona los Dockerfile. Un Dockerfile es un archivo de texto plano que contiene una serie de instrucciones necesarias para armar una imagen. A partir de esa imagen podemos crear varias instancias o contenedores, que integran todo lo necesario para ejecutar una aplicación.

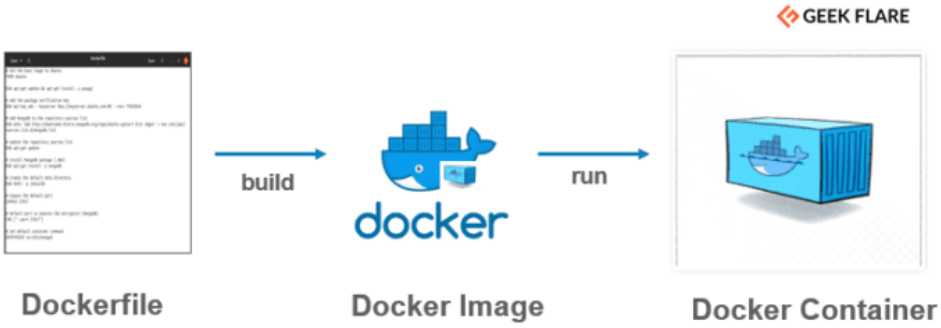

Figura 2.11: Creación de un contenedor Docker a partir de un Dockerfile

Una imagen de Docker consiste en una serie de capas de "solo-lectura", cada una de las cuales se representa con una instrucción del Dockerfile. Las capas se amontonan unas sobre otras y cada una añade algo sobre la anterior. Cuando creamos un contenedor, que no es más que una instancia en ejecución de una imagen, estamos añadiendo una capa escribible encima de todas las demás capas de solo-lectura. Así, la estructura de una imagen Docker se podría representar como sigue:

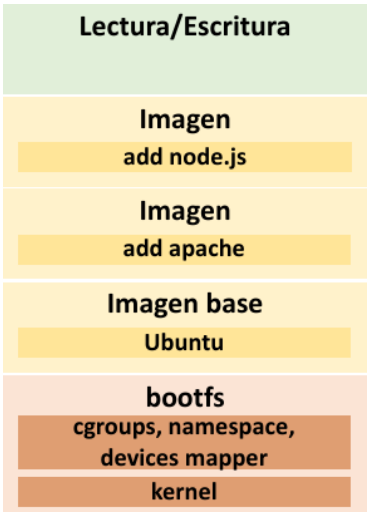

Figura 2.12: Representación de las capas de un Dockerfile

La imagen tendrá una base mínima, que han de tener todas las imágenes para el correcto funcionamiento del contenedor. Esta base incluye el kernel y algunas de sus características necesarias de las que ya hemos hablado como cgroups y namespaces. Encima de la base se representan las capas de sólo lectura, que serían las instrucciones del Dockerfile. En este ejemplo se parte de una imagen Ubuntu a la que se le instala nodejs y apache. Finalmente al crear el contenedor con la instrucción docker run se añade la última capa, correspondiente a los sistemas de ficheros de lectura/escritura, sobre el resto de capas, de forma que podemos interactuar con el contenedor.

La sintaxis de un Dockerfile es sencilla y flexible. Cada instrucción va en una fila, siguiendo el formato `<INSTRUCCIONES><ARGUMENTOS>`, y se ejecuta de manera independiente a las demás. En el Dockerfile del ejemplo anterior se han usado las instrucciones `FROM` para indicar que se parte de una imagen base Ubuntu, y `RUN` para ejecutar comandos en la imagen (correspondientes a la instalación de los paquetes de apache y node). Pero hay multitud de instrucciones más, como por ejemplo `ENV` para establecer variables de entorno persistentes en el contenedor o `COPY` para copiar archivos o directorios de la máquina host al contenedor. Cuando construyamos nuestras propias imágenes, se profundizará más en estas instrucciones y su funcionamiento.

Vagrant

Vagrant [15] es una herramienta de línea de comandos que permite crear y configurar máquinas virtuales a partir de ficheros de configuración llamados Vagrantfile.

Figura 2.13: Logo de Vagrant

Al poseer estos ficheros de configuración, que nos permiten definir los servicios a instalar así como también sus configuraciones, se centraliza toda la configuración de la VM que creamos, de forma que es posible utilizar el mismo Vagrantfile para crear una VM exactamente igual cuantas veces se quiera. Esto es algo muy ventajoso ya que nos permite ahorrar la carga de trabajo que supone el desplegar el mismo entorno una y otra vez, con la seguridad de que nuestro entorno siempre tendrá la misma configuración.

Cabe destacar que vagrant no tiene la capacidad para correr una máquina virtual sino que simplemente se encarga de las características con las que debe crearse esa VM y los complementos a instalar. Para poder trabajar con las máquinas virtuales es necesario la instalación de VirtualBox, Docker, Hyper-v, o la tecnología de virtualización que se desee (y sea compatible[6]).

[6]Ver en la documentación oficial [15]

Un ejemplo sencillo de Vagrantfile sería el siguiente: [16]

```
Vagrant.configure("2") do |config|
    config.vm.box = "genebean/centos6-64bit"
    # network
    config.vm.network "private_network", ip: "192.168.56.100"
    # hardware
    config.vm.provider "virtualbox" do |vb|
        vb.memory = "1024"
        vb.cpus = 1
    end

    config.vm.provision "shell", inline: <<-SHELL
        yum update
        yum install -y nginx
        yum install -y epel-release
        rpm -Uvh https://mirror.webtatic.com/yum/el6/latest.rpm
        yum install -y php56w-fpm php56w-opcache
        yum replace php-common --replace-with=php56w-common
        yum -y install --enablerepo=remi,remi-test mysql mysql-server
   SHELL
end

```

Código 2.1: Ejemplo de Vagrantfile

La instrucción `vagrant up` usaría este fichero para crear una VM en VirtualBox con 1 CPU y una memoria de 1024 MB partiendo de una imagen Centos 6 de 64bit, en la que se realiza la instalación de Nginx, PHP 5.6 y MariaDB utilizando un shell script. El aprovisionamiento sólo se ejecuta al crear la máquina, todas las demás veces que se inicie la máquina virtual no se ejecutará este script, a no ser que se indique expresamente mediante el comando `vagrant provision`. Una vez creada, bastaría con la orden `vagrant ssh` para acceder a ella o bien podríamos detenerla o eliminarla con `vagrant halt` y `vagrant destroy`, respectivamente.

En resumen, Vagrant es muy fácil de instalar y utilizar, permitiéndonos configurar entornos locales solo con un par de comandos. Además, nos proporciona la seguridad de que todos los entornos que creemos con el mismo Vagrantfile serán iguales. Como desventaja, es importante que todos los comandos a ejecutar no necesiten la interacción del usuario, ya que si no, va a fallar.

Vagrant permite realizar algunas configuraciones de red, pero no está pensado para trabajar con grandes cantidades de máquinas virtuales, para infraestructuras más complejas existen otras herramientas como Terraform que pertenece a la misma empresa HashiCorp, y que vamos a presentar posteriormente como una de las tecnologías de orquestación de escenarios.

2.2.2. Aprovisionamiento dinámico

Chef

Ansible

2.3. Tecnologías de orquestación

Docker Compose

Kubernetes

Terraform

Tecnologías de orquestación de proveedores cloud

Bibliografía

[1] VMWare. *¿En qué consiste la virtualización?* [Online]. URL: https://www.vmware.com/es/solutions/virtualization.html.

[2] Microsoft. *¿Qué es virtualización?* [Online]. URL: https://azure.microsoft.com/es-es/overview/what-is-virtualization/.

[3] Red Hat. *¿Qué es un hipervisor?* [Online]. URL: https://www.redhat.com/es/topics/virtualization/what-is-a-hypervisor.

[4] Linux KVM. *Documentación Oficial.* [Online]. URL: https://www.linux-kvm.org/.

[5] VirtualBox. *Documentación Oficial.* [Online]. URL: https://www.virtualbox.org/.

[6] VMware. *Documentación Oficial.* [Online]. URL: https://www.vmware.com/.

[7] MuyComputer. *¿Cuáles son las diferencias entre VirtualBox, VMWare e Hyper-V?* [Online]. URL: https://www.muycomputer.com/2020/03/27/hipervisor-virtualbox-vmware-hyperv/.

[8] Microsoft. *¿Qué es un contenedor?* [Online]. URL: https://azure.microsoft.com/es-es/overview/what-is-a-container/.

[9] S. K. S. "Practical LXC and LXD: Linux Containers for Virtualization and Orchestration". En: (2017). 1st ed.

[10] NGINX. *What Are Namespaces and cgroups, and How Do They Work?* [Online]. URL: https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/.

[11] Linux Containers. *Documentación oficial.* [Online]. URL: https://linuxcontainers.org/.

[12] Docker. *Documentación oficial.* [Online]. URL: https://www.docker.com/.

[13] Redes Zone. *Docker y cómo funciona la virtualización de contenedores.* [Online]. URL: https://www.redeszone.net/2016/02/24/docker-funciona-la-virtualizacion-contenedores/.

[14] Red Hat. *¿Qué es la infraestructura como código (IaC)?* [Online]. URL: https://www.redhat.com/es/topics/automation/what-is-infrastructure-as-code-iac.

[15] Vagrant. *Documentación oficial.* [Online]. URL: https://www.vagrantup.com/.

[16] Estamos Rodeados. *Provisioning con Vagrant.* [Online]. URL: https://estamosrodeados.com/linux/vagrant-provisioning/.