

Sistemas Informáticos

Unidad 04. Máquinas virtuales y contenedores



Autores: Sergi García, Alfredo Oltra

Actualizado Septiembre 2025



Licencia



Reconocimiento - No comercial - CompartirIgual (BY-NC-SA): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se ha de hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán diferentes símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 **Importante**

 **Atención**

 **Interesante**

ÍNDICE

1. Virtualización	3
2. Definición de máquina virtual	3
3. Tipos de máquinas virtuales	4
3.1 Máquinas virtuales de sistema (System Virtual Machines, SVM)	4
3.2 Máquinas virtuales de proceso (Process Virtual Machines, PVM)	5
4. Ventajas/Desventajas de las máquinas virtuales	6
5. Como configurar How to configure a Virtual machine: VirtualBox	6
6. Máquinas virtual vs Contenedores	7
6.1 ¿Contenedores o máquinas virtuales?	8
7. Contenedores: Docker	8
7.1 Como funciona Docker internamente	8
7.2 Ventajas de Docker	9
8. Instalando Docker	9
8.1 Comandos básicos útiles de Docker	10
8.2 Docker Compose	11
9. Bibliografía	11

UNIDAD 04. MÁQUINAS VIRTUALES Y CONTENEDORES

1. VIRTUALIZACIÓN

La **virtualización** es un conjunto de técnicas de hardware y software que permiten **abstraer el hardware y el software reales** para **simular recursos físicos, sistemas operativos u otros entornos informáticos**.

En otras palabras, consiste en crear una capa intermedia entre los recursos físicos de un ordenador (CPU, memoria, disco, dispositivos) y las aplicaciones que los utilizan, de modo que se puedan ejecutar de manera aislada y flexible.

La virtualización se utiliza de forma habitual en **instituciones, empresas, centros educativos** y también en el ámbito doméstico, ya que resulta relativamente sencilla de implementar y aporta numerosas ventajas:

- Mejor aprovechamiento de los recursos.
- Reducción de costes en hardware.
- Mayor seguridad y aislamiento entre entornos.
- Flexibilidad para pruebas y despliegues.
- Escalabilidad en entornos de servidores y nube.

Un ejemplo clásico de virtualización son las **máquinas virtuales**, aunque también existen otras formas como la **emulación de consolas de videojuegos** o los **contenedores** (Docker, Podman, etc.).

Es importante recordar que la virtualización **no se limita a un único producto o herramienta**, sino que es un **conjunto de técnicas** que hacen posibles estas soluciones.

2. DEFINICIÓN DE MÁQUINA VIRTUAL

A veces necesitamos probar un nuevo sistema operativo, una configuración o un programa en un entorno limpio, sin alterar el ordenador principal. ¿Cómo hacerlo de manera sencilla?

La respuesta es utilizar una **máquina virtual (VM)**.

Una **máquina virtual** es un software que permite **simular un ordenador completo**, con su propio sistema operativo, en el que podemos instalar aplicaciones, cambiar configuraciones o realizar pruebas de manera segura y aislada.

Las máquinas virtuales se crean gracias a un **programa de virtualización** que se ejecuta sobre el sistema operativo de la máquina real (host).

Ejemplos de programas para crear y gestionar máquinas virtuales:

- **VirtualBox** (gratuito).
- **VMware Workstation / VMware Player**.
- **Microsoft VirtualPC** (más antiguo).
- **Parallels** (muy usado en Mac).
- **Java Virtual Machine (JVM)** y **.NET CLR**, que virtualizan procesos, no sistemas completos.

3. TIPOS DE MÁQUINAS VIRTUALES

Según su funcionalidad, las máquinas virtuales se pueden clasificar en dos grandes categorías:

- **Máquinas virtuales de sistema (System Virtual Machines, SVM)**
- **Máquinas virtuales de proceso (Process Virtual Machines, PVM)**

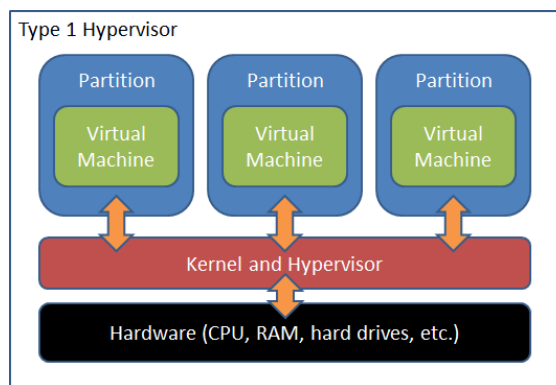
3.1 Máquinas virtuales de sistema (System Virtual Machines, SVM)

Las **máquinas virtuales de sistema** permiten **replicar una máquina real en varias máquinas virtuales**, cada una con su propio sistema operativo independiente.

El software encargado de gestionar esta virtualización se denomina **hipervisor**.

Existen **dos tipos principales de hipervisores**:

- **Hipervisor tipo 1 (bare-metal o nativo)**
 - El hipervisor se ejecuta directamente sobre el hardware físico, sin necesidad de un sistema operativo anfitrión.
 - Ofrece mejor rendimiento y estabilidad, ya que controla directamente los recursos de la máquina.
 - Suele emplearse en entornos profesionales, servidores y centros de datos.
- **Ejemplos de hipervisor tipo 1:**
 - VMware ESXi (gratuito).
 - VMware ESX.
 - Xen (software libre).
 - Citrix XenServer (gratuito).
 - Microsoft Hyper-V Server (gratuito).

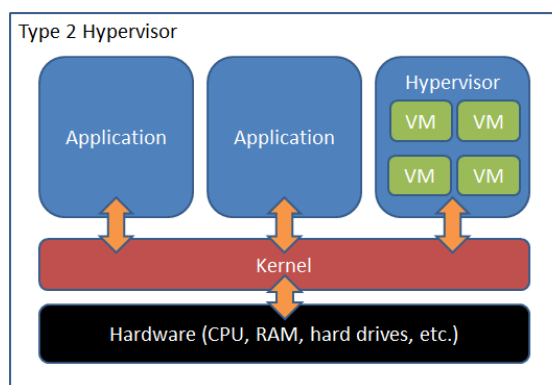


- **Hipervisor tipo 2 (hosted o alojado)**

- El hipervisor se instala como una aplicación dentro de un sistema operativo anfitrión (host).
- Este sistema anfitrión gestiona los recursos físicos, mientras que el hipervisor los distribuye entre las máquinas virtuales invitadas (guest).
- Es el tipo de hipervisor más utilizado en entornos educativos, domésticos y de pruebas.
- Desventaja principal: los sistemas invitados comparten recursos con el sistema anfitrión, por lo que el rendimiento es inferior al de un hipervisor tipo 1.

- **Ejemplos de hipervisor tipo 2:**

- VirtualBox (gratuito).
- VMware Workstation.
- VMware Player (gratuito).
- QEMU (gratuito).



⚠ Atención: en nuestra asignatura utilizaremos un hipervisor tipo 2. En los apuntes lo denominaremos simplemente “Máquina Virtual”.

3.2 Máquinas virtuales de proceso (Process Virtual Machines, PVM)

Las **máquinas virtuales de proceso** funcionan como una **aplicación normal dentro de un sistema operativo anfitrión**, y están diseñadas para **ejecutar un único proceso**.

- Se **crean cuando el proceso se inicia** y se **eliminan al finalizar**.
- Su propósito principal es proporcionar un **entorno de programación independiente de la plataforma**, es decir, que el mismo programa pueda ejecutarse en diferentes sistemas operativos.

Ejemplos de máquinas virtuales de proceso:

- **Java Virtual Machine (JVM):** el compilador de Java genera “bytecodes”, que son interpretados por la JVM en cualquier sistema operativo donde esté instalada.
- **.NET CLR:** permite ejecutar aplicaciones desarrolladas en .NET en diferentes entornos (Windows, Linux mediante Mono, etc.)

4. VENTAJAS/DESVENTAJAS DE LAS MÁQUINAS VIRTUALES

Ventajas

- Posibilidad de usar varios sistemas operativos al mismo tiempo.
- Probar un sistema operativo antes de instalarlo en un equipo real.
- Ejecutar aplicaciones que no están disponibles en el sistema anfitrión.
- Emular arquitecturas diferentes (otros conjuntos de instrucciones).
- Crear entornos de prueba aislados.
- Guardar el estado de una máquina y restaurarlo posteriormente.
- Ahorro de energía, recursos y espacio, permitiendo emular ordenadores antiguos sin necesidad de hardware físico.
- Facilidad para clonar o hacer copias de seguridad.
- Contribuyen a la sostenibilidad, al evitar fabricar y desechar ordenadores y componentes innecesarios.

Desventajas

- Los recursos deben compartirse entre el sistema anfitrión y las máquinas virtuales invitadas.
- El rendimiento suele ser inferior al de una máquina física real.

5. COMO CONFIGURAR HOW TO CONFIGURE A VIRTUAL MACHINE: VIRTUALBOX

VirtualBox es un **hipervisor de tipo 2** completamente gratuito y disponible para los sistemas operativos más populares (Windows, Linux, macOS).

Puedes descargarlo directamente desde su página oficial:

👉 <https://www.virtualbox.org/>

Recursos y tutoriales

En Internet existen numerosos tutoriales que explican, paso a paso, cómo instalar y configurar VirtualBox en diferentes sistemas. Algunos ejemplos:

- **Instalar VirtualBox en Ubuntu:** 🎥 <https://www.youtube.com/watch?v=QkJmahizwO4>
- **Instalar VirtualBox en Windows 10:** 🎥 <https://youtu.be/8mns5yqMfZk>
- **Cómo crear una máquina virtual en VirtualBox:** 🎥 https://youtu.be/H_ustCy4Ks8

Mejorando el rendimiento: “Guest Additions”

Para mejorar la experiencia de uso y el rendimiento de las máquinas virtuales, VirtualBox ofrece una herramienta muy útil llamada **Guest Additions**.

Este paquete de utilidades se instala dentro del sistema operativo invitado (*guest OS*) y proporciona:

- Mejor integración entre el sistema anfitrión y el invitado.
- Soporte para **pantalla completa** y **resoluciones dinámicas**.
- Posibilidad de **compartir carpetas** y archivos fácilmente.
- Integración del **portapapeles** (copiar/pegar entre host e invitado).
- Mejoras en el rendimiento gráfico y de red.

Por estas razones, **se recomienda encarecidamente instalarlo en cada máquina virtual** que crees.

Tutoriales recomendados para instalar Guest Additions:

- En Ubuntu (Linux):  <https://youtu.be/Q84boOmiPW8>
- En Windows 10:  https://youtu.be/Bb_kJd3ISxQ

6. MÁQUINAS VIRTUALES VS CONTENEDORES

Las máquinas virtuales funcionan gracias a un hipervisor que se ejecuta sobre un hardware físico, simulando uno o varios “hardware virtuales”.

Cada uno de estos hardware virtuales permite ejecutar máquinas virtuales independientes, cada una con su propio sistema operativo.

La virtualización ha tenido un enorme éxito en los últimos años para el despliegue de aplicaciones y la provisión de infraestructuras. Sin embargo, en la actualidad el paradigma que predomina es el de los contenedores ligeros (lightweight containers), o simplemente contenedores.

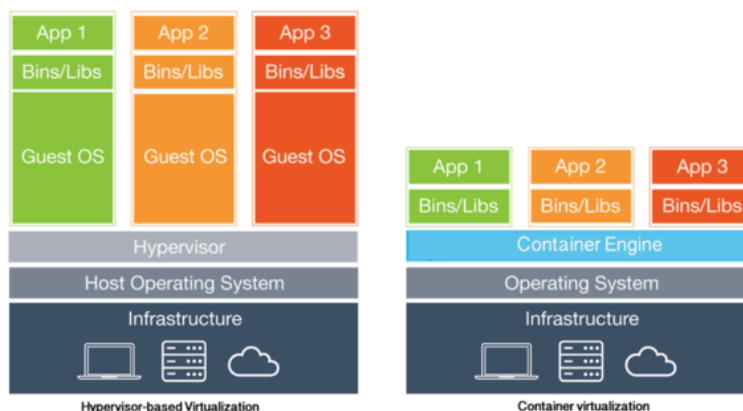
¿Qué son los contenedores?

Los **contenedores** son una tecnología similar a las máquinas virtuales, pero con una diferencia fundamental:

- En lugar de utilizar un hipervisor y replicar hardware virtual, **los contenedores comparten un mismo sistema operativo anfitrión (host OS)**.
- Cada contenedor es, en realidad, un **entorno aislado** dentro del sistema anfitrión.
- No se virtualiza hardware, sino que se aprovechan los recursos del sistema operativo compartido.

Ventajas de los contenedores frente a las máquinas virtuales

- Mayor **rapidez** (no existe la capa de virtualización del hardware).
- **Tamaño mucho más reducido**, lo que permite:
 - Migrarlos fácilmente.
 - Arrancarlos y detenerlos con rapidez.
 - Recuperarlos o replicarlos de manera eficiente.
 - Desplegarlos en la **nube** sin apenas cambios.



6.1 ¿Contenedores o máquinas virtuales?

En resumen, es útil ejecutar un **contenedor** si queremos:

- Obtener **mejor rendimiento** que con una máquina virtual clásica.
- Desarrollar aplicaciones que puedan **distribuirse fácilmente**, sin problemas de dependencias ni configuraciones, siempre dentro de un mismo sistema operativo que el del host.
- Crear aplicaciones **portables a la nube** con cambios mínimos.
- Ejecutar **múltiples copias de una misma aplicación** o conjuntos de aplicaciones que colaboran entre sí (microservicios).

En cambio, debemos usar **máquinas virtuales** si necesitamos:

- **Flexibilidad**, por ejemplo, utilizar un sistema operativo distinto al del host.
- Ejecutar **aplicaciones muy diferentes entre sí** en sistemas operativos heterogéneos.

7. CONTENEDORES: DOCKER


Existen diferentes tecnologías que permiten usar contenedores en Linux, como **LXC** o **LXD**, aunque la más popular hoy en día es **Docker**.

 Más información sobre las diferencias entre tecnologías de contenedores:

<https://unix.stackexchange.com/questions/254956/what-is-the-difference-between-docker-lxd-and-lxc>

Docker es un proyecto **open source** que automatiza el despliegue de aplicaciones dentro de contenedores software.

- Proporciona una capa de **abstracción y automatización** sobre la virtualización a nivel de sistema operativo en Linux.
- Se ha convertido en un estándar de facto para el desarrollo moderno basado en microservicios.

 **Interesante:** tienes mucha información disponible sobre como usar Docker en <https://sergarb1.github.io/CursoIntroduccionADocker/>.

7.1 Como funciona Docker internamente

Docker aprovecha funcionalidades del **kernel de Linux** para aislar recursos y garantizar independencia entre procesos:

- **cgroups (control groups):** permiten limitar, contabilizar y aislar el uso de recursos (CPU, memoria, disco, red, etc.) entre distintos procesos.
- **namespaces:** definen qué recursos puede ver y usar cada conjunto de procesos.

Gracias a estas características, múltiples contenedores pueden ejecutarse de forma **independiente** en una única instancia de Linux, **evitando la sobrecarga de iniciar y mantener máquinas virtuales completas**.

Con Docker podemos:

- Crear contenedores con *docker run*.
- Conectarnos a un contenedor desde la consola con *docker attach*.
- Copiar archivos entre el host y el contenedor con *docker cp*.

📌 Por defecto, Docker está orientado a procesos y **no soporta interfaces gráficas**. Sin embargo, existen alternativas:

- Instalar un **servidor X** y conectarse mediante un cliente **XWindows**.
- Utilizar programas de administración remota como **VNC**. Una opción recomendable es instalar **NoVNC** dentro del contenedor, lo que permite gestionarlo directamente desde el navegador sin clientes adicionales.

📖 Más información: 🖱️ [https://es.wikipedia.org/wiki/Docker_\(software\)](https://es.wikipedia.org/wiki/Docker_(software))

7.2 Ventajas de Docker

Algunas de las principales ventajas de Docker son:

- **Independencia de la plataforma:** puede ejecutarse en Windows, macOS o Linux (siempre que haya compatibilidad).
 - En Windows con capacidades Linux funciona en modo nativo.
 - En otros casos, se instala una máquina virtual ligera (por ejemplo con VirtualBox) y dentro de ella se lanza Docker.
- **Facilidad de uso:** crear y arrancar un contenedor es muy sencillo.
- **Entornos de red independientes:** cada contenedor tiene su propia red, configurable y con opción de compartirla con otros contenedores.
- **Imágenes Docker:** funcionan como “plantillas” desde las que podemos crear múltiples contenedores.
 - Podemos usar imágenes oficiales, creadas por terceros o crear las nuestras.
- **Control de versiones del software interno:** dentro de un contenedor controlamos versión del sistema operativo, base de datos, servidor de aplicaciones, etc. Esto elimina problemas de configuración al portar un sistema de un equipo a otro.
- **Escalabilidad y alta disponibilidad:** soporta orquestadores como **Kubernetes** o **Docker Swarm** en entornos de producción.
- **Repositorio oficial de imágenes (Docker Hub):** con un potente buscador que incluye imágenes oficiales y compartidas por la comunidad.
- **Compatibilidad en la nube:** está soportado por las principales plataformas: **Azure, AWS, Google Cloud, OVH**, entre otras.
- **Uso en VPS:** muchos servidores virtuales privados realmente son contenedores.

8. INSTALANDO DOCKER

⚠️ **Atención:** aunque es posible instalar Docker en Windows o macOS, no es recomendable hacerlo directamente. Para fines educativos, es mejor instalar una máquina virtual Linux y, dentro de ella, configurar Docker. Esto evita problemas de compatibilidad y asegura un entorno más estable. Hazlo bajo tu propio riesgo.

- **Instalar Docker en Ubuntu (Linux):**

🖱️ <https://docs.docker.com/engine/install/ubuntu/>

👤 <https://www.youtube.com/watch?v=wCSMDtHPBso>

- **Instalar Docker en Windows:**
👉 <https://docs.docker.com/desktop/install/windows-install/>
- **Instalar Docker en macOS:**
👉 <https://docs.docker.com/desktop/install/mac-install/>

8.1 Comandos básicos útiles de Docker

📖 Referencia oficial: 👉 <https://docs.docker.com/reference/>

A continuación, algunos de los comandos más usados en Docker:

- **docker run -it imagen**
Crea un contenedor a partir de la imagen indicada.
 - Si la imagen no existe en tu máquina, **se descarga automáticamente desde Docker Hub**.
 - El parámetro **-it** enlaza la entrada/salida del contenedor con tu consola actual.
⚠ **Cuidado:** si ejecutas este comando varias veces, estarás creando varios contenedores distintos.
👉 Para reutilizar un contenedor ya creado usa **docker start**.
- **docker start -i contenedor**
Inicia un contenedor ya creado.
 - El parámetro **-i** conecta la entrada del contenedor a la consola actual.
- **docker ps**
Muestra únicamente los contenedores que están en ejecución.
- **docker ps -a**
Muestra **todos** los contenedores, estén en ejecución o no.
- **docker image ls**
Lista las imágenes descargadas en tu máquina.
⚠ No confundir **imágenes** con **contenedores**:
 - **Imágenes** = plantillas base.
 - **Contenedores** = instancias en ejecución de esas imágenes.
- **docker cp origen destino**
Permite copiar archivos entre el host (tu máquina real) y un contenedor Docker.
- **docker login**
Inicia sesión en **Docker Hub** desde la consola.
- **docker pull nombre_imagen:etiqueta**
Descarga una imagen desde Docker Hub.
Ejemplo: **docker pull ubuntu:22.04**
- **docker commit etiqueta**
Crea una nueva imagen a partir de los cambios realizados en un contenedor.
- **docker push nombre_imagen:etiqueta**
Sube tu imagen modificada a Docker Hub.

🔗 Recurso adicional - Cheatsheet de Docker:

<https://raw.githubusercontent.com/sergarb1/CursoIntroduccionADocker/main/FuentesCurso/Docker%20CheatSheet%20COMPLETA.pdf>

8.2. Docker Compose

Docker Compose es una herramienta que permite configurar y ejecutar uno o varios contenedores mediante un archivo en **formato YAML**.

✓ Ventajas:

- Muy fácil de usar.
- Permite levantar o eliminar entornos completos rápidamente.
- Ideal para aplicaciones con varios servicios (por ejemplo: base de datos + servidor web + backend).

📖 Más información oficial: 👉 <https://docs.docker.com/compose/>

🔗 Ejemplos prácticos en el curso: 👉 <https://sergarb1.github.io/CursoIntroduccionADocker/>

9. BIBLIOGRAFÍA

[1] Virtualization

<https://en.wikipedia.org/wiki/Virtualization>

[2] Hypervisors

<https://en.wikipedia.org/wiki/Hypervisor>

[3] Virtual machine

https://en.wikipedia.org/wiki/Virtual_machine

[4] Docker

[https://en.wikipedia.org/wiki/Docker_\(software\)](https://en.wikipedia.org/wiki/Docker_(software))

[5] Curso de introducción a Docker

<https://sergarb1.github.io/CursoIntroduccionADocker/>