

Introducción a Docker y Kubernetes

# UD 01. Introducción a los contenedores y a Docker

---



Autor: Sergi García Barea

Actualizado Enero 2025

## Licencia



**Reconocimiento – NoComercial - 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 debe hacer con una licencia igual a la que regula la obra original.

## Nomenclatura

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

### Importante

### Atención

### Interesante

<b>1. Introducción</b>	<b>3</b>
<b>2. Conceptos previos</b>	<b>3</b>
2.1 Virtualización	3
2.2 ¿Qué es una máquina virtual?	3
2.3 ¿Qué es una máquina virtual de proceso?	3
2.4 ¿Qué es un emulador?	3
2.5 ¿Qué es un hipervisor?	4
<b>3. Contenedores</b>	<b>4</b>
3.1 ¿Qué son los contenedores?	4
3.2 Analogía con contenedores de transporte marítimo	5
3.3 Contenedores para desarrollo y despliegue de aplicaciones	5
3.4 Contenedores para despliegue de servicios	5
3.5 Ventajas e inconvenientes del uso de contenedores	6
3.6 En resumen, ¿Cuándo es adecuado usar contenedores?	6
<b>4. Contenedores en sistemas Linux</b>	<b>7</b>
4.1 ¿Es nuevo el concepto de entornos privados en sistemas Unix?	7
4.2 Sistemas privados modernos en Linux: contenedores	7
4.3 ¿Cómo funcionan los contenedores modernos en Linux?	7
4.4 ¿Puedo poner en marcha un contenedor Linux "A mano"?	7
4.5 Los contenedores Linux ¿Pueden funcionar en sistemas como Windows o MacOS?	8
<b>5. Contenedores Docker</b>	<b>8</b>
5.1 ¿Qué es Docker?	8
5.3 Docker en sistemas Windows y MacOS	9
5.4 Docker corriendo contenedores Windows Server Core y contenedores MacOS	10
<b>6. Conclusión</b>	<b>10</b>
<b>7. Bibliografía</b>	<b>10</b>
<b>8. Licencias de elementos externos utilizados</b>	<b>10</b>

## UD01. INTRODUCCIÓN A LOS CONTENEDORES Y A DOCKER

### 1. INTRODUCCIÓN

En esta unidad realizaremos una introducción al concepto de contenedores. Nos centraremos en contenedores Linux y en concreto en la tecnología de Docker.

### 2. CONCEPTOS PREVIOS

#### 2.1 Virtualización

La virtualización es un conjunto de tecnologías de hardware y software que permiten la abstracción de hardware, creando así la “ilusión” de administrar recursos virtuales como si fueran recursos reales, de forma transparente para los usuarios.

La virtualización es muy utilizada para el despliegue de sistemas, desarrollo de software, análisis de malware, escalado horizontal, etc. Ya que es relativamente sencilla de implementar y puede ahorrar significativamente costes (consumo de energía, mantenimiento, etc.)

#### 2.2 ¿Qué es una máquina virtual?

A veces, necesitamos probar un nuevo sistema operativo, una determinada configuración, probar a desplegar un software, etc. pero no está disponible para ello una máquina real. La creación de una máquina virtual utilizando técnicas de virtualización es la solución a este problema.

De este modo, una máquina virtual permite simular una máquina (con su sistema operativo) y ejecutar programas como si estuvieran utilizando una máquina real e independiente.

Para la creación de máquinas virtuales generalmente existen varios tipos de tecnologías:

- Máquinas virtuales de proceso.
- Emuladores.
- Hipervisores.
- Contenedores. **Docker se engloba en esta categoría.**

#### 2.3 ¿Qué es una máquina virtual de proceso?

Las máquinas virtuales de proceso, son un tipo de máquinas virtuales que permiten ejecutar un programa diseñado para un sistema operativo/arquitectura concreta (distinta de la máquina actual), como un proceso más de nuestra máquina actual.

Esto se consigue implementando una máquina virtual de proceso que emula la arquitectura necesaria. Teóricamente, podremos lanzar nuestro programa en cualquier sistema que tenga la máquina virtual de proceso implementada.

Algunos de los principales ejemplos de este tipo de virtualización son:

- Máquina virtual de Java (JVM): ejecuta los bytecodes de Java en cualquier sistema y arquitectura que la tenga implementada.
- Plataforma .NET: análoga a la máquina virtual de Java con productos Microsoft.

#### 2.4 ¿Qué es un emulador?

Un emulador es un software encargado de emular un hardware completo muy específico (por ejemplo, emuladores de videoconsolas antiguas) o una API concreta (por ejemplo, Wine, un software que emula la API de sistemas Windows en otros sistemas operativos, permitiendo ejecutar programas diseñados para Windows en ellos).

## 2.5 ¿Qué es un hipervisor?

Un hipervisor es una máquina virtual que simula total o parcialmente un hardware de una máquina, permitiendo la instalación de distintos sistemas operativos (por ejemplo, virtualizar un sistema Windows 10 Home en una máquina real Linux).

Algunos software conocidos que implementan un hipervisor son: Virtualbox, VMWare, emuladores de consolas, etc.

Para saber más: <https://es.wikipedia.org/wiki/Hipervisor>

## 3. CONTENEDORES

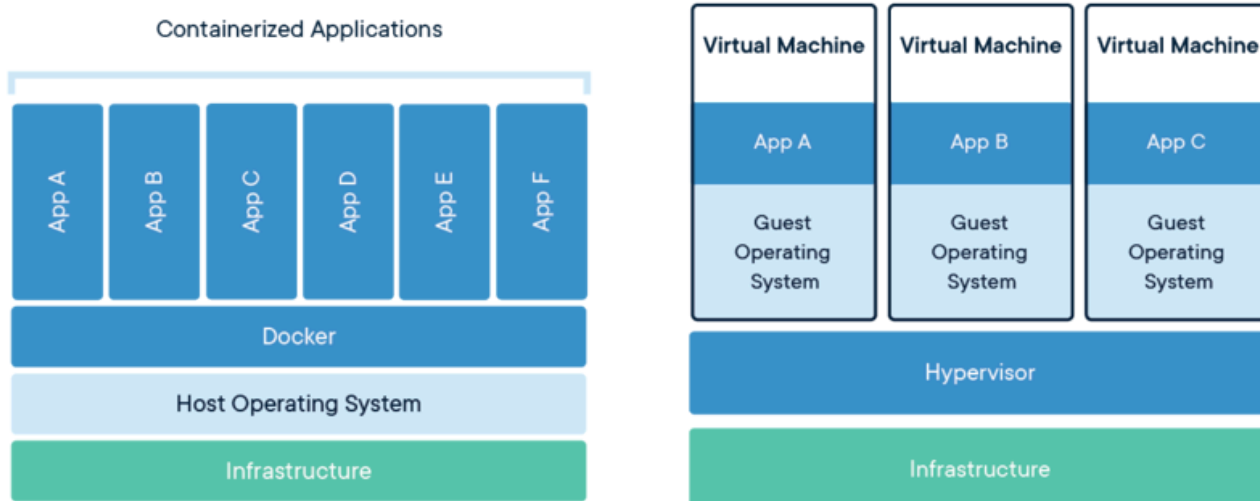
### 3.1 ¿Qué son los contenedores?

Los contenedores son una tecnología de virtualización, que al contrario que un hipervisor (que trata de emular un sistema completo), utiliza el sistema base de la máquina anfitrión y actúa realmente como un “entorno privado” que comparte recursos con el sistema anfitrión, sin virtualizar el hardware completo. En concreto, los contenedores suelen tener entornos privados aislados **a nivel de procesos, memoria, sistema de ficheros y red**.

Técnicamente, los contenedores son un tipo de virtualización englobada en lo que se llama “OS Level virtualization”. Para saber más: [https://en.wikipedia.org/wiki/OS-level\\_virtualization](https://en.wikipedia.org/wiki/OS-level_virtualization)

**! Atención:** esto implica, que, de forma nativa, no puedes ejecutar un contenedor en un sistema operativo distinto del que utiliza la tecnología de contenedores.

La siguiente imagen puede ayudarnos a entender el concepto de contenedor.



Fuente imagen: <https://commons.wikimedia.org/wiki/File:Docker-containerized-and-vm-transparent-bq.png>

A la derecha observamos el funcionamiento de un hipervisor, encargado de virtualizar el hardware y donde cada máquina virtual tiene su propio sistema operativo. A la izquierda, observamos un sistema de contenedores, donde no existe esa virtualización del hardware y cada contenedor es un entorno privado.

### 3.2 Analogía con contenedores de transporte marítimo



Fuente imagen: <https://www.flickr.com/photos/68359921@N08/50125348052/>

Para facilitar la comprensión del funcionamiento de los contenedores, vamos a hacer una analogía con los contenedores de transporte marítimo:

- Los contenedores de transporte marítimo, deben cumplir unos estándares (tamaño, peso y forma) para ser transportados.
  - Lo mismo ocurre con los contenedores en virtualización. Mientras cumplan un estándar, pueden ser virtualizados en cualquier máquina que lo soporte (local, servidor, etc.).
- Una vez cumplido el estándar, el tipo de carga del contenedor marítimo es independiente.
  - Lo mismo ocurre con los contenedores en virtualización. Si se cumple el estándar, el software que contenga podrá ser ejecutado sin problemas

### 3.3 Contenedores para desarrollo y despliegue de aplicaciones

Uno de los principales usos de los contenedores (aunque no el único) es facilitar el desarrollo, distribución y el despliegue de aplicaciones.

- Compilar software es tedioso. Utilizando un contenedor, tenemos el entorno de compilación/depuración montado con las versiones que necesitamos.
- Usar contenedores facilita el testeo, permitiendo la creación de distintos entornos de prueba con diferentes configuraciones, etc.
- Los contenedores nos evitan problemas de compatibilidad al desplegar nuestras aplicaciones, teniendo siempre las versiones adecuadas para ejecutar nuestro software.

**Interesante:** Muchos sistemas de CI/CD (Continuous Integration/Continuous Delivery) se basan en el uso de contenedores. Más información en <https://es.wikipedia.org/wiki/CI/CD>

### 3.4 Contenedores para despliegue de servicios

Otro de los principales usos de los contenedores es el despliegue de servidores de distinto tipo (web, correo, bases de datos, DNS, etc.).

Además de las ventajas anteriormente citadas de mantener versiones de software, los contenedores nos permiten unificar configuraciones de servidores en local, incluso involucrando a distintos servicios en distintos contenedores, de forma que al desplegarlos en la nube, funcionen exactamente igual que en las pruebas realizadas localmente.

**! Atención:** *“En mi máquina funcionaba... falla solo al subirlo al servidor...”*. El uso de contenedores contribuye a que esta situación desaparezca :)

Además, los contenedores facilitan el “escalado horizontal” de servicios, especialmente si se apoyan de herramientas llamadas orquestadores.

Para saber más [https://es.wikipedia.org/wiki/Escalabilidad#Escalabilidad\\_horizontal](https://es.wikipedia.org/wiki/Escalabilidad#Escalabilidad_horizontal)

### 3.5 Ventajas e inconvenientes del uso de contenedores

Algunas de las ventajas del uso de contenedores son:

- Los contenedores ocupan menos espacio, al no tener que replicar en cada uno el sistema operativo que están virtualizando, ya que utilizan el sistema de la máquina anfitrión.
- Al no tener que realizar una virtualización de hardware, la ejecución del software de los contenedores es mucho más rápida, con velocidades cercanas a las nativas.
- Multitud de empresas de software (Microsoft, Apache, Nginx, MySQL, Oracle, Wordpress, Moodle, y un largo etc.) apoyan estas tecnologías y dan soporte tanto incorporando sistemas de contenedores a sus sistemas operativos, como ofreciendo imágenes oficiales de sus productos para que con una sencilla orden, se pueda poner en marcha alguno de sus servicios o aplicaciones.

Algunas de las principales desventajas de los contenedores son:

- Pese a que mejoran enormemente la velocidad respecto a una virtualización por hipervisor, siguen teniendo un rendimiento peor que una ejecución “bare metal” (sobre un sistema real), ya que el aislamiento consume recursos.
- La persistencia y el acceso/modificación a datos persistentes entre contenedores es más tedioso que si se realiza sobre una máquina real.
- Los contenedores están pensados generalmente para el uso vía línea de comandos. Aunque técnicamente es posible configurar los contenedores para tener su propio entorno gráfico, este proceso es tedioso.

### 3.6 En resumen, ¿Cuándo es adecuado usar contenedores?

El uso de contenedores, suele ser adecuado en los contextos:

- Como usuarios: queremos probar algo rápido y sin complicarnos mucho en la configuración (por ejemplo, montar un servicio en local para aprender).
  - Para ello, podemos utilizar servicios de distribución de imágenes de contenedores públicas como Docker Hub <https://hub.docker.com/>
- Como desarrolladores: queremos desarrollar una aplicación que se pueda distribuir en local o desplegar en la nube sin problemas de configuración
  - Podemos usar contenedores, tanto para tener el entorno de desarrollo listo, como para distribuir la aplicación en sí.
- Queremos testear nuestra aplicación con distintas configuraciones, límites de recursos, juegos de prueba, etc.
  - Útil para generar entornos de prueba y despliegue utilizando CI/CD (Continuous Integration/Continuous Delivery) <https://es.wikipedia.org/wiki/CI/CD>
- Queremos realizar “escalado horizontal” de servicios, es decir, ejecutar múltiples copias de una misma aplicación/conjunto de aplicaciones que funcionan como un clúster.
  - [https://es.wikipedia.org/wiki/Escalabilidad#Escalabilidad\\_horizontal](https://es.wikipedia.org/wiki/Escalabilidad#Escalabilidad_horizontal)

## 4. CONTENEDORES EN SISTEMAS LINUX

### 4.1 ¿Es nuevo el concepto de entornos privados en sistemas Unix?

El concepto de entornos privados, utilizado en los controladores, no es algo novedoso de los sistemas Unix modernos. Desde hace muchos años existían algunas soluciones tales como:

- Chroot (Sistemas Unix): <https://es.wikipedia.org/wiki/Chroot> (1982)
- Jail (FreeBSD): [https://es.wikipedia.org/wiki/FreeBSD\\_jail](https://es.wikipedia.org/wiki/FreeBSD_jail) (1999)

Estas utilidades son los “abuelos” del concepto actual de contenedor en los sistemas Unix.

### 4.2 Sistemas privados modernos en Linux: contenedores

En el año 2008, aparecen los contenedores modernos de sistemas Linux, con el sistema LXC (Linux Container). Para su desarrollo, se introdujeron nuevas capacidades en el kernel de Linux, que han sido aprovechadas por otros sistemas de contenedores.

Aunque en este curso nos vamos a centrar en contenedores Docker, siempre podéis obtener más información de contenedores Linux como LXC, LXI y LXCFS en <https://linuxcontainers.org/>

Además, aquí os presento un ejemplo práctico desarrollado por José Castillo donde usa LXI en sus clases: <https://www.youtube.com/watch?v=ynglk64Hecg>

### 4.3 ¿Cómo funcionan los contenedores modernos en Linux?

Los sistemas más populares de contenedores sobre Linux, han utilizado (entre otras) dos características del kernel aparecidas en versiones relativamente recientes:

- **Linux namespaces:** permite aislar procesos de forma que vean unos recursos concretos. Los procesos que tienen un “namespace” común pueden ver recursos comunes.
  - Esto, entre otras cosas, nos permite tener procesos “diferentes” de la máquina real a los contenedores, incluso con privilegios diferentes (un proceso puede ser “root” en el contenedor, pero no tiene esos privilegios en la máquina real).
  - Para saber más:
    - [https://en.wikipedia.org/wiki/Linux\\_namespaces](https://en.wikipedia.org/wiki/Linux_namespaces)
    - <https://www.linux.com/news/understanding-and-securing-linux-namespaces/>
- **Cgroups:** permite aislar, configurar y limitar el uso de recursos(memoria, procesos, E/S, etc.). Para saber más <https://en.wikipedia.org/wiki/Cgroups>

En resumen, Linux namespaces nos facilita aislar el sistema y cgroups facilita la limitación/configuración de la disponibilidad de recursos de cada contenedor.

### 4.4 ¿Puedo poner en marcha un contenedor Linux “A mano”?

Sí, es posible. Tal como nos muestra la conocida autora de materiales en formato comic **Julia Evans**, aquí un ejemplo de script donde se muestra como crear un contenedor de Linux “a mano”.

<https://gist.github.com/jvns/ea2e4d572b4e2285148b8e87f70eed73>

Aprovecho para recomendar su web <https://wizardzines.com/>, donde parte de su trabajo es Creative Commons, y en concreto su “WizardZine” sobre contenedores <https://wizardzines.com/zines/containers/>

## 4.5 Los contenedores Linux ¿Pueden funcionar en sistemas como Windows o MacOS?


En principio, es posible ejecutar contenedores Linux en sistemas diferentes a Linux, aunque es posible que el rendimiento no sea el mismo. Algunos sistemas, para poder utilizar contenedores Linux, necesitan que un hipervisor virtualice un sistema Linux completo y que desde ahí se lancen los contenedores Linux.

La instalación en sistemas Windows (distinguiendo entre escritorio y servidor) se detalla en <https://docs.docker.com/desktop/install/windows-install/>

- En sistemas Windows de escritorio, se puede instalar “Docker Desktop” que por debajo utiliza la virtualización de WSL2 (Windows Subsystem for Linux 2).
- En sistemas Windows servidor, utilizan Hyper-V e incluso pueden lanzar “Windows containers”.

En sistemas MacOS la forma de instalación de “Docker Desktop” se detalla en <https://docs.docker.com/desktop/install/mac-install/>

Actualmente, dado el crecimiento de Docker, existen otras optimizaciones que comentaremos más adelante. Aun así, esta estrategia sigue siendo posible utilizarla para virtualizar contenedores Linux (LXC, LXD, Docker, etc.) en otros sistemas.

 **Importante:** Estos casos pueden ser útiles en algún contexto (pruebas, aprendizaje, desarrollo para otra plataforma), pero se pierden ventajas relativas al rendimiento.

## 5. CONTENEDORES DOCKER

### 5.1 ¿Qué es Docker?

Docker es un sistema de contenedores Linux que utiliza las características del núcleo de Linux para permitir el desarrollo y despliegue de aplicaciones.

Su web oficial <https://www.docker.com/> y su entrada en la wikipedia donde se da información detallada del proyecto [https://es.wikipedia.org/wiki/Docker\\_\(software\)](https://es.wikipedia.org/wiki/Docker_(software))

Docker es un proyecto de código abierto. Generalmente, dispone de varias versiones:

- **Docker CE (Community Edition):** el motor de Docker, de código abierto.
- **Docker EE (Enterprise Edition):** lo mismo que la versión CE, solo que además incluye certificación de funcionamiento en algunos sistemas concretos y soporte con la empresa Docker Inc.

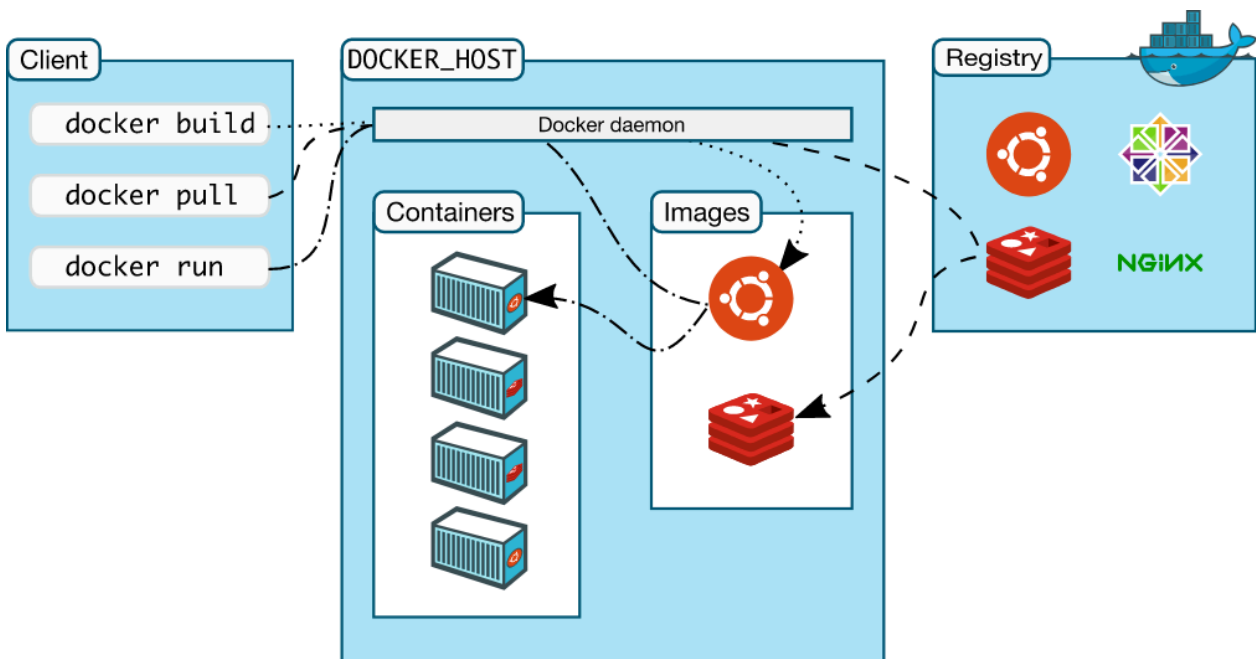
El sistema de contenedores de Docker es integrable con otros servicios populares en la nube, tales como Google Cloud, Amazon AWS, Microsoft Azure, Digital Ocean y OVH, entre otros.

- AWS: <https://aws.amazon.com/es/getting-started/hands-on/deploy-docker-containers/>
- Azure: <https://docs.microsoft.com/es-es/learn/modules/run-docker-with-azure-container-instances/>
- Google Cloud: <https://cloud.google.com/container-optimized-os?hl=es-419>



## 5.2 La arquitectura de Docker

En esta imagen podemos ver como funciona la arquitectura básica de Docker.



Fuente imagen: <https://github.com/docker/docker.github.io/blob/master/engine/images/architecture.svg>

Esta arquitectura, la podemos resumir en 3 partes:

- **Cliente:** es el software encargado de comunicarse con el servidor Docker.
- **Servidor (Docker Host):** servicio Docker, donde se atienden a las peticiones de los clientes y se gestionan los contenedores e imágenes.
- **Registro (Registry):** lugar donde se almacenan imágenes Docker (públicas o privadas). Incluso, de una misma imagen, se almacenan las distintas versiones. El registro más popular y configurado por defecto en Docker es “**Docker Hub**” <https://hub.docker.com/>

## 5.3 DOCKER EN SISTEMAS WINDOWS Y MacOS

En apartados anteriores, comentamos que una de las estrategias para lanzar Docker (y otros contenedores Linux) en sistemas Windows y MacOS, era virtualizar un sistema Linux y ahí lanzar el sistema de contenedores.

Actualmente, en sistemas Windows y MacOS se plantea como alternativa el uso del software “Docker Desktop” para utilizar contenedores en estos sistemas. Docker Desktop nos instala todas las aplicaciones necesarias para correr contenedores en estos sistemas, de la forma más óptima posible.

A continuación, realizamos una pequeña comparativa a nivel de rendimiento entre la ejecución de Docker en diferentes sistemas:

- **Ejecución de Docker en Linux:**
  - Están implementados por el kernel, con velocidad cercana a la nativa.
  - En este tutorial puedes ver los pasos para instalarlo en una distribución Ubuntu o derivadas <https://docs.docker.com/engine/install/ubuntu/>
- **Ejecución de Docker en Windows:**
  - Los contenedores Linux de Docker, funcionan usando Hyper-V en sistemas Windows Server y WSL2 (Windows Subsystem for Linux 2) en Windows Home

- <https://www.docker.com/blog/docker-hearts-wsl-2/>
  - Los contenedores “Windows Server Core” (contenedores Windows), están implementados por el núcleo de Windows.
  -
- **Ejecución de Docker en MacOS:**
  - Los contenedores Linux funcionan usando Docker Desktop para MacOS o con Hyperkit.
    - <https://docs.docker.com/desktop/install/mac-install/>
    - <https://github.com/moby/hyperkit>

Más información en:

- <https://docs.docker.com/desktop/>
- <https://docs.docker.com/docker-for-windows/release-notes/>
- <https://docs.docker.com/docker-for-mac/release-notes/>

## 5.4 DOCKER CORRIENDO CONTENEDORES WINDOWS SERVER CORE Y CONTENEDORES MacOS

Aunque el uso habitual de Docker es lanzar contenedores con sistemas Linux, nuevas mejoras han permitido que puedan utilizarse contenedores que lancen otros sistemas operativos.

Docker en sistemas Windows puede lanzar contenedores que corren el sistema operativo “Windows Server Core”. Debe virtualizarse con un sistema anfitrión Windows. Más información en:

- [https://hub.docker.com/\\_/microsoft-windows-servercore](https://hub.docker.com/_/microsoft-windows-servercore)
- <https://blog.ipswitch.com/creating-your-first-windows-container-with-docker-for-windows>

Asimismo, es posible lanzar un contenedor que ejecute MacOS en un sistema Linux que tenga instalado KVM, utilizando el proyecto disponible en <https://github.com/sickcodes/Docker-OSX>

## 6. CONCLUSIÓN

En esta unidad hemos repasado conceptos básicos sobre virtualización. Tras ello, hemos introducido el concepto de contenedor y sus características, centrándonos en contenedores Linux. Comprendidos los conceptos de contenedores, hemos introducido la solución Docker, la cual instalaremos y utilizaremos en futuras unidades.

## 7. BIBLIOGRAFÍA

- [1] WizardZines “How containers work” <https://wizardzines.com/zines/containers/>
- [2] Docker Docs <https://docs.docker.com/>
- [3] Linux containers <https://linuxcontainers.org/>
- [4] OS Level virtualization [https://en.wikipedia.org/wiki/OS-level\\_virtualization](https://en.wikipedia.org/wiki/OS-level_virtualization)

## 8. LICENCIAS DE ELEMENTOS EXTERNOS UTILIZADOS

Figura 1: Imagen con licencia Apache 2.0. Fuente:

<https://github.com/docker/docker.github.io/blob/master/engine/images/architecture.svg>

Figura 2: Imagen con licencia CC BY SA. Fuente:

<https://commons.wikimedia.org/wiki/File:Docker-containerized-and-vm-transparent-bg.png>

Figura 3: Imagen con licencia CC BY SA. Fuente:

<https://www.flickr.com/photos/68359921@N08/50125348052/>

Introducción a Docker y Kubernetes

# UD 03. Principales acciones con Docker

---



Autor: Sergi García Barea

Actualizado Enero 2025

## Licencia



**Reconocimiento – NoComercial - 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 debe hacer con una licencia igual a la que regula la obra original.

## Nomenclatura

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

 **Importante**

 **Atención**

 **Interesante**

1. Introducción	3
2. ¿Gestionaremos Docker mediante interfaz gráfica?	3
3. Imágenes y contenedores	3
3.1 ¿Qué es una imagen y un contenedor?	3
3.2 ¿Dónde se almacenan imágenes, contenedores y datos?	4
4. Registro: Docker Hub	4
5. Creando y arrancando contenedores con "docker run"	5
5.1 ¿Qué hace el comando "docker run"?	5
5.2 Creando contenedores sin arrancarlos	5
5.3 Repasando caso práctico "Hello World"	5
5.3.1 Repaso parte 1: obteniendo la imagen	5
5.3.2 Repaso parte 2: el contenedor se crea y ejecuta un comando	6
6. Listar contenedores disponibles en el sistema con "docker ps"	7
7. Parando y arrancando contenedores existentes con "docker start/stop/restart"	8
8. Inspeccionando contenedores con "docker inspect"	8
9. Ejecutando comandos en un contenedor con "docker exec"	8
10. Copiando ficheros entre anfitrión y contenedores con "docker cp"	9
11. Accediendo a un proceso en ejecución con "docker attach"	9
12. Obteniendo información de los logs con "docker logs"	10
13. Renombrando contenedores con "docker rename"	10
14. Principales ejemplos básicos de uso de docker para crear y manipular contenedores	10
14.1 Ejemplo 1: creando y lanzando contenedor Ubuntu y accediendo a una terminal	10
14.2 Ejemplo 2: accediendo a terminal desde el contenedor parado con "docker start"	11
14.3 Ejemplo 3: accediendo a la terminal de un contenedor en ejecución con "docker exec"	11
14.4 Ejemplo 4: ejecutando una versión de una imagen y autoeliminando el contenedor	12
14.5 Ejemplo 5: lanzando un servidor web en background y asociando sus puertos	12
14.6 Ejemplo 6: cambiando el "index.html" y consultando logs	13
14.7 Ejemplo 7: estableciendo variables de entorno	13
15. Bibliografía	13

## UD03. PRINCIPALES ACCIONES CON DOCKER

### 1. INTRODUCCIÓN

En esta unidad explicaremos algunas de las principales acciones básicas que podemos realizar con Docker. Al finalizar la unidad ya estaremos listos para usar Docker con cierta soltura.

### 2. ¿GESTIONAREMOS DOCKER MEDIANTE INTERFAZ GRÁFICA?

Existen distintas herramientas para gestionar Docker desde una interfaz gráfica, haciendo la tarea más visual e intuitiva. Aunque estas herramientas pueden ser muy útiles, en el momento de aprender a trabajar con Docker, pueden hacer que se nos escape la comprensión de determinados mecanismos del funcionamiento de Docker. Por ese motivo, en el curso no gestionaremos Docker mediante interfaz gráfica y realizaremos todas las operaciones mediante la línea de comandos.

### 3. IMÁGENES Y CONTENEDORES

#### 3.1 ¿Qué es una imagen y un contenedor?

Antes de comenzar, es importante aclarar algunos conceptos sobre qué son imágenes y contenedores y cuáles son sus características. Algunos conceptos a tener en cuenta son:

- **Imágenes:**
  - La imagen es una plantilla de solo lectura que se utiliza para crear contenedores. A partir de una imagen pueden crearse múltiples contenedores.
  - Las imágenes, además de tener su sistema de ficheros predefinido, tienen una serie de parámetros predefinidos (comandos, de variables de entorno, etc.) con valores por defecto y que se pueden personalizar en el momento de crear el contenedor.
  - Docker permite crear nuevas imágenes basándose en imágenes anteriores. Se podría decir que una imagen puede estar formada por un conjunto de “capas” que han modificado una imagen base.
    - Al crear una nueva imagen, simplemente estamos añadiendo una capa a la imagen anterior, la que actúa como base.
- **Contenedores:**
  - Son instancias de una imagen.
  - Pueden ser arrancados, parados y ejecutados.
  - Cada contenedor Docker posee un **identificador único de 64 caracteres**, pero habitualmente se utiliza una **versión corta con los primeros 12 caracteres**.
    - Los comandos Docker habitualmente soportan ambas versiones.
  - También podemos referirnos a los contenedores con menos caracteres, siempre que su expresión regular sea única.
    - Por ejemplo, si tienes dos contenedores y el ID de un contenedor empieza por “7” y el ID del otro por “9”, simplemente escribiendo como ID “7” se referirá al primer contenedor, ya que no hay otro contenedor que empiece por “7”. Si hubiera varios contenedores que empezaran igual, habría que añadir más caracteres hasta encontrar un patrón no común.

Un símil para entender estos conceptos: una instalación de una distribución de Linux mediante un DVD. Ese DVD sería nuestra imagen y el sistema operativo instalado sería el contenedor.

Detallaremos conceptos relacionados sobre la creación de imágenes en futuras unidades.

### 3.2 ¿Dónde se almacenan imágenes, contenedores y datos?

El lugar donde se almacenan contenedores e imágenes puede variar según distribución/sistema operativo, driver de almacenamiento y versión de Docker. Normalmente, mediante el siguiente comando de Docker, podemos ver información del sistema, incluyendo el directorio de Docker.

```
docker info
```

Ese comando nos ofrece información sobre el estado de Docker. Para conocer donde se almacena la información, dos datos son importantes: directorio de Docker y driver de almacenamiento.

Driver de almacenamiento utilizado por Docker:

```
Storage Driver: overlay2
Backing Filesystem: extfs
Supports d_type: true
Native Overlay Diff: true
```

En el ejemplo se nos indica el driver de almacenamiento utilizado ("**overlay2**") y sobre qué sistema de archivos está funcionando ("**extfs**", en concreto "**ext4**").

Para saber más sobre los distintos drivers de almacenamiento de Docker, podéis consultar <https://docs.docker.com/storage/storagedriver/select-storage-driver/>

Directorio de Docker:

```
Docker Root Dir: /var/lib/docker
```

Directorio donde se almacena todo lo relacionado con Docker.

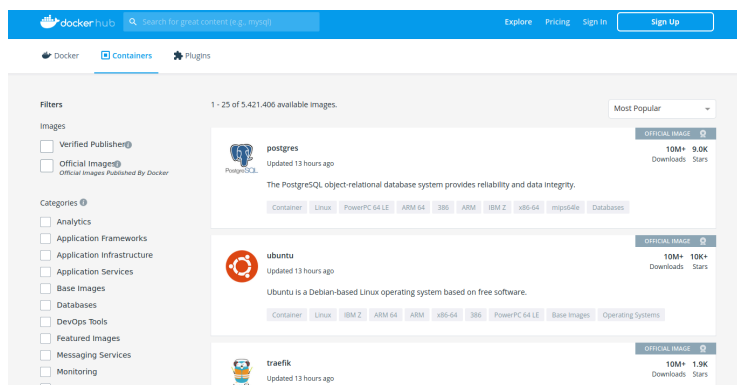
Al utilizar el driver "**overlay2**" sabemos que

- La información de las imágenes se encuentra en "**/var/lib/docker/overlay2**".
  - Recordemos que las imágenes se basan en capas, una imagen puede estar creada por un conjunto de capas.
  - El almacenamiento temporal de contenedores (es decir, pequeños cambios en los contenedores), se realiza como "versiones de las imágenes", es decir, una capa más de la imagen base.
- La configuración de los contenedores se almacena en "**/var/lib/docker/containers**".
- Para acceso a datos compartidos, persistencia, etc. existe la figura de los volúmenes, la cual detallaremos en posteriores unidades.

## 4. REGISTRO: DOCKER HUB

Docker Hub es una "plataforma de registro" de Docker. Los servicios básicos son gratuitos y nos permite registrar imágenes Docker, haciéndolas públicas o privadas.

Contiene un gran ecosistema de imágenes ya creadas, usualmente con instrucciones de instalación y uso, además de un buscador que nos permite encontrar imágenes según distintos parámetros. Enlace al buscador <https://hub.docker.com/search?q=&type=image>



Por defecto, Docker utiliza esta plataforma registro como “registro por defecto”, aunque es posible, si se requiere, elegir otro servicio de registro, e incluso montar un servicio privado de registro.

Más información sobre cómo crear un registro privado en

- <https://www.digitalocean.com/community/tutorials/how-to-set-up-a-private-docker-registry-on-ubuntu-18-04-es>

## 5. CREANDO Y ARRANCANDO CONTENEDORES CON “DOCKER RUN”

### 5.1 ¿Qué hace el comando “docker run”?

Es posiblemente el comando “docker run” más utilizado. Podríamos decir que **este comando crea un contenedor a partir de una imagen y lo arranca.**

**Importante:** un error común es creer que “*docker run*” solo arranca contenedores. Si haces varios “*docker run*”, estás creando varios contenedores, no arrancando varias veces un contenedor.

La descripción completa del comando “docker run” la podéis encontrar en <https://docs.docker.com/engine/reference/commandline/run/>

Comentaremos a lo largo de esta unidad algunas de sus opciones básicas más relevantes. Además, en futuras unidades, ampliaremos los conocimientos sobre este comando.

### 5.2 Creando contenedores sin arrancarlos

Para crear un contenedor sin arrancarlo (recordamos, “*docker run*” crea y arranca), existe el comando “*docker create*”. La descripción completa del comando “*docker create*” la podéis encontrar en <https://docs.docker.com/engine/reference/commandline/create/>

### 5.3 Repasando caso práctico “Hello World”

En anteriores unidades propusimos un sencillo caso práctico para comprobar que funcionaba Docker usando el siguiente comando:

```
docker run hello-world
```

#### 5.3.1 Repaso parte 1: obteniendo la imagen

Al ejecutar este comando por primera vez, obtenemos un resultado similar a este:

```
sergi@ubuntu:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:89b647c604b2a436fc3aa56ab1ec515c26b085ac0c15b0d105bc475be15738fb
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

La documentación en Docker Hub del contenedor que estamos lanzando la tenemos disponible en [https://hub.docker.com/\\_/hello-world](https://hub.docker.com/_/hello-world)

En primer lugar, nos fijamos en el comienzo de la información mostrada, concretamente en:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:89b647c604b2a436fc3aa56ab1ec515c26b085ac0c15b0d105bc475be15738fb
Status: Downloaded newer image for hello-world:latest
```

Ahí se nos indica que la imagen ***“hello-world:latest”*** no está localmente en nuestro sistema. Al no estar, se descarga del registro por defecto (normalmente Docker Hub) y se almacena localmente.

De hecho, si volvemos a hacer el comando “docker run hello-world”, al tener la imagen ya en el sistema, **no nos aparecerá este texto, ya que la imagen la tenemos almacenada localmente.**

Otro aspecto a destacar, es que pese a que solo hemos escrito ***“hello-world”***, nos ha descargado una imagen llamada ***“hello-world:latest”***. Esto es porque cada imagen creada tiene un nombre de versión. Si no indicamos nada o indicamos ***“latest”***, nos instala la última versión. Si queremos instalar una versión concreta de una imagen se indica de la forma ***“imagen:nombreversión”***.

### 5.3.2 Repaso parte 2: el contenedor se crea y ejecuta un comando

Una vez descargada la imagen, se crea el contenedor, se inicia y ejecuta un proceso. Este proceso lo podemos proporcionar dentro de la orden ***“docker run”*** o si, como en el caso concreto de este ejemplo, no lo hemos proporcionado, ejecutará un comando predefinido por la propia imagen.

En este caso concreto, al no haber especificado ningún comando, al iniciarse el contenedor lanza un programa por defecto llamado “hello” y nos muestra por la salida estándar información de como Docker ha generado este mensaje:



```

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

```

Si tenéis curiosidad por ver el código fuente del programa “hello”, está disponible en <https://github.com/docker-library/hello-world/blob/master/hello.c>

Este programa genera un texto que básicamente nos explica que el cliente de Docker se ha conectado con el servicio de Docker, este se ha descargado la imagen de Docker Hub (o localmente si ya estaba en nuestro sistema), se ha creado un contenedor, que por defecto tenía un comando que generaba la salida que estamos leyendo y finalmente, el servicio de Docker lo ha enviado a la terminal.

## 6. LISTAR CONTENEDORES DISPONIBLES EN EL SISTEMA CON “DOCKER PS”

Mediante el comando “**docker ps**” podemos listar los contenedores en ejecución en el sistema. Si ejecutamos el siguiente comando:

```
docker ps
```

Nos aparecerá un listado como este si no tenemos ningún contenedor en ejecución:

```

sergi@ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES

```

O si tenemos contenedores en ejecución, podemos obtener algo similar a esto:

```

sergi@ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
434d318b3771   ubuntu    "bash"    9 seconds ago   Up 7 seconds   ports   stupefied_colden

```

Lanzando el siguiente comando:

```
docker ps -a
```

Obtendremos un listado de todos los contenedores, tanto aquellos en funcionamiento como aquellos que están parados.

```

sergi@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
434d318b3771   ubuntu    "bash"    About a minute ago   Up About a minute   ports   stupefied_colden
b0619341b82c   ubuntu    "/bin/bash"   3 minutes ago   Exited (0) 2 minutes ago   intelligent_ritchie
3fb53852ce99   hello-world  "/hello"    37 minutes ago   Exited (0) 37 minutes ago   recursing_meninsky
77dc9f7f80e5   hello-world  "/hello"    41 minutes ago   Exited (0) 41 minutes ago   priceless_pare

```

La información que obtenemos de los contenedores es la siguiente:

- **CONTAINER\_ID**: identificador único del contenedor (versión 12 primeros caracteres).
- **IMAGE**: imagen utilizada para crear el contenedor.
- **COMMAND**: comando que se lanza al arrancar el contenedor.
- **CREATED**: cuando se creó el contenedor.
- **STATUS**: si el contenedor está en marcha o no (indicando cuánto lleva en marcha o cuánto hace que se paró).
- **PORTS**: redirección de puertos del contenedor (lo veremos más adelante en la unidad).
- **NAMES**: nombre del contenedor. Se puede generar como parámetro al crear el contenedor, o si no se indica nada, el propio Docker genera un nombre aleatorio.

La descripción completa del comando **“docker ps”** la podéis encontrar en <https://docs.docker.com/engine/reference/commandline/ps/>

## 7. PARANDO Y ARRANCANDO CONTENEDORES EXISTENTES CON “DOCKER START/STOP/RESTART”

Para arrancar/parar un contenedor ya creado (recordamos, **“docker run”** crea y arranca), existen los comandos **“docker start”**, **“docker stop”** y **“docker restart”**.

La forma más habitual de usar estos comandos, es usar el nombre del comando, seguido del identificador único o nombre asignado al contenedor. Por ejemplo, con identificador:

```
docker start 434d318b3771
```

o con nombre del contenedor

```
docker start stupefied_colden
```

La descripción completa de estos comandos la podéis encontrar en

- <https://docs.docker.com/engine/reference/commandline/start/>
- <https://docs.docker.com/engine/reference/commandline/stop/>
- <https://docs.docker.com/engine/reference/commandline/restart/>

## 8. INSPECCIONANDO CONTENEDORES CON “DOCKER INSPECT”

El comando **“docker inspect”** es un comando que nos proporciona diversos detalles de la configuración de un contenedor. Ofrece distintos datos, entre ellos, identificador único (versión 64 caracteres), almacenamiento, red, imagen en que se basa, etc. Su sintaxis es:

```
docker inspect IDENTIFICADOR/NOMBRE
```

La descripción completa del comando **“docker inspect”** la podéis encontrar en <https://docs.docker.com/engine/reference/commandline/inspect/>.

## 9. EJECUTANDO COMANDOS EN UN CONTENEDOR CON “DOCKER EXEC”

El comando **“docker exec”** nos permite ejecutar un comando dentro de un contenedor que esté en ese momento en ejecución. La forma sintaxis habitual para utilizar este comando es la siguiente

```
docker exec [OPCIONES] IDENTIFICADOR/NOMBRE COMANDO [ARGUMENTOS]
```

Algunos ejemplos de uso, suponiendo un contenedor en marcha, llamando “contenedor”:

```
docker exec -d contenedor touch /tmp/prueba
```

Ejemplo que se ejecuta en “background”, gracias al parámetro “-d”. Este ejemplo simplemente crea mediante el comando “touch” un fichero “prueba” en “/tmp”.

```
docker exec -it contenedor bash
```

Orden que ejecutará la “shell” bash en nuestra consola (gracias al parámetro “-it” se enlaza la entrada y salida estándar a nuestra terminal). A efectos prácticos, con esta orden accederemos a una “shell” bash dentro del contenedor.

```
docker exec -it -e VAR1=1 contenedor bash
```

Comando que establece un variable de entorno con el parámetro “-e”. Se enlaza la entrada y salida de la ejecución del comando con “-it”. A efectos prácticos, en esa “shell” estará disponible la variable de entorno “VAR1” con valor 1. Lo podemos probar con “echo \$VAR1”.

La descripción completa del comando “**docker exec**” la podéis encontrar en <https://docs.docker.com/engine/reference/commandline/exec/>.

## 10. COPIANDO FICHEROS ENTRE ANFITRIÓN Y CONTENEDORES CON “DOCKER CP”

El comando “**docker cp**” es un comando que nos permite copiar ficheros y directorios del anfitrión a un contenedor o viceversa. No se permite actualmente la copia de fichero entre contenedores.

Algunos ejemplos de uso:

```
docker cp idcontainer:/tmp/prueba ./
```

Copia el fichero “**/tmp/prueba**” del contenedor con identificador o nombre “idcontainer” al directorio actual de la máquina que ejerce como anfitrión.

```
docker cp ./miFichero idcontainer:/tmp
```

Copia el fichero “**miFichero**” del directorio actual al directorio “/tmp” del contenedor.

La descripción completa del comando “**docker cp**” la podéis encontrar en <https://docs.docker.com/engine/reference/commandline/cp/>.

## 11. ACCEDIENDO A UN PROCESO EN EJECUCIÓN CON “DOCKER ATTACH”

En algunos casos, deseamos enlazar la entrada o salida estándar de nuestra terminal a un contenedor que está ejecutando un proceso en segundo plano, de forma similar a la siguiente

```
docker attach [OPCIONES] IDENTIFICADOR/NOMBRE
```

Para probarlo utilizaremos el siguiente ejemplo:

El ejemplo consiste en crear un contenedor que lanza un proceso que genera texto (imprimiendo la fecha) por la salida estándar de forma indefinida. El comando llama a “sh” con el parámetro -c (que indica que la siguiente cadena es algo a procesar por la “shell” sh), seguido de una cadena con un “shell script”. Aquí vemos el comando, que podríamos lanzar en cualquier terminal.

```
sh -c "while true; do $(echo date); sleep 1; done"
```

Aplicamos este comando a nuestro ejemplo creando un contenedor:

```
docker run -d --name=muchotexto busybox sh -c "while true; do $(echo date); sleep 1; done"
```

**! Atención:** Los parámetros de este “docker run” son explicados más adelante en el documento.

Con ese contenedor en marcha, ya podemos probar “**docker attach**”. Podremos enlazar la entrada y salida del proceso en ejecución a nuestra terminal y observar el texto generado usando:

```
docker attach muchotexto
```

La descripción completa del comando “**docker attach**” la podéis encontrar en <https://docs.docker.com/engine/reference/commandline/attach/>.

## 12. OBTENIENDO INFORMACIÓN DE LOS LOGS CON “DOCKER LOGS”

Podemos consultar la información generada con el comando “**docker logs**”

```
docker logs [OPCIONES] IDENTIFICADOR/NOMBRE
```

Este uso es similar a “docker attach”, solo que tiene opciones específicas para tratar la información obtenida como un log. Partiendo del mismo ejemplo usado en “**docker attach**”.

```
docker run -d --name=muchotexto busybox sh -c "while true; do $(echo date); sleep 1; done"
```

Un ejemplo de uso para obtener logs podría ser

```
docker logs -f --until=2s muchotexto
```

Con este ejemplo, te mostraría los logs generados (realmente la salida estándar y de error), incluyendo aquellos que se fueran generando, parando a los dos segundos.

La descripción completa del comando “**docker logs**” la podéis encontrar en <https://docs.docker.com/engine/reference/commandline/logs/>.

## 13. RENOMBRANDO CONTENEDORES CON “DOCKER RENAME”

El comando “**docker rename**” nos permite cambiar el nombre asociado a un contenedor.

```
docker rename contenedor1 contenedor2
```

Cambia el nombre de “**contenedor1**” a “**contenedor2**”.

La descripción completa del comando “**docker rename**” la podéis encontrar en <https://docs.docker.com/engine/reference/commandline/rename/>.

## 14. PRINCIPALES EJEMPLOS BÁSICOS DE USO DE DOCKER PARA CREAR Y MANIPULAR CONTENEDORES

### 14.1 Ejemplo 1: creando y lanzando contenedor Ubuntu y accediendo a una terminal

Anteriormente, hemos indicado que el comando “**docker run**” es de gran importancia en el uso de Docker y que este nos permite crear contenedores a partir de una imagen y arrancarlos.

La estructura principal del comando es la siguiente

```
docker run [PARAMETROS] IMAGEN [COMANDO AL ARRANCAR] [ARGUMENTOS]
```

A continuación mostramos algunos ejemplos de “**docker run**”.

Utilizando el comando

```
docker run -it --name=nuestroUbuntu1 ubuntu /bin/bash
```

Estamos creando un nuevo contenedor a partir de la imagen “**ubuntu**”. Al crear este contenedor hemos especificado los siguientes parámetros:

- **Parámetro “-i”**: indica que el proceso lanzado en el contenedor docker estará en modo interactivo, es decir, enlaza la entrada estándar cuando se asigna un proceso a una terminal.
- **Parámetro “-t”**: asigna al proceso lanzado al arrancar el contenedor una pseudo terminal, facilitando el acceso al mismo desde nuestra terminal.
- **Parámetro “--name”**: nos permite establecer un nombre a nuestro contenedor. Si no indicamos este parámetro, nos creará un nombre aleatorio.

Por último, el comando ejecutado al lanzarse el contenedor es **`"/bin/bash"`**. Este, combinado con los parámetros **`"-it"`** (que entraban en modo interactivo y asociaban una terminal), nos hace que justo después de lanzar el comando, estemos en una **`"shell"`** dentro del contenedor creado. Al finalizar dicha **`"shell"`** (con **`"exit"`**, **`"control+c"`**, etc.) el contenedor se parará.

Los cambios que hayamos hecho con la **`"shell"`**, como por ejemplo, crear un directorio, se almacenarán como una imagen temporal (veremos en profundidad las imágenes en otra unidad) y a efectos prácticos, los cambios serán permanentes al arrancar de nuevo este contenedor.

#### 14.2 Ejemplo 2: accediendo a terminal desde el contenedor parado con **`"docker start"`**

El anterior ejemplo nos permitía acceder, crear un contenedor y acceder de forma interactiva a dicha **`"shell"`**, pero al salir de la shell, simplemente se paraba el contenedor.

Entonces, ¿Cómo podríamos volver a ese contenedor y a dicha **`"shell"`**? Usaremos **`"docker start"`**.

Este comando nos permitirá arrancar el contenedor parado. Al arrancar no especificaremos un comando a lanzar, ya que se lanza el comando que hayamos especificado (o por defecto de la imagen si no hemos especificado nada) al hacer **`"docker run"`** o **`"docker create"`**.

El comando **`"docker start"`** sigue la siguiente estructura:

```
docker start [PARAMETROS] IDENTIFICADOR/NOMBRE
```

Podemos obtener identificador único o nombre usando el comando:

```
docker ps -a
```

Tras ello, lanzamos el contenedor de la siguiente forma:

```
docker start -ai IDENTIFICADOR
```

Los parámetros especificados a **`"docker start"`** son los siguientes:

- **Parámetro `"-a"`**: al arrancar el contenedor, enlaza la salida estándar y de error del contenedor a nuestra terminal.
- **Parámetro `"-i"`**: al arrancar el contenedor, lo hace en modo interactivo, es decir, enlazando la entrada estándar del contenedor a nuestra terminal.

#### 14.3 Ejemplo 3: accediendo a la terminal de un contenedor en ejecución con **`"docker exec"`**

En el ejemplo anterior vimos cómo acceder a un contenedor que estaba parado y volver a su terminal interactiva. Pero, ¿qué pasa si el contenedor ya está en ejecución y queremos conectarnos a su terminal (shell) sin reiniciarlo? Para ello, utilizaremos el comando **`"docker exec"`**.

Este comando nos permite ejecutar una orden dentro de un contenedor en ejecución sin detenerlo ni afectarlo. En este caso, lo usaremos para abrir una terminal dentro del contenedor.

La estructura básica del comando es:

```
docker exec [PARAMETROS] IDENTIFICADOR/NOMBRE COMANDO
```

Podemos obtener el identificador único o el nombre del contenedor con el siguiente comando:

```
docker ps
```

Una vez identificado el contenedor, podemos acceder a su terminal interactiva con el siguiente

comando:

```
docker exec -it IDENTIFICADOR bash
```

Los parámetros utilizados en “docker exec” son los siguientes:

**-i:** Mantiene la entrada estándar del contenedor abierta para poder interactuar con ella.

**-t:** Asigna una terminal virtual para que podamos trabajar cómodamente con la shell del contenedor.

En algunos casos, la imagen utilizada en el contenedor no tiene bash instalado, por lo que podemos probar con sh en su lugar:

```
docker exec -it IDENTIFICADOR sh
```

Con esto, podremos acceder a la terminal de un contenedor en ejecución sin necesidad de detenerlo o reiniciarlo.

#### 14.4 Ejemplo 4: ejecutando una versión de una imagen y autoeliminando el contenedor

Lanzando el siguiente comando

```
docker run -it --rm ubuntu:24.04 /bin/bash
```

Estamos creando un contenedor con la versión de la imagen “**ubuntu**” etiquetada como “**14.04**” en Docker Hub y arrancándolo de forma similar al ejemplo anterior.

Los parámetros nuevos incluidos en esta orden son:

- **Parámetro “--rm”:** este parámetro hará que nada más el contenedor se pare, se borre el contenedor del sistema.

#### 14.5 Ejemplo 5: lanzando un servidor web en background y asociando sus puertos

Lanzando el siguiente comando

```
docker run -d -p 1200:80 nginx
```

Estamos creando un contenedor con la versión de la imagen “**nginx:latest**”, la cual contiene un servidor web Nginx en funcionamiento en el puerto 80 del contenedor y al que podremos acceder en nuestra máquina como “**localhost:1200**”.

Los parámetros nuevos incluidos en esta orden son:

- **Parámetro “-d”:** parámetro “**deattached**”, que indica que lanza el contenedor en segundo plano. Al lanzarlo con esta opción, no se nos muestra ninguna información de la entrada/salida del contenedor. La única información que se nos muestra es el ID del contenedor lanzado.
- **Parámetro “-p”:** siguiendo el estilo “**pAnf:pCont**” nos indica que en el puerto de la máquina anfitrión “**pAnf**” está enlazado con el puerto interno del contenedor “**pCont**”.
  - Si solo se indica un puerto, algo del estilo “**-p 80**”, el sistema tomará dicho puerto como el puerto interno del contenedor y asociará un puerto aleatorio libre de la máquina anfitrión. Podremos consultar los puertos expuestos de un contenedor mediante el comando “**docker ps**” o el específico para esta tarea “**docker port**”.

**! Atención:** el mapeo de puertos solo puede realizarse en el momento de crear el contenedor. No se puede modificar el mapeo de puertos con el contenedor ya creado.

Para saber más sobre la imagen que hemos utilizado, en este caso “nginx” podemos consultar su página en Docker Hub [https://hub.docker.com/\\_/nginx](https://hub.docker.com/_/nginx)

### 14.6 Ejemplo 6: cambiando el “index.html” y consultando logs

Observando la documentación que nos ofrece sobre la imagen [https://hub.docker.com/\\_/nginx](https://hub.docker.com/_/nginx), observamos que la ruta donde se encuentra la página que sirve “nginx” se encuentra en “/usr/share/nginx/html”. Accediendo a esa ruta, podríamos modificar el “index.html” que se ve cuando nos conectamos al puerto 1200 en nuestra máquina.

Con las herramientas que tenemos, tenemos varias acciones para modificarlo

- Accede con una “shell” con “**docker exec**”, instalar un editor de texto (por ejemplo, con “**apt update; apt install nano**”) y editar el fichero desde la consola.
- Copiar “index.html” desde nuestra máquina anfitriona con “**docker cp**”.

También podemos acceder a los logs que nos va generando durante su ejecución. Si por ejemplo queremos acceder a las últimas 10 líneas de logs generados, podemos utilizar

```
docker logs -n 10 NOMBRE_CONTENEDOR
```

### 14.7 Ejemplo 7: estableciendo variables de entorno

Vamos a ver un sencillo ejemplo donde vamos a establecer una variable de entorno e imprimir su valor en pantalla. Ejecutamos el siguiente comando

```
docker run -it -e MENSAJE=HOLA ubuntu bash
```

Con este ejemplo, al crear el contenedor hemos establecido la variable de entorno “MENSAJE” y lanzado una terminal. Podemos probar que la variable se ha establecido correctamente usando:

```
echo $MENSAJE
```

Los parámetros nuevos incluidos en esta orden son:

- **Parámetro “-e”**: simplemente nos permite establecer una o varias “variables de entorno”.

Este simple ejemplo nos indica cómo establecer variables de entorno al construir un contenedor.

También, en el momento de la creación de imágenes, se puede establecer variables de entorno con valores por defecto de cada imagen. Estos valores se mantendrán, salvo que sean sobrescritos con el parámetro “-e”.

## 15. BIBLIOGRAFÍA

[1] Docker Docs <https://docs.docker.com/>



Completa - Autor: Sergi García Barea

## Docker run

---

```
docker run -it --name=cont1 ubuntu /bin/bash
```

- Crea un contenedor con la imagen “ubuntu” (al no especificar, toma versión “latest”), le establece un nombre “cont1” y lanza en modo interactivo una shell “bash”.

```
docker run -d -p 1200:80 nginx
```

- Crea un contenedor con la versión “latest” de la imagen “nginx” y lo lanza en “background”, exponiendo el puerto 80 del contenedor en el puerto 1200 de la máquina anfitrión.

```
docker run -it -e MENSAJE=HOLA ubuntu:14.04 bash
```

- Crea un contenedor con la imagen “ubuntu”, versión “14.04” y establece la variable de entorno “MENSAJE”.

## Docker ps

---

```
docker ps
```

- Muestra información de los contenedores en ejecución.

```
docker ps -a
```

- Muestra información de todos los contenedores, tanto parados como en ejecución.

## Docker start/stop/restart

---

```
docker start micontenedor
```

- Arranca el contenedor con nombre “mi contenedor”.

```
docker start -ai micontenedor
```

- Arranca el contenedor con nombre “mi contenedor”, enlazando el comando ejecutado al arranque a la entrada y salida estándar de la terminal del anfitrión.

## Docker Exec

---

```
docker exec -it idcont bash
```

- Lanza en el contenedor “idcont” (que debe estar arrancado) el comando “bash”, enlazando la ejecución de forma interactiva a la entrada y salida estándar del anfitrión en la terminal.

```
docker exec -it -e FICHERO=prueba cont bash
```

- Lanza en el contenedor “cont” (que debe estar arrancado) el comando “bash”, estableciendo la variable de entorno “FICHERO” y enlazando la ejecución de forma interactiva a la entrada y salida estándar del anfitrión.





Completa - Autor: Sergi García Barea

```
docker exec -d cont touch /tmp/prueba
```

- Lanza en el contenedor “cont” (que debe estar arrancado) el comando “touch /tmp/prueba”. Este comando se ejecuta en segundo plano, generando el fichero “/tmp/prueba”.



## Docker attach

---

```
docker attach idcontainer
```

- Enlaza nuestra terminal la entrada/salida de nuestra al proceso en segundo plano del contenedor “idcontainer”.



## Docker logs

---

```
docker logs -n 10 idcontainer
```

- Muestra las 10 últimas líneas de la salida estandar producida por el proceso en ejecución en el contenedor.



## Docker cp

---

```
docker cp idcontainer:/tmp/prueba ./
```

- Copia el fichero “/tmp/prueba” del contenedor “idcontainer” al directorio actual del anfitrión.

```
docker cp ./miFichero idcontainer:/tmp
```

- Copia el fichero “miFichero” del directorio actual del anfitrión a la carpeta “/tmp” del contenedor.



## Gestión de imágenes

---

```
docker images
```

- Información de imágenes locales disponibles.

```
docker search ubuntu
```

- Busca la imagen “ubuntu” en el repositorio remoto (por defecto Docker Hub).

```
docker pull alpine
```

- Descarga localmente imagen “alpine”.

```
docker history alpine
```

- Muestra la historia de creación de la imagen “alpine”.

```
docker rmi ubuntu:14.04
```

- Elimina localmente la imagen “ubuntu” con tag “14.04”.

```
docker rmi $(docker images -q)
```

- Borra toda imagen local que no esté siendo usada por un contenedor.



Completa - Autor: Sergi García Barea

```
docker rm IDCONTENEDOR
```

- Borra un contenedor con IDCONTENEDOR.

```
docker stop $(docker ps -a -q)
```

- Para todos los contenedores del sistema.

```
docker rm $(docker ps -a -q)
```

- Borra todos los contenedores parados del sistema.

```
docker system prune -a
```

- Borra todas las imágenes y contenedores parados del sistema.



## Creación de imágenes a partir de contenedores

---

```
docker commit -m "comentario" IDCONTENEDOR usuario/imagen:version
```

- Hace commit de un contenedor existente a una imagen local.

```
docker save -o copiaSeguridad.tar imagenA
```

- Guarda una copia de seguridad de una imagen en fichero ".tar".

```
docker load -i copiaSeguridad.tar
```

- Restaura una copia de seguridad de una imagen en fichero ".tar".



## Docker Hub

---

```
docker login
```

- Permite introducir credenciales del registro (por defecto "Docker Hub").

```
docker push usuario/imagen:version
```

- Permite subir al repositorio una imagen mediante "push".



## Ejemplo de Dockerfile

---

```
FROM alpine
LABEL maintainer="email@gmail.com"
#Actualizamos e instalamos paquetes con APK para Alpine
RUN apk update && apk add apache2 php php-apache2 openrc tar
#Copiamos script para lanzar Apache 2
ADD ./start.sh /start.sh
#Descargamos un ejemplo de <?php phpinfo(); ?> por enseñar como bajar algo de Internet
#Podría haber sido simplemente
#RUN echo "<?php phpinfo(); ?>" > /var/www/Localhost/htdocs/index.php
ADD https://gist.githubusercontent.com/SyntaxC4/5648247/raw/94277156638f9c309f2e36e19bff378ba7364907/info.php
/var/www/localhost/htdocs/index.php
# Si quisiéramos algo como Wordpress haríamos
```



Completa - Autor: Sergi García Barea

```
#ADD http://wordpress.org/latest.tar.gz /var/www/localhost/htdocs/wordpress.tar.gz
#RUN tar xvfz /var/www/localhost/htdocs/wordpress.tar.gz && rm -rf /var/www/localhost/htdocs/wordpress.tar.gz
# Usamos usuario y grupo www-data. El grupo lo crea Apache, pero si quisiéramos crear grupo
# Grupo www-data RUN set -x && addgroup -g 82 -S www-data
RUN adduser -u 82 -D -S -G www-data www-data
# Hacemos todos los ficheros de /var/www propiedad de www-data
# Y damos permisos a esos ficheros y a start.sh
RUN chown -R www-data:www-data /var/www/ && chmod -R 775 /var/www/ && chmod 755 /start.sh
#Indicamos puerto a exponer (para otros contenedores) 80
EXPOSE 80
#Comando lanzado por defecto al instalar el contenedor
CMD /start.sh
```

- Ejemplo de fichero “Dockerfile”.



## Gestión de redes

```
docker network create redtest
```

- Creamos la red “redtest”

```
docker network ls
```

- Nos permite ver el listado de redes existentes.

```
docker network rm redtest
```

- Borramos la red “redtest”.

```
docker run -it --network redtest ubuntu /bin/bash
```

- Conectamos el contenedor que creamos a la red “redtest”.

```
docker network connect IDRED IDCONTENEDOR
```

- Conectamos un contenedor a una red.

```
docker network disconnect IDRED IDCONTENEDOR
```

- Desconectamos un contenedor de una red



## Volúmenes

```
docker run -d -it --name appcontainer -v /home/sergi/target:/app nginx:latest
```

- Creamos un contenedor y asignamos un volumen con “binding mount”.

```
docker run -d -it --name appcontainer -v micontenedor:/app nginx:latest
```

- Creamos un contenedor y asignamos un volumen Docker llamado “micontenedor”.

```
docker volume create/ls/rm mivolumen
```

- Permite crear, listar o eliminar volúmenes Docker.

```
docker run -d -it --tmpfs /app nginx
```



Completa - Autor: Sergi García Barea

- Permite crear un contenedor y asociar un volumen “tmpfs”.

```
docker volume rm $(docker volume ls -qf dangling=true)
```

- Borra todos los volúmenes huérfanos (sin contenedor asociado).

```
docker run --rm --volumes-from contenedor1 -v /home/sergi/backup:/backup ubuntu bash -c "cd /datos && tar cvf /backup/copiaseguridad.tar ."
```

- Permite realizar una copia de seguridad de un volumen asociado a “contenedor1” y que se monta en “/datos”. Dicha copia finalmente acabará en “/home/sergi/backup” de la máquina anfitrión.

```
docker volume rm $(docker volume ls -q)
```

- Permite eliminar todos los lúmenes de tu máquina.



## Ejemplo básico de fichero “docker-compose.yml”

```
version: "3.9"
services:
  db:
    image: mariadb:10.11.2
    volumes:
      - db_data:/var/lib/mysql
    environment:
      MARIADB_ROOT_PASSWORD: somewordpress
      MARIADB_DATABASE: wordpress
      MARIADB_USER: wordpress
      MARIADB_PASSWORD: wordpress
  wordpress:
    image: wordpress:latest
    ports:
      - "8000:80"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
  db_data:
```



## Principales comandos de “Docker Compose”

```
docker compose up -d
```

- Inicia el sistema definido en “**docker-compose.yml**” en segundo plano. Genera y descarga imágenes requeridas.

```
docker compose down
```

- Detiene y elimina los contenedores según la configuración de “**docker-compose.yml**”.

```
docker compose build/pull
```

- Construye/descarga las imágenes de contenedores según la configuración de “**docker-compose.yml**”.



Completa - Autor: Sergi García Barea

```
docker compose ps
```

- Muestra información de los contenedores según la configuración de “**docker-compose.yml**”.

```
docker compose up -d --scale web=3
```

- Similar a “**docker compose up -d**” solo que además, el servicio definido como “web” en el fichero “**docker-compose.yml**” lo escala creando 3 copias y realizando balanceo automático si se realiza una petición al host llamado como el servicio “**web**”.



## Principales comandos de “Kubernetes”

---

```
kubectl apply -f “fichero.yaml”
```

- Aplica en Kubernetes la configuración especificada en “fichero.yaml”.

```
kubectl create deployment midespliegue --image=sergarb1/flaskparakubernetes --port=5000
```

- Crea un despliegue basado en una imagen dada y en el puerto 5000.

```
kubectl expose deployment midespliegue --type=LoadBalancer --name=midespliegue-http
```

- Crea un servicio de tipo “LoadBalancer” exponiendo “midespliegue”.

```
kubectl get pods; kubectl get services; kubectl get deployments
```

- Muestra información de pods, servicios o despliegues.

```
kubectl scale deployment midespliegue --replicas=3
```

- Escala horizontalmente un despliegue a 3 réplicas.

```
kubectl autoscale deployment midespliegue --min=5 --max=10
```

- Configura autoescalado horizontal, aceptando entre 5 y 10 réplicas.

```
kubectl delete pod/deployment/service/autoscale nombre
```

- Permite eliminar un pod, despliegue, servicio o autoescalado.



## Principales comandos de “MniKube”

---

```
minikube start
```

- Inicia la máquina virtual que contiene MiniKube y pone el cluster Kubernetes en marcha

```
minikube service miservicio
```

- Nos permite acceder a un servicio dentro de MiniKube desde la máquina local.

```
minikube tunnel
```

- Mientras esté en ejecución, expone un servicio dentro de MiniKube a la máquina local



Completa - Autor: Sergi García Barea

## Ejemplo de fichero YAML despliegue/servicio/persistencia con Kubernetes

```
#Definimos la información del servicio
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    #El servicio se expone en el puerto 80
    - port: 80
  selector:
    app: wordpress
    tier: frontend
    #Aplicamos balanceo de carga para facilitar su escalado horizontal
  type: LoadBalancer
---
#Definimos un volumen persistente
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  labels:
    app: wordpress
spec:
  #Indica que solo puede ser montado para lectura/escritura por un nodo. Para el resto lectura.
  #En este caso, se usa para modificar un fichero de configuración.
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
#definimos el despliegue
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
    spec:
```

Completa - Autor: Sergi García Barea

```
#Imagen
containers:
- image: wordpress:4.8-apache
  name: wordpress
#Indicamos variables de entorno
env:
- name: WORDPRESS_DB_HOST
  value: wordpress-mysql
- name: WORDPRESS_DB_PASSWORD
  value: CEFIREdocker
ports:
- containerPort: 80
  name: wordpress
volumeMounts:
- name: wordpress-persistent-storage
  mountPath: /var/www/html
volumes:
- name: wordpress-persistent-storage
  persistentVolumeClaim:
    claimName: wp-pv-claim
```

Unidad 06 - Autor: Sergi García Barea

## Ejemplo básico de fichero "docker-compose.yml"

```
version: "3.9"
services:
  db:
    image: mariadb:10.11.2
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MARIADB_ROOT_PASSWORD: somewordpress
      MARIADB_DATABASE: wordpress
      MARIADB_USER: wordpress
      MARIADB_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress

volumes:
  db_data:
```

## Principales comandos de "Docker Compose"

`docker compose up -d`

- Inicia el sistema definido en "**docker-compose.yml**" en segundo plano. Genera y descarga imágenes requeridas.

`docker compose down`

- Detiene y elimina los contenedores según la configuración de "**docker-compose.yml**".

`docker compose build/pull`

- Construye/descarga las imágenes de contenedores según la configuración de "**docker-compose.yml**".

`docker compose ps`

- Muestra información de los contenedores según la configuración de "**docker-compose.yml**".

`docker compose up -d --scale web=3`

- Similar a "**docker compose up -d**" solo que además, el servicio definido como "web" en el fichero "**docker-compose.yml**" lo escala creando 3 copias y realizando balanceo automático si se realiza una petición al host llamado como el servicio "**web**".



Introducción a Docker

# UD 06. Docker Compose

---



Autor: Sergi García Barea

Actualizado Febrero 2025

## Licencia



**Reconocimiento – NoComercial - 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 debe hacer con una licencia igual a la que regula la obra original.

## Nomenclatura

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

**Importante**

**Atención**

**Interesante**

<b>Introducción</b>	<b>3</b>
<b>Docker Compose</b>	<b>3</b>
<b>¿Qué es Docker Compose?</b>	<b>3</b>
<b>Instalando Docker Compose</b>	<b>3</b>
<b>Formato YAML y Fichero "docker-compose.yml"</b>	<b>4</b>
<b>¿Qué es YAML?</b>	<b>4</b>
<b>¿Qué es el fichero "docker-compose.yml"?</b>	<b>4</b>
<b>¿Qué información podemos especificar en nuestro fichero "docker-compose.yml"?</b>	<b>5</b>
<b>Entendiendo un fichero "docker-compose.yml" con ejemplo Wordpress</b>	<b>6</b>
<b>Palabra clave "version"</b>	<b>6</b>
<b>Palabra clave "services" y contenedor "db"</b>	<b>7</b>
<b>Palabra clave "services" y contenedor "wordpress"</b>	<b>8</b>
<b>Palabra clave "volumes" fuera de las plantillas de contenedores</b>	<b>9</b>
<b>Lanzando nuestro ejemplo Wordpress "docker-compose.yml"</b>	<b>9</b>
<b>Comandos de "Docker Compose"</b>	<b>9</b>
<b>Comando "up"</b>	<b>9</b>
<b>Comando "down"</b>	<b>9</b>
<b>Comando "build"</b>	<b>10</b>
<b>Comando "pause"/"unpause"</b>	<b>10</b>
<b>Comando "start"/"stop"</b>	<b>10</b>
<b>Comando "ps"</b>	<b>10</b>
<b>Comando "exec"</b>	<b>11</b>
<b>Bibliografía</b>	<b>11</b>

## UD06. DOCKER COMPOSE

### 1. INTRODUCCIÓN

En esta unidad hablaremos de “**Docker Compose**”, una utilidad para definir y poner en marcha múltiples contenedores Docker, mediante un fichero de configuración en formato **YAML**.

### 2. DOCKER COMPOSE

#### 2.1 ¿QUÉ ES DOCKER COMPOSE?

A lo largo de anteriores unidades hemos visto aplicaciones alojadas en contenedores Docker que para ponerse en marcha no utilizaban un único contenedor, sino que requerían el uso de varios contenedores. El proceso de poner en marcha estas aplicaciones era tedioso, ya que usualmente debíamos levantar “a mano” esos contenedores, respetando el orden de levantado, etc.

“**Docker Compose**” es una aplicación para simplificar la tarea de lanzar múltiples contenedores con una configuración específica y enlazarlos entre sí. Además, “**Docker Compose**” actúa como un orquestador simple, permitiendo cierto escalado local (aunque con bastantes limitaciones en comparación con otros orquestadores, como Kubernetes).

Para definir el comportamiento que realizará Docker Compose, utilizaremos un fichero de configuración escrito en formato **YAML**.

Para más información <https://docs.docker.com/compose/>

#### 2.2 Instalando Docker Compose

En versiones antiguas de “**Docker Compose**”, *este se instalaba con un ejecutable aparte (por este motivo en muchos manuales verás el comando de “Docker Compose” escrito como “docker-compose”*. Aquí una captura antigua de este curso donde se observa este nombre:

```
sergi@ubuntu:~$ docker-compose --version
docker-compose version 1.29.0, build 07737305
```

Sin embargo, en las versiones actuales, dado su importancia, “Docker Compose” se ha integrado como un plugin de Docker y si hemos realizado la instalación de Docker como se propone en este curso y en la documentación oficial, ya lo tendremos.

En el momento de la redacción de este documento, la última versión es la **2.32.4**. Podemos ver una captura con la forma de llamarlo actual, como un comando más de Docker “**docker compose**”.

```
sergi@ubuntu:~$ docker compose version
Docker Compose version v2.32.4
```

### 3. FORMATO YAML Y FICHERO “DOCKER-COMPOSE.YML”

#### 3.1 ¿Qué es YAML?

YAML es el acrónimo recursivo de “**YAML Ain't Markup Language**”, que en castellano significa, “**YAML no es un lenguaje de marcas**”. Más información en <https://es.wikipedia.org/wiki/YAML>

**YAML** es a efectos prácticos una forma de definir información utilizando formato texto. Funciona de forma similar a **XML** o **JSON**. Si tenéis experiencia con **XML** o **JSON**, en este blog plantean y explican un ejemplo representando la misma información en los 3 formatos:

<https://journey2theccie.wordpress.com/2020/02/28/devnet-associate-1-1-compare-data-formats-xml-json-yaml/>

El uso de **YAML** dentro de “**Docker Compose**” es sencillo y fácilmente entendible con cada uno de los ejemplos. Podéis hacer un repaso previo a los principales elementos de **YAML** utilizados en ficheros “**Docker Compose**” revisando los ejemplos de **YAML** en la Wikipedia, disponibles en: <https://es.wikipedia.org/wiki/YAML#Ejemplos>.

🗨 **Interesante:** Utilidades como <https://onlineyamltools.com/convert-yaml-to-json> nos permiten convertir **YAML** a **XML** o **JSON**.

### 3.2 ¿Qué es el fichero “docker-compose.yml”?

El fichero “**docker-compose.yml**” es un fichero en formato **YAML** que definirá el comportamiento de cada configuración de “**Docker Compose**”. Lo habitual, es tener ese fichero en la raíz de nuestro proyecto. En este el próximo punto veremos que es el formato **YAML** y posteriormente presentaremos un ejemplo básico y comprensible de un fichero “**docker-compose.yml**”.

### 3.3 ¿Qué información podemos especificar en nuestro fichero “docker-compose.yml”?

La especificación completa para la versión 3 del fichero “**docker-compose.yml**” está definida en <https://docs.docker.com/compose/compose-file/compose-file-v3/>

A continuación, vamos a definir los más importantes:

- **version:** permite indicar la versión de la especificación del fichero “**docker-compose.yml**” no es necesario desde la versión 1.27.0 de “**Docker Compose**”.
- **services:** array asociativo con las diferentes plantillas de cada contenedor.
- **build:** usado en las plantillas de contenedores. Sirve para indicar si debemos construir la imagen a partir de un “**Dockerfile**”. Aunque tiene varias formas de uso detalladas en <https://docs.docker.com/compose/compose-file/compose-file-v3/#build>, la forma más habitual de usarla es indicar en qué directorio está nuestro “**Dockerfile**” de la forma “**build: ./directorio**”.
- **command:** sobrescribe el comando por defecto a la imagen. Se usa de la forma “**command: /bin/bash**”.
- **container\_name:** especifica un nombre de contenedor (si no, se generará automáticamente). Se usa de la forma “**container\_name: micontenedor**”
- **depends\_on:** indica que esta plantilla de contenedor, depende de que se haya creado un contenedor previo de la/s plantilla/s especificada/s. Los servicios también son detenidos en orden inverso a la dependencia. Se pueden ver ejemplos en [https://docs.docker.com/compose/compose-file/compose-file-v3/#depends\\_on](https://docs.docker.com/compose/compose-file/compose-file-v3/#depends_on)
- **env\_file y environment:** permite definir variables de entorno en la plantilla del contenedor.
  - **env\_file** especifica un fichero o lista de ficheros donde están definidas las variables de entorno, similar a “**env\_file: .env**”
  - **environment** especifica una lista de variables de entorno con su valor. Ejemplos en <https://docs.docker.com/compose/compose-file/compose-file-v3/#environment>
- **expose / ports:** permite definir un conjunto de puertos que se exportarán en el contenedor. Ejemplos en <https://docs.docker.com/compose/compose-file/compose-file-v3/#expose> y en <https://docs.docker.com/compose/compose-file/compose-file-v3/#ports>.

- **image:** especifica la imagen en la que se basa el contenedor. No es necesario cuando se especifica a partir de un "Dockerfile".
- **network\_mode:** especifica el modo de red, de forma similar al parámetro --network de Docker. Los modos soportados se detallan en el siguiente enlace [https://docs.docker.com/compose/compose-file/compose-file-v3/#network\\_mode](https://docs.docker.com/compose/compose-file/compose-file-v3/#network_mode)
- **networks:** define las redes a crear para poner en marcha nuestros contenedores. Ejemplos en <https://docs.docker.com/compose/compose-file/compose-file-v3/#networks>
- **restart:** indica cuando debe reiniciarse el contenedor. El valor por defecto es "**no**" (no se reinicia). Otros valores soportados son "**always**" (se reinicia cuando el contenedor se para) y "**on-failure**" (se reinicia si el contenedor se para y devuelve un valor de salida distinto de cero) y "**unless-stoped**" (se reinicia siempre, excepto si el contenedor es parado manualmente con "**docker stop**").
  - Ejemplos <https://docs.docker.com/compose/compose-file/compose-file-v3/#restart>
- **tmpfs:** establece una lista de directorios a montar en formato "tmpfs" en el contenedor. Ejemplos en <https://docs.docker.com/compose/compose-file/compose-file-v3/#tmpfs>
- **volumes:** establece una lista de volúmenes, ya sea en formato "**bind mount**" o volumen Docker. Si quieres reutilizar un volumen entre distintas plantillas, además debes definir la variable "**volumes**" fuera de las plantillas de contenedores. Ejemplos <https://docs.docker.com/compose/compose-file/compose-file-v3/#volumes>

### 3.4 Entendiendo un fichero "docker-compose.yml" con ejemplo Wordpress

En este apartado vamos a ver un ejemplo de "**docker-compose.yml**" para poner en marcha Wordpress. En primer lugar, crearemos un directorio "**miwordpress**" o similar, nos situaremos dentro del directorio y allí crearemos el fichero "**docker-compose.yml**".

El contenido completo del fichero sería el siguiente:

```
version: "3.9"

services:
  db:
    image: mariadb:10.11.2
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
```

```
restart: always
environment:
  WORDPRESS_DB_HOST: db:3306
  WORDPRESS_DB_USER: wordpress
  WORDPRESS_DB_PASSWORD: wordpress
  WORDPRESS_DB_NAME: wordpress
volumes:
  db_data:
```

Procedemos a analizar el contenido del fichero:

### 3.4.1 Palabra clave “version”

```
version: "3.9"
```

Aquí tenemos definido en formato YAML un par variable/valor (la variable es “**version**” y el valor es “**3.9**”). “**Docker Compose**” ha ido ampliándose con el tiempo, así que aquí indicamos indicando la versión del formato de fichero. Este dato es opcional desde la versión “**1.27.0**” de “**Docker Compose**”, ya que es capaz de auto detectarlo.

### 3.4.2 Palabra clave “services” y contenedor “db”

```
services:
  db:
    image: mariadb:10.11.2
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MARIADB_ROOT_PASSWORD: somewordpress
      MARIADB_DATABASE: wordpress
      MARIADB_USER: wordpress
      MARIADB_PASSWORD: wordpress
```

En primer lugar, establece la variable “**services**”. Siguiendo el formato **YAML**, lo que almacena “**services**” es un array asociativo con las variables “**db**” y posteriormente “**wordpress**” (no se ve en este fragmento). Para saber que un elemento está contenido en otro, lo detecta mediante el nivel de los espacios (de forma similar a lenguajes como **Python**).

Cada una de esas variables, define un contenedor. En este caso, la plantilla de contenedor “**db**”, es también un array asociativo en formato **YAML** y define las siguientes propiedades:

```
image: mariadb:10.11.2
```

Define la imagen que utilizará el contenedor.

```
volumes:
  - db_data:/var/lib/mysql
```

Define qué volúmenes contendrá esta imagen. En este caso, los volúmenes en sí, se definen como una lista, de un solo elemento. Sabemos que es una lista porque en YAML empiezan con “-”.

En este caso, se define el volumen “db\_data” que se mapea con el directorio del contenedor “/var/lib/mysql”. Si quisiéramos tener varios volúmenes, podríamos hacer algo similar a:

```
volumes:
  - db_data:/var/lib/mysql
  - otrovolumen:/directorio/otrovolumen
```

Con esto ya hemos visto los 3 principales tipos elementos de YAML (par “variable/valor”, array asociativo, lista).

```
restart: always
```

Este fragmento, usando la sintaxis YAML de par variable/valor, indica que si el servidor se detiene, “**Docker Compose**” lo relance automáticamente (útil para contenedores que puedan caer por un fallo, pero sean necesarios para que la aplicación funcione).

```
environment:
  MARIADB_ROOT_PASSWORD: somewordpress
  MARIADB_DATABASE: wordpress
  MARIADB_USER: wordpress
  MARIADB_PASSWORD: wordpress
```

En este último fragmento, se define la variable “**environment**” del contenedor. Estas contendrán, mediante un vector asociativo, el par variable/valor de cada variable de entorno definida en el contenedor. A efectos prácticos, está definiendo el password de root de **MySQL**, el nombre de una base de datos “wordpress”, un usuario con permisos de root para **MySQL** llamado “wordpress” (necesario para conexiones remotas) y su password como “wordpress”.

### 3.4.3 Palabra clave “services” y contenedor “wordpress”

```
wordpress:
  depends_on:
    - db
  image: wordpress:latest
  ports:
    - "8000:80"
  restart: always
  environment:
    WORDPRESS_DB_HOST: db:3306
    WORDPRESS_DB_USER: wordpress
    WORDPRESS_DB_PASSWORD: wordpress
    WORDPRESS_DB_NAME: wordpress
```

En este caso, se está definiendo, dentro de services, la plantilla del contenedor “**wordpress**”.

Pasamos a revisar sus fragmentos:

```
depends_on:
  - db
```

Indica que este contenedor depende del contenedor **“db”**, por lo cual el contenedor **“db”** deberá ponerse en marcha antes de iniciar nuestro contenedor **“wordpress”**.

```
image: wordpress:latest
```

Como se ha comentado anteriormente, indica la imagen en que se basa el contenedor.

```
ports:  
  - "8000:80"
```

Indica que el puerto **“80”** de ese contenedor se mapea con el puerto **“8000”** del anfitrión.

```
restart: always
```

Como se ha comentado anteriormente, indica que si el contenedor se para, se reinicie automáticamente.

```
environment:  
  WORDPRESS_DB_HOST: db:3306  
  WORDPRESS_DB_USER: wordpress  
  WORDPRESS_DB_PASSWORD: wordpress  
  WORDPRESS_DB_NAME: wordpress
```

De forma similar al anterior punto, definimos el valor de distintas “variables de entorno” del contenedor **“wordpress”**.

#### 3.4.4 Palabra clave “volumes” fuera de las plantillas de contenedores

```
volumes:  
  db_data:
```

Simplemente, indica que el volumen **“db\_data”** está disponible para otros contenedores puestos en marcha con **“Docker Compose”**.

### 3.5 Lanzando nuestro ejemplo Wordpress “docker-compose.yml”

Una vez creado y entendido nuestro fichero **“docker-compose.yml”**, podemos poner en marcha nuestro servicio situándonos en el directorio donde está este fichero y escribiendo:

```
docker compose up -d
```

Con la opción **“up”** indicamos que se interprete la plantilla definida en **“docker-compose.yml”** y con **“-d”** indicamos la ejecución en segundo plano (Nota: si nos da problemas, probemos sin usar **“-d”** para poder recabar mayor información de lo que está ocurriendo en los contenedores). Al acabar observaremos algo similar a:

```
sergi@casa:~/Desktop/miwordpress$ docker compose up -d  
[+] Running 31/2  
  :: db Pulled  
  :: wordpress Pulled  
[+] Running 3/3  
  :: Network miwordpress_default          Created  
  :: Container miwordpress-db-1          Started  
  :: Container miwordpress-wordpress-1    Started
```



Observamos, que tal como hemos definido, primero se descarga y pone en marcha el contenedor basado en la plantilla **“db”**, al cual se le da nombre **“miwordpress-db-1”**, ya que establece el nombre del directorio como base **“miwordpress”**, luego el de la plantilla, y luego un número según el número de contenedor creado.

Posteriormente, se crea de igual forma el contenedor basado en la plantilla **“wordpress”**.

Si accedemos a <http://localhost:8000> observaremos que todo funciona correctamente.

Podemos detener todos los contenedores con el comando:

```
docker compose down
```

## 4. COMANDOS DE “DOCKER COMPOSE”

En este apartado vamos a definir los principales comandos de “Docker Compose”.

### 4.1 Comando “up”

Con el comando **“up”** se interpretará la plantilla **“docker-compose.yml”** y se lanzarán los contenedores necesarios. El detalle de su funcionamiento y parámetros está definido en la siguiente dirección <https://docs.docker.com/compose/reference/up/>

**Algunos de los usos más típicos son:**

```
docker compose up -d
```

Interpreta la plantilla y crea la aplicación, pero con el parámetro **“-d”** actúa en segundo plano.

```
docker compose up -d -f mifichero.yml
```

El parámetro **“-f”** permite indicar un nombre de fichero **YAML** para **“Docker Compose”**.

```
docker compose up -d --force-recreate
```

El parámetro **“--force-recreate”** fuerza a reconstruir contenedores incluso si ya existen en tu máquina.

```
docker compose up -d --scale db=3
```

El parámetro **“--scale”** indica que del servicio **“db”** se creen tres contenedores, realizando un escalado local. El propio **“Docker Compose”** se encargará mediante algoritmo “round robin” de distribuir las peticiones al host **“db”** (como el nombre del servicio) a cada uno de los contenedores escalados. Esto se verá más claramente en uno de los casos prácticos propuestos.

### 4.2 Comando “down”

Con el comando **“down”** se interpretará la plantilla y se paran los contenedores necesarios.

El detalle de su funcionamiento y parámetros está definido en la siguiente dirección <https://docs.docker.com/compose/reference/down/>

**Algunos de los usos más típicos son:**

```
docker compose down
```

Detiene todos los contenedores.

```
docker compose down -v
```

Elimina los volúmenes creados en la plantilla

```
docker compose down -t 5
```

Establece un “timeout” para la parada de contenedores de 5 segundos. El valor por defecto si no se indica nada es de 10 segundos.

#### 4.3 Comando “build”

Comando que simplemente crea las imágenes definidas en la plantilla “**docker-compose.yml**”.

Su funcionamiento se detalla en <https://docs.docker.com/compose/reference/build/>

#### 4.4 Comando “pull”

Comando descarga las imágenes de contenedores necesarias para “**docker-compose.yml**”.

Su funcionamiento se detalla en <https://docs.docker.com/compose/reference/pull/>

#### 4.5 Comando “run”

Comando que permite lanzar un comando contra un servicio concreto definido en la plantilla “**docker-compose.yml**”. El detalle de su funcionamiento está disponible en <https://docs.docker.com/compose/reference/run/>

#### 4.6 Comando “pause”/”unpause”

Con el comando “**pause**”/”**unpause**” se pausan/retoman los contenedores de la plantilla.

El detalle de su funcionamiento y parámetros está definido en la siguiente dirección <https://docs.docker.com/compose/reference/pause/>

#### 4.7 Comando “start”/”stop”

Con el comando “**start**”/”**stop**” podemos iniciar/parar sin parámetros todos los servicios de la plantilla o con parámetros, el servicio que deseemos de esa plantilla. También inicia/para sus dependencias, si las tiene.

El detalle de su funcionamiento y parámetros está definido en:

- <https://docs.docker.com/compose/reference/start/>
- <https://docs.docker.com/compose/reference/stop/>

#### **Ejemplo:**

```
docker compose start db
```

Pone en marcha el servicio “**db**” de la plantilla.

```
docker compose stop db
```

Detiene el servicio “**db**” de la plantilla.

#### 4.8 Comando “ps”

Similar al comando “**docker ps**”, solo que aplicado a los contenedores de la plantilla. El detalle de su funcionamiento y parámetros está definido en la siguiente dirección <https://docs.docker.com/compose/reference/ps/>

#### 4.9 Comando “exec”

El comando “**exec**” es similar a “**docker exec**”. Nos permite ejecutar un comando los contenedores con una plantilla determinada. El detalle de su funcionamiento está definido en

<https://docs.docker.com/compose/reference/exec/>

**Ejemplo:**

```
docker compose exec db mariadb --version
```

#### 4.10 Comando “rm”

El comando “**rm**” elimina todos los contenedores parados según la plantilla determinada en “**docker-compose.yml**”. El detalle de su funcionamiento está definido en <https://docs.docker.com/compose/reference/rm/>

## 5. BIBLIOGRAFÍA

- [1] Docker Docs <https://docs.docker.com/>
- [2] Docker Compose <https://docs.docker.com/compose/>

Sistemas de Gestión Empresarial

# **UD 01. Sistemas ERP-CRM-BI**

---

Actualizado Agosto 2021

## Licencia



**Reconocimiento – NoComercial - 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 debe hacer con una licencia igual a la que regula la obra original.

## Nomenclatura

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

 **Importante**

 **Atención**

 **Interesante**

## ÍNDICE DE CONTENIDO

<b>1. Introducción</b>	<b>3</b>
<b>2. ¿Qué es un sistema ERP-CRM-BI?</b>	<b>3</b>
<b>2.1 Sistemas ERP</b>	<b>3</b>
<b>2.2 Sistemas CRM</b>	<b>3</b>
<b>2.3 Sistemas BI</b>	<b>3</b>
<b>2.4 Sistemas ERP-CRM-BI</b>	<b>3</b>
<b>3. Ventajas e inconvenientes de la implantación de un sistema ERP</b>	<b>4</b>
<b>4. ¿Programar nuestro Sistema ERP o personalización de sistema ERP?</b>	<b>4</b>
<b>5. ¿Licencias privativas o libres para sistemas ERP?</b>	<b>5</b>
<b>6. Retos al implantar un sistema ERP</b>	<b>5</b>
<b>7. Ejemplos de sistemas ERP</b>	<b>6</b>
<b>8. Glosario: conceptos básicos del mundo empresarial</b>	<b>6</b>
<b>9. Bibliografía</b>	<b>8</b>
<b>10. Autores (en orden alfabético)</b>	<b>8</b>

## UD01. SISTEMAS ERP-CRM-BI

### 1. INTRODUCCIÓN

Cualquier empresa para una óptima gestión empresarial y funcionar correctamente debe tener a sus espaldas un sistema informático. Para poner en marcha este tipo de sistemas informáticos, además de una infraestructura a nivel de hardware y comunicaciones, es necesario software específico para la gestión empresarial.

En el mercado siempre han existido programas para gestionar de forma separada o integrada elementos tales como: contabilidad, facturación, gestión comercial, gestión de nóminas, relación con los clientes, y un largo etc...

Una tendencia actual es integrar todo este tipo de sistemas en sistemas ERP-CRM.

### 2. ¿QUÉ ES UN SISTEMA ERP-CRM-BI?

#### 2.1 Sistemas ERP

Un sistema ERP ("Enterprise Resource Planning"), se traduce literalmente como "un sistema de planificación de recursos empresariales", aunque a efectos prácticos en nuestro contexto puede considerarse como un programa de gestión empresarial integrada.

Este sistema de gestión empresarial integrada suele incluir elementos comunes a distintos tipos de empresa tales como contabilidad, ventas, compras, recursos humanos, clientes, y un largo etc.

#### 2.2 Sistemas CRM

Un sistema CRM ("Customer Relationship Management") es un sistema que gestiona las relaciones con los clientes. Este software facilita la automatización de procesos tales como gestión de datos de clientes, gestión y seguimiento de ventas, gestión y seguimiento de campañas, etc....

#### 2.3 Sistemas BI

Un sistema BI ("Business Intelligence") es un sistema la llamada inteligencia de negocio. Estos sistemas proporcionan herramientas para facilitar el análisis y visualización de datos. Estos sistemas permiten analizar el pasado de la empresa (ver qué ocurrió), la situación presente y la situación futura, realizando predicciones y facilitando la toma de decisiones.

#### 2.4 Sistemas ERP-CRM-BI

Por sus características, los sistemas CRM y BI han sido complementos naturales de los sistemas ERP. Actualmente la mayoría de sistemas ERP integran los sistemas CRM y BI. Además, los sistemas ERP incorporan sistemas de plugins modulares que permiten ampliarlos de formas muy variadas.

Por ello, la mayoría de sistemas de gestión integrados son comúnmente llamados simplemente "Sistemas ERP" e incluyen tanto las partes ERP, CRM como BI.

**! Atención:** en adelante, por simplicidad usaremos habitualmente la expresión "Sistemas ERP" como abreviación de "Sistemas ERP-CRM-BI".

### 3. VENTAJAS E INCONVENIENTES DE LA IMPLANTACIÓN DE UN SISTEMA ERP

La implantación de un sistema ERP frente a la implantación de programas de gestión individuales para cada necesidad tiene una serie de ventajas e inconvenientes.

Las principales ventajas son:

- Realizar una única instalación, configuración y mantenimiento, frente a las varias instalaciones, configuraciones y mantenimiento de aplicaciones individuales.
- Facilidad de personalizar tu empresa frente a aplicaciones de gestión individuales sin apenas posibilidad de personalización.
- Sistema integrado, facilitando el trabajo y evitando desarrollar software de comunicación entre distintas aplicaciones.
- Instalación de nuevas funcionalidades de manera modular mediante sistemas de plugins.
- Aplicación de medidas de seguridad de forma centralizada (actualizaciones, protección del sistema, encriptación, etc...)
- La de datos integrados facilita tareas de análisis de datos para la toma de decisiones ("Business intelligence") en la empresa.

Las principales desventajas son:

- Para empresas pequeñas un sistema ERP puede estar sobredimensionado respecto a las necesidades de la empresa.
- La instalación y configuración inicial puede ser más costosa que unas pocas aplicaciones más sencillas de instalar.
- Al ser un único sistema, un ataque llevado a cabo con éxito contra el sistema puede dejar expuesta toda la información utilizada.
- Disponibilidad de más recursos de los que el usuario necesita, que pueden llevarle a confusión y dificultar su uso.

### 4. ¿PROGRAMAR NUESTRO SISTEMA ERP O PERSONALIZACIÓN DE SISTEMA ERP?

Cuando en un entorno empresarial se debe tomar la decisión de implantar un sistema ERP, esta no debe ser tomada a la ligera y debe tener detrás un profundo análisis.

Programar a medida nuestro sistema ERP puede tener ciertas ventajas en tareas de personalización, pero es una tarea que requiere un gran esfuerzo, ya que crear un software que sea robusto y con una amplia funcionalidad es una tarea compleja.

Para la mayoría de organizaciones, utilizar como solución un sistema ERP existente y de cierta entidad, aporta una cierta garantía de robustez, funcionalidad, soporte, etc. **Suele ser la opción más recomendable.** Estos sistemas además suelen tener amplias opciones de personalización tanto a nivel funcional (mediante sistemas de módulos/plugins) como visuales (mediante el uso de temas).

## 5. ¿LICENCIAS PRIVATIVAS O LIBRES PARA SISTEMAS ERP?

Si nos decidimos por utilizar un sistema ERP existente, otro aspecto a considerar es la licencia de software que utilice el sistema ERP. Una licencia cerrada nos puede crear excesiva dependencia con el desarrollador del sistema ERP, con sus consiguientes problemas (cambio de tarifas, cierra empresa, trabas para migración a otro sistema ERP, etc.).

Un sistema ERP con licencia libre nos permite tener una menor dependencia de una empresa concreta (distintas empresas desarrollan para el mismo ERP, si cierra la empresa nosotros y otra empresa puede seguir desarrollando y dando soporte, etc.).

**Generalmente es recomendable implantar sistemas ERP con una licencia libre.** En el caso de implantar sistemas ERP con licencia privativa, se debe estudiar y evaluar concienzudamente los riesgos que supone el depender un software privativo vinculado a una empresa concreta.

Más información sobre los distintos tipos de licencias aquí:

[https://es.wikipedia.org/wiki/Licencia\\_de\\_software](https://es.wikipedia.org/wiki/Licencia_de_software)

## 6. RETOS AL IMPLANTAR UN SISTEMA ERP

Tanto si se utiliza un sistema a medida como si se realiza una personalización de un sistema ERP, surgen retos comunes a la hora de implantar un sistema ERP. Algunos de los principales retos a los que se debe enfrentar la implantación de un sistema ERP son:

- **Problemas técnicos:** problemas en la implantación a nivel de software, hardware, infraestructuras de red, etc.
- **Problemas relacionados con los datos:** nos podemos encontrar problemas para migrar datos ya existentes: dificultades técnicas para la migración, errores de organización de datos, datos incompletos o erróneos, sistemas de codificación de caracteres heterogéneos, sistemas de seguridad (ejemplo: encriptado de datos), etc.
- **Problemas en la mentalidad empresarial:** es posible que muchos cargos no vean las ventajas y posibilidades que ofrece la implantación de un sistema ERP.
  - La solución pasa por intentar que puedan ver qué oportunidades ofrece el sistema ERP para actualizar tanto la organización empresarial como el propio, y que esto se tenga en cuenta en la estrategia de futuro de la empresa.
- **Resistencia de empleados:** es muy frecuente encontrar resistencia de empleados al cambio por distintos motivos: prefieren sistemas obsoletos, no ven las ventajas del cambio, no se les ha formado (o la formación es un extra a su jornada laboral), miedo a perder un puesto de trabajo, etc.
  - La solución pasa por establecer canales de información y retroalimentación con los empleados, ofrecer una formación completa, en horario de trabajo, donde se celebren los logros de los empleados en formación.



## 7. EJEMPLOS DE SISTEMAS ERP

En [https://es.wikipedia.org/wiki/Anexo:Software\\_ERP](https://es.wikipedia.org/wiki/Anexo:Software_ERP) existe un listado con el software ERP popular. En el listado distinguen entre software ERP con licencias gratuitas o de código abierto y sistemas ERP con licencias privativas. Algunos de los más destacados son:

- **Licencia privativa**
  - JD Edwards de Oracle
  - Microsoft Dynamics NAV (Navision)
  - SAP
- **Licencia libre o gratuita**
  - AbanQ
  - Odoo
  - OpenBravo

A lo largo de este módulo trabajaremos con Odoo <https://www.odoo.com>.

## 8. GLOSARIO: CONCEPTOS BÁSICOS DEL MUNDO EMPRESARIAL

En este apartado, introduciremos un glosario con algunos conceptos básicos de la empresa. El fin de este glosario es el de facilitar la comprensión de algunos términos utilizados en el mundo empresarial al alumnado que no esté familiarizado con ellos.

- **Empresa:** persona, organización o institución dedicada a actividades con ánimo de lucro para satisfacer las necesidades de bienes o servicios de la sociedad.
- **Persona física y jurídica:**
  - **Persona física:** es un individuo humano. Este posee derechos y puede contraer obligaciones. En caso de deudas, una persona física debe responder con su patrimonio. Los autónomos, aunque sean emprendedores, se consideran personas físicas.
  - **Persona jurídica:** es una entidad susceptible tanto de adquirir derechos como de contraer obligaciones. En caso de deudas, una persona jurídica responde con los bienes de la entidad. Son personas jurídicas las sociedades, corporaciones, fundaciones, etc.
- **Clientes y proveedores:**
  - **Cliente:** persona física o jurídica que adquiere un bien o servicio a cambio de dinero u otros bienes y servicios.
  - **Proveedor:** un proveedor es una persona física o empresa que provee de bienes y/o servicios a otras personas y/o empresas.
- **Empleados:** un empleado es una persona que brinda servicios a un empleador a cambio de un salario. Los detalles de este vínculo se definen mediante un contrato.

- **Impuestos y tributos:** son una obligación de pago que se realiza al Estado sin que exista una contraprestación directa de bienes o servicios.
  - **IVA:** En España, Impuesto sobre Valor Añadido. Este impuesto puede cambiar según el tipo de bien y servicio o Comunidad Autónoma.
- **Facturas comerciales:** es un documento mercantil que refleja toda la información de una compra/venta: cliente, proveedor, fecha de entrega del producto/servicio, detalle del producto servicio (incluyendo precios unitarios y precio total), momento del devengo (momento en que nace obligación tributaria), impuestos (como IVA), etc.
  - **Facturas completa:** factura comercial que contiene los datos tanto del emisor como del receptor de la misma, y vienen detallados los conceptos con su correspondiente IVA desglosado.
  - **Factura simplificada (Ticket):** es una factura comercial que no contiene todos los datos para ser considerada factura completa.
- **Facturas proforma:** es un documento cuyo encabezado debe contener de forma bien visible el título “proforma”, para evitar confundirse con facturas comerciales. Las facturas proforma sirven como una oferta previa y no tienen valor contable. Estas facturas suelen contener identificador de proveedor y cliente, descripción de los productos y servicios, así como su precio unitario y total.
- **Producción/fabricación, línea de producción y orden de producción:**
  - **Producción/fabricación:** proceso por el cual se elabora un producto. Este proceso suele constar de distintas fases (cada una incluso con diferentes ubicaciones) y algunos elementos asociados tales como materias primas, energía y mano de obra.
  - **Línea de producción:** definición del conjunto de operaciones secuenciales y recursos necesarios para organizar la fabricación de un producto.
  - **Orden de producción:** es la concreción de la fabricación de un producto, indicando la cantidad de producto, fecha de producción y línea de producción utilizada.
- **Logística, almacenamiento, stock/inventario, distribución:**
  - **Logística:** son aquellos procesos de coordinación, gestión y transporte de los bienes comerciales desde el lugar de distribución hasta el cliente final.
  - **Almacenamiento:** un proceso en el cual se guardan tanto los materiales necesarios para el proceso productivo, como el stock de artículos para su venta y distribución.
  - **Stock/Inventario:** es el conjunto de existencias de artículos que posee la empresa tanto para comerciar con ellos como para poder producir bienes y servicios.
  - **Distribución:** proceso por el cual la empresa hace llegar bienes y servicios desde el lugar de almacenamiento hasta el consumidor final.

## 9. BIBLIOGRAFÍA

- Sistemas de Gestión Empresarial IOC:  
[https://ioc.xtec.cat/materials/FP/Materials/2252\\_DAM/DAM\\_2252\\_M10/web/html/index.html](https://ioc.xtec.cat/materials/FP/Materials/2252_DAM/DAM_2252_M10/web/html/index.html)
- Wikipedia:  
[https://es.wikipedia.org/wiki/Sistema\\_de\\_planificaci%C3%B3n\\_de\\_recursos\\_empresales](https://es.wikipedia.org/wiki/Sistema_de_planificaci%C3%B3n_de_recursos_empresales)

## 10. AUTORES (EN ORDEN ALFABÉTICO)

A continuación ofrecemos en orden alfabético el listado de autores que han hecho aportaciones a este documento.

- Jose Castillo Aliaga
- Sergi García Barea

Sistemas de Gestión Empresarial

# **UD 02. Instalación y configuración de un ERP**

---

Actualizado Septiembre 2024

## Licencia



**Reconocimiento – NoComercial - 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 debe hacer con una licencia igual a la que regula la obra original.

## Nomenclatura

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

 **Importante**

 **Atención**

 **Interesante**

## ÍNDICE DE CONTENIDO

<b>1. Introducción</b>	<b>3</b>
<b>1.1 Contexto histórico de los ERP-CRM</b>	<b>3</b>
<b>1.2 Sistemas ERP-CRM en la propia empresa</b>	<b>3</b>
<b>1.3 Sistemas ERP-CRM en la nube</b>	<b>4</b>
<b>1.4 ¿Qué elegir? ¿Sistema en la propia empresa o en la nube?</b>	<b>4</b>
<b>1.5 ¿Y el software para nuestro sistema ERP?</b>	<b>5</b>
<b>2. Tipos de instalación de un sistema ERP</b>	<b>5</b>
<b>3. Licencias de software</b>	<b>6</b>
<b>4. Preparación del servicio para instalar el sistema ERP</b>	<b>7</b>
<b>5. Instalación de un sistema ERP Odoo 17</b>	<b>8</b>
<b>5.1. Odoo 17 en Ubuntu Server</b>	<b>9</b>
<b>5.2. Odoo 17 en Docker: - Parte 1: Contenedor Odoo en producción</b>	<b>9</b>
<b>5.3. Odoo 17 en Docker: - Parte 2: Contenedor Odoo para desarrollo</b>	<b>10</b>
<b>5.4. Odoo 17 en Docker: - Parte 3: Docker Compose para Odoo (RECOMENDADA)</b>	<b>11</b>
<b>6. Puesta en marcha de Odoo 17</b>	<b>13</b>
<b>7. Autores (en orden alfabético)</b>	<b>14</b>

## UD02. INSTALACIÓN Y CONFIGURACIÓN DE UN ERP

### 1. INTRODUCCIÓN

En esta unidad veremos qué factores generales debemos tener en cuenta para la instalación de un sistema ERP y tras ello veremos distintas formas de como instalar “Odoo”, el software que utilizaremos durante este curso.

#### 1.1 Contexto histórico de los ERP-CRM

Las posibilidades para la instalación de un sistema ERP-CRM se han multiplicado en los últimos años. Las tendencias, posibilidades técnicas y precios del hardware, software y la electricidad han ido modificando la manera en la que las empresas utilizan los ordenadores personales y los servidores.

Haciendo un breve e inexacto repaso histórico podemos reflexionar sobre los motivos técnicos y económicos de esta evolución:

- En los primeros años de la informática empresarial, el Mainframe era casi la única opción: un único ordenador muy grande y caro con varias terminales. Este Mainframe era instalado en el edificio de la empresa. Esta solución solo estaba al alcance de grandes empresas.
- La llegada de los computadores personales y miniordenadores hizo que muchas empresas más pequeñas utilizaran esa solución.
  - En ocasiones era un único ordenador pequeño gestionado por la empresa.
  - Mientras tanto, los mainframes seguían ocupando el espacio de las grandes empresas.
  - En sus inicios, el PC y similares tenían carencias tanto a nivel de potencia y como de sistemas operativos orientados al mercado doméstico y no al mercado profesional. Aun así, su aterrizaje en la vida cotidiana influenció a la sociedad y a la industria, llevando que fabricaran servidores con la arquitectura de los PC y similares, pero con una mayor fiabilidad requerida en el contexto industrial.
    - Esto es una gran mejora que permanece hasta hoy en día. Los servidores con esta arquitectura mejoraron considerablemente el hardware con procesadores y memorias más estables y sistemas operativos libres y propietarios más robustos (Unix, Linux, Windows NT/Server...).
- La llegada de Internet supuso otra alternativa. Ahora las empresas podían usar una red común para interconectar sus sedes, tanto con mainframes como con servidores PC, se podía tener un programa centralizado.
  - Al mismo tiempo, algunas empresas pudieron ofrecer servicios por Internet a otras empresas. Este fue el inicio de lo que llamamos “servicios en la nube”.

#### 1.2 Sistemas ERP-CRM en la propia empresa

La opción de tener un servidor en la propia empresa tiene una serie de ventajas e inconvenientes (relacionados entre sí) y cuyos principales problemas vienen dados tanto por los costes económicos como por la seguridad de los datos.

Tener un servidor en la propia empresa con un sistema ERP-CRM (o con cualquier otro tipo de servicio) supone algunos retos:

- Poner en marcha un servicio requiere una inversión inicial en hardware.
- El escalado de hardware para aumentar potencia/disminuir potencia es problemático:
  - Habitualmente se tiene hardware infrautilizado.
  - El aumento de potencia de un servicio requiere escalado vertical (aumento de prestaciones del servidor) o escalado horizontal (comprar más equipos).
    - El hardware limita las posibilidades de cómputo, así que es imposible escalar para aumentar potencia en momentos puntuales.
- El mantenimiento del sistema informático, el suministro de energía y la seguridad del sistema son gastos y responsabilidades asociados a la empresa.
  - Una ventaja de la gestión de datos interna es que no están en ordenadores fuera de la empresa, evitando riesgos tales como el espionaje industrial.

### 1.3 Sistemas ERP-CRM en la nube

El gasto tanto energético como de potencia de computación en una sola empresa suele ser desigual durante el día, suponiendo esto una ineficiencia energética. Por este motivo, algunas empresas, especialmente aquellas que ofrecen servicios por Internet, se plantearon vender esa potencia cuando no la necesitaban o compartirla entre muchos clientes. Esto propició el nacimiento de lo que hoy llamamos “los servicios en la nube”.

Con la llegada de los “servicios en la nube” (que realmente son servidores de otra empresa) las empresas pagan una cuota (fija, por tiempo de cómputo, por uso, etc.), pero a cambio:

- No realizan gastos relacionados con el hardware (instalación y escalado).
- No realizan gastos relacionados con el consumo eléctrico.
- Se reducen de forma drástica los gastos en mantenimiento y seguridad.
- Facilitan el acceso: para operar con estos sistemas la empresa solo necesita un dispositivo (ordenador personal, smartphone, etc.) con conexión a Internet.
- Si se necesita más potencia, solo hay que contratarla (escalado vertical/horizontal).

Aun así, el uso de “servicios en la nube” poseen varios inconvenientes:

- En algunos contextos, puede resultar más caro que poner en marcha tu la infraestructura.
- Los datos están almacenados físicamente en un servidor de otra empresa, con posibles problemas relacionados (por ejemplo, espionaje industrial).
- Para el uso del sistema dependen tanto de una buena conexión a Internet como del buen funcionamiento general de la empresa que presta servicios.

### 1.4 ¿Qué elegir? ¿Sistema en la propia empresa o en la nube?

No existe una respuesta “tajante” a esta pregunta, ya que depende del contexto y es algo a estudiar concienzudamente antes de implantar un sistema.

Hoy en día en los entornos empresariales conviven ambas opciones mencionadas, e incluso opciones híbridas (servidor en la empresa, pero apoyo puntual o algunos servicios en la nube).

Los principales factores a la hora de tomar esta decisión son:

- El cumplimiento de las leyes de protección de datos. Este punto sobre todo influye en la decisión de contratar o no un “servicio en la nube”, ya que guardando determinados datos en determinados servicios en la nube podríamos estar incumpliendo la ley.
  - En Europa existe el reglamento RGPD y en España la legislación vigente en materia de protección de datos viene definida por la LOPDGDD.
- El precio de la electricidad y el consumo del hardware.
  - Existen dispositivos hardware orientados a tener un bajo consumo.
- El precio del hardware.
  - Existen dispositivos orientados a tareas de servidor muy baratos.

### 1.5 ¿Y el software para nuestro sistema ERP?

Durante la introducción hemos hablado de costes relacionados con el sistema que alojaría nuestro ERP. Pero... ¿Y el software?. Existen multitud de opciones software ERP, tanto gratuitos, de pago, libres, mixtos (partes libres, partes de pago), etc.

Existe un software ERP llamado Odoo, que se presenta en dos versiones: “Community Edition” (software libre y gratuito) y “Enterprise Edition” (de pago).

En este curso utilizaremos **Odoo “Community edition”**, ya que es libre y gratuito. Por simplicidad, cuando nos refiramos a Odoo, estaremos refiriéndonos a esta versión.

Más información en [https://www.odoo.com/es\\_ES/](https://www.odoo.com/es_ES/) y <https://github.com/odoo/odoo>.

## 2. TIPOS DE INSTALACIÓN DE UN SISTEMA ERP

En este apartado vamos a tratar las distintas maneras en la que se puede instalar un sistema ERP y a mencionar los pros y contras de cada tipo de solución.

Comenzaremos con las soluciones no recomendadas (“cutres”), para pasar aquellas simples pero correctas e ir evolucionando hacia las más sofisticadas:

- **La carpeta compartida:** se trata de una práctica cada vez menos usada pero que se mantiene en muchas pequeñas empresas.
  - Hace años, algunos programas de gestión estaban pensados para un solo usuario. La única opción que daban para poder ser accedidos por varios ordenadores en una red local era compartir la carpeta de la base de datos y configurar varias instalaciones para acceder al mismo archivo. Además, esa “base de datos” era un fichero de “Access” o ficheros de texto plano.
  - **Esta solución no es correcta ni recomendable:** es una solución que propone muchos problemas, tanto con el acceso simultáneo, como problemas tanto de seguridad como de integridad.
- **Instalación On-Premise:** aquí entra en juego un servidor instalado en la propia empresa.



- Los ordenadores cliente pueden acceder al sistema ERP mediante un software cliente del propio ERP o incluso mediante un navegador web.
  - Se debe configurar correctamente la red del sistema tanto para evitar conexiones externas no permitidas, como para proporcionar acceso seguro (si se requiere) desde fuera de la red.
  - Si se necesita acceder desde fuera de la red local, hay que configurar correctamente la red y la seguridad del acceso externo. Se suele pagar por el software entero o la instalación y el mantenimiento.
- Si el servidor funciona de una forma segura y esta opción es viable económicamente para la empresa, es una solución correcta.
- **Instalación como servicio en la nube:** en esta solución prescinde de servidor en la empresa y se subcontrata la computación. Simplifica la instalación y el acceso externo. Además, se paga por lo que se necesita y es fácilmente escalable. Los “servicios en la nube” puede ser de muchos tipos, pero se suele distinguir entre:
  - **IAAS** (Infraestructura como servicio): nos proporcionan acceso a servidores, máquinas virtuales o contenedores. Hay que instalar y asegurar el sistema operativo, el ERP y todo lo demás. Puede que la empresa de la nube ya nos lo ofrezca con un sistema operativo preinstalado, pero deberíamos tener control total de ese sistema operativo.
    - **Ejemplo:** los VPS entran dentro de este tipo de nube.
  - **PAAS** (Plataforma como servicio): En esta nube ya está el sistema operativo y algunos programas configurados. Sobre este sistema se puede desplegar el sistema ERP.
    - **Ejemplo:** lo que consideramos como “hosting tradicional” (PHP, MySQL, etc.).
  - **SAAS** (Software como servicio): en este caso ya está el ERP instalado y nos dan acceso a determinadas características según contratemos (cantidad de usuarios, acceso simultáneo, espacio de almacenamiento, copia de seguridad, etc.). No hay que preocuparse de nada más, pero estás limitado al programa que el proveedor ofrece.
    - **Ejemplo:** soluciones tipo Gmail o Google Drive, donde tenemos todo puesto en marcha y solo consumimos el servicio.
  - Aparte de estos tipos de “servicios en la nube”, podemos tener algunos servicios de forma individual en la nube, como por ejemplo servicios de bases de datos (como Firebase) o servicios de APIs REST o GRAPHQL, etc. En cualquier caso, deben garantizar una alta disponibilidad, seguridad y escalabilidad.

### 3. LICENCIAS DE SOFTWARE

A la hora de elegir el software de un sistema ERP, uno de los factores es la licencia y su precio. Vamos a distinguir entre licencias libres y licencias propietarias.

Se considera licencia libre aquella que permita la modificación y redistribución del software. Cualquier licencia que no permita la modificación y redistribución del software se considera licencia propietaria.

**! Atención:** Una licencia libre no implica que el software sea gratuito. Una licencia

propietaria no implica que el software sea de pago.

El modelo de negocio del software propietario es fácil de intuir a primera vista (creo un producto y lo vendo, cobro servicios asociados, etc.). El modelo de negocio del software libre es muy diverso, desde el pago por el mantenimiento, cursos, instalación o personalización del sistema, donaciones o venta de productos adicionales.

Existe una gran cantidad de licencias libres. En el siguiente enlace existe una comparativa de las principales

[https://en.wikipedia.org/wiki/Comparison\\_of\\_free\\_and\\_open-source\\_software\\_licences](https://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licences)

Una de las primeras licencias libres y la más garantista es GPL y sus distintas versiones. Otras de las licencias libres más conocidas son la MIT, BSD o Apache.

#### 4. PREPARACIÓN DEL SERVICIO PARA INSTALAR EL SISTEMA ERP

Cada sistema ERP es distinto y posee necesidades de potencia distintas. Estas necesidades suelen depender de factores como:

- Cantidad de datos que alberga el sistema.
- Número de usuarios simultáneos
- Tecnología utilizada por el software ERP.
- Sistema operativo sobre el que corre el software ERP.

Para poner en marcha un sistema ERP es recomendable utilizar hardware diseñado para servidores. La calidad de los componentes, la refrigeración o tecnologías como el ECC en la memoria RAM hacen más estables estos ordenadores frente a PC domésticos u otras alternativas para el usuario final (Raspberry Pi, Arduino, etc.).

Los usuarios que interactúan con el sistema ERP, sí pueden usar PCs, tablets, smartphones u otros sistemas domésticos como cliente del servicio.

Otro aspecto a tener en cuenta es la seguridad, teniendo en cuenta tanto medidas de seguridad pasiva como activa. Deben tenerse en cuenta aspectos como:

- Protección contra personas (ataques físicos y remotos).
- Protección contra los elementos: calor, humedad, sobretensiones o incendios.
- Disponibilidad del sistema y copia de seguridad.
- Cumplimiento de la legislación vigente.

No es tarea de este módulo explorar todas las protecciones disponibles, pero deben tenerse en cuenta y exigirlas en una instalación real. Algunas de las medidas de protección básicas son:

- El servidor debe tener un SAI que lo proteja y lo mantenga encendido en caso de fallo de la red eléctrica.
- Es recomendable que el servidor en sí disponga de elementos (procesador, RAM, discos, etc.) por encima de la potencia mínima necesaria para que funcione el sistema ERP.
  - No es necesario pasarse demasiado, ya que si lo hacemos incrementaremos tantos

gastos de hardware y el consumo eléctrico sin obtener un gran beneficio. Normalmente, estos programas tienen una documentación en la que describen los requerimientos mínimos.

- Respecto a la seguridad de los datos, se recomienda redundancia en los discos, ya sea con RAID o con sistemas de archivos redundantes como ZFS o BTRFS.
  - Esta redundancia no excluye la necesidad de establecer una política de copias de seguridad externas al sistema.

Una vez puesto en marcha el servidor hay que elegir el sistema operativo base y un posible sistema de virtualización. En el caso del sistema ERP Odoo, que tratamos en estos apuntes, para una puesta en marcha en un sistema real se recomienda Ubuntu Server.

Este sistema operativo puede ser el sistema instalado en la máquina o estar virtualizado.

La virtualización podemos realizarla con:

- Máquinas virtuales con hipervisor (tipo Virtualbox). Estas máquinas realizan una simulación completa del hardware, influyendo esto en una disminución de rendimiento.
- Una alternativa a la virtualización con hipervisor, pero con mejor rendimiento (cercano al rendimiento nativo) son los contenedores, ya sean completos como LXD o contenedores de aplicaciones como Docker.

Sistemas operativos base como Proxmox simplifican la gestión tanto de máquinas virtuales como de contenedores, copias de seguridad de los mismos y almacenamiento en red.

En caso de optar por la nube, si contratamos un IAAS también debemos tener en cuenta la potencia contratada. De hecho, es más importante afinar correctamente, ya que podemos incrementar los costes sin tener un beneficio en cuanto a rendimiento.

## 5. INSTALACIÓN DE UN SISTEMA ERP ODOO 17

En este apartado en primer lugar trataremos los requisitos mínimos para instalar el sistema ERP Odoo en su versión 17. Tras ello, explicaremos cómo realizar una instalación manual en un sistema Ubuntu. Finalmente, explicaremos cómo poner en marcha Odoo 17 mediante contenedores Docker usando Docker y Docker Compose.

A efectos prácticos, Odoo 17 no necesita mucha potencia para funcionar. Puede funcionar sin problemas en cualquier ordenador con varios núcleos y al menos 512MB de RAM, aunque con esa configuración, si recibe muchos accesos simultáneos, la máquina se puede quedar corta.

Como en todas las aplicaciones que consultan bases de datos, el acceso al disco puede suponer un cuello de botella. Por eso es recomendable utilizar unidades SSD, RAIDs o sistemas de archivos como ZFS o BTRFS con varios discos.

Odoo 17 funciona perfectamente en máquinas virtuales y contenedores. Algunas opciones de configuración pueden ser:

- Sistema operativo: Ubuntu Server e instalación directa de Odoo.
- Sistema operativo: Ubuntu Server, Virtualización con KVM o similar. Las máquinas virtuales

tendrán instalado Ubuntu Server.

- Sistema operativo: Ubuntu Server, contenedores con LXD de Ubuntu.
- Sistema operativo: Ubuntu Server, contenedores Docker.
- Sistema operativo: Proxmox, máquinas virtuales o contenedores LXC gestionados por Proxmox.

Como se ve, en todas las ocasiones se eligen máquinas reales y virtuales Ubuntu. Esto es porque Odoo está desarrollado en ese sistema y esto nos ayuda a garantizar que la implantación del sistema funcione correctamente.

### 5.1. Odoo 17 en Ubuntu Server

Para realizar una instalación manual de Odoo, recomendamos seguir los pasos indicados en su documentación oficial, disponible en:

<https://www.odoo.com/documentation/17.0/administration/install.html>

### 5.2. Odoo 17 en Docker: - Parte 1: Contenedor Odoo en producción

💬 **Interesante:** si no conoces como usar “Docker”, puedes serte muy útil revisar este curso con ejemplos prácticos <https://sergarb1.github.io/CursoIntroduccionADocker/>

💬 **Interesante:** si ya conoces como usar “Docker”, puedes serte muy útil esta “CheatSheet” <https://raw.githubusercontent.com/sergarb1/CursoIntroduccionADocker/main/FuentesCurso/Docker%20CheatSheet%20COMPLETA.pdf>

Para poner en marcha Odoo 17 en modo producción crearemos dos contenedores:

- El primer contenedor contendrá la base de datos PostgreSQL en su versión 15.
- El segundo contenedor contendrá el servidor Odoo.


Creamos el contenedor de PostgreSQL con:

```
docker run -d -v /home/usuario/OdooDesarrollo/dataPG:/var/lib/postgresql/data -e
POSTGRES_USER=odoo -e POSTGRES_PASSWORD=odoo -e POSTGRES_DB=postgres --name db
postgres:15
```

Donde los parámetros indican lo siguiente:

- **“-d”**: ejecuta el contenedor en segundo plano.
- **“-v /home/usuario/OdooDesarrollo/dataPG:/var/lib/postgresql/data”**: monta el directorio del contenedor “/var/lib/postgresql/data” (donde se encuentra la información almacenada por PostgreSQL) en el directorio del anfitrión “/home/usuario/OdooDesarrollo/dataPG”.
  - El fin de esto es almacenar la información de la base de datos en la máquina anfitriona.
- **“-e POSTGRES\_USER=odoo -e POSTGRES\_PASSWORD=odoo -e POSTGRES\_DB=postgres”**: establece dentro del contenedor esas variables de entorno. A efectos prácticos, esas variables le indican que creen en la base de datos un usuario “odoo”, con contraseña “odoo” y que la base de datos a usar se llama “postgres”.
- **“--name db”**: nombre que le daremos a nuestro contenedor Docker.

- **“postgres:15”**: indicamos que usaremos la imagen de Docker Hub llamada “postgres” y de entre ellas usaremos la versión 15.
  - Más información de esta imagen en [https://hub.docker.com/\\_/postgres](https://hub.docker.com/_/postgres)

 **Importante:** si en lugar del parámetro “-d”, utilizamos el parámetro “-t”, ejecutaremos el contenedor en primer plano y veremos en la terminal información del inicio de PostgreSQL u Odoo. Esto es interesante para detectar problemas.

Con el contenedor PostgreSQL ya en marcha, creamos el contenedor con Odoo con:

```
docker run -d -v -p 8069:8069 --name odooprod --user="root" --link db:db odoo:17
```

Donde los parámetros indican lo siguiente:

- **“-d”**: ejecuta el contenedor en segundo plano.
- **“-p 8069:8069”**: mapeamos el puerto 8069 del contenedor (donde accedemos a Odoo) al puerto 8069 de la máquina anfitrión, para poder acceder a Odoo con <http://localhost:8069>
- **“--name odooprod”**: damos el nombre “odooprod” a nuestro contenedor.
- **“--user=“root””**: fuerza a que el contenedor se ejecute internamente como “root” y no como el usuario “odoo” que va por defecto en la imagen.
- **“--link db:db”**: enlazamos con una red virtual este contenedor con el contenedor “db”.

### 5.3. Odoo 17 en Docker: - Parte 2: Contenedor Odoo para desarrollo

Para lanzar Odoo en un contenedor preparado para desarrollo, creamos también dos contenedores. Creamos el contenedor de PostgreSQL de forma similar a como hicimos en el apartado anterior con:

```
docker run -d -v /home/usuario/OdooDesarrollo/dataPG:/var/lib/postgresql/data -e POSTGRES_USER=odoo -e POSTGRES_PASSWORD=odoo -e POSTGRES_DB=postgres --name db postgres:15
```

Un servidor de producción está pensado para ponerlo en marcha, realizar pocas paradas y mantener el contenido del contenedor. Se suele hacer copias completas del contenido para únicamente restaurar el contenedor en caso de urgencia.

Sin embargo, en entornos de desarrollo, es más habitual “romper cosas”. A efectos prácticos, es habitual reiniciar contenedores o incluso borrarlos y construirlos de nuevo.

Para ello, vamos a modificar la forma de crear los contenedores guardando algunas cosas en volúmenes para realizar “persistencia” del servicio Odoo.

Creamos el contenedor de Odoo, con algunas diferencias respecto al anterior:

```
docker run -d -v /home/usuario/OdooDesarrollo/volumesOdoo/addons:/mnt/extra-addons -v /home/usuario/OdooDesarrollo/volumesOdoo/firestore:/var/lib/odoo/filestore -v /home/usuario/OdooDesarrollo/volumesOdoo/sessions:/var/lib/odoo/sessions -p 8069:8069 --name odoodev --user="root" --link db:db -t odoo:17 --dev=all
```

A continuación, comentamos las diferencias:

- “-v /home/usuario/OdooDesarrollo/addons:/mnt/extra-addons”: la imagen de Odoo 17 por defecto carga los módulos del directorio del contenedor “/mnt/extra-addons”, por lo cual mapeamos ese directorio a nuestro directorio de la máquina anfitrión “/home/usuario/OdooDesarrollo/addons”, donde desarrollaremos usando un IDE externo.
- “-v /home/usuario/OdooDesarrollo/volumesOdoo/firestore:/var/lib/odoo/filestore
- ” y “-v /home/usuario/OdooDesarrollo/volumesOdoo/sessions:/var/lib/odoo/sessions”: como en desarrollo es posible que paremos y montemos muchas veces los contenedores Docker, montamos estos volúmenes para tener persistencia de los directorios de Odoo “firestore” y la “sessions”. Para ello, mapeamos esos directorios del contenedor a nuestra máquina anfitrión dentro del directorio “/home/usuario/OdooDesarrollo/volumesOdoo/”.
- “--dev=all”: le pasa ese parámetro a Odoo para facilitar tareas de desarrollo.
  - El detalle de que realiza esta opción se puede ver en el siguiente enlace <https://www.odoo.com/documentation/17.0/es/developer/reference/cli.html>

! **Atención:** para poder desarrollar sin problemas, es recomendable darle todos los permisos al directorio “/home/usuario/OdooDesarrollo/volumesOdoo/addons”, con un comando similar a “**sudo chmod -R 777 /home/usuario/volumesOdoo/addons**”.

Con esto, tendremos listo nuestro entorno de desarrollo “Dockerizado”. Hemos conseguido que los contenedores corran de manera aislada los servicios de base de datos y Odoo, mientras que nosotros podremos desarrollar usando un IDE instalado en el anfitrión trabajando dentro del directorio “/home/usuario/OdooDesarrollo/addons”.

#### 5.4. Odoo 17 en Docker: - Parte 3: Docker Compose para Odoo (RECOMENDADA)

💬 **Interesante:** si conoces como usar “Docker Compose”, puedes serte muy útil revisar este curso con ejemplos prácticos <https://sergarb1.github.io/CursoIntroduccionADocker/>

💬 **Interesante:** si ya conoces como usar “Docker Compose”, puedes usar esta “CheatSheet” <https://raw.githubusercontent.com/sergarb1/CursoIntroduccionADocker/main/FuentesCurso/Docker%20CheatSheet%20COMPLETA.pdf>

Docker Compose es una herramienta que nos facilita el despliegue de varios contenedores usando una configuración definida en un fichero. Este fichero por defecto debe llamarse “**docker-compose.yml**”.

Si nos situamos en el directorio donde esté nuestro fichero “docker-compose.yml”, podemos iniciar el servicio completo simplemente escribiendo:

```
docker compose up -d
```

Al lanzar esta orden, se crearán en el directorio actual:

- **Carpeta “addons”**: ahí desarrollaremos nuestros módulos mediante un IDE externo.
- **Carpeta “pgData”**: ahí se almacenará la persistencia de datos de nuestra base de datos.

**! Atención:** para poder desarrollar sin problemas, es recomendable darle todos los permisos al directorio “addons”, con un comando similar a “**sudo chmod -R 777 ./addons**”.

Podemos parar el servicio completo simplemente escribiendo:

```
docker compose down
```

Adjuntamos a esta unidad un zip con el fichero “docker-compose.yml” para entorno de producción y con el fichero “docker-compose.yml” para entorno de desarrollo. A continuación, además mostramos el contenido del fichero “docker-compose.yml” para entorno de desarrollo.

**Fichero “docker-compose.yml” (desarrollo):**

```
version: '3.3'

services:
  #Definimos el servicio Web, en este caso Odoo
  web:
    #Indicamos que imagen de Docker Hub utilizaremos
    image: odoo:17
    #Indicamos que depende de "db", por lo cual debe ser procesada primero "db"
    depends_on:
      - db

    # Port Mapping: indicamos que el puerto 8069 del contenedor se mapeara con el
    # mismo puerto en el anfitrión
    # Permitiendo acceder a Odoo mediante http://localhost:8069
    ports:
      - 8069:8069


    # Mapeamos el directorio de los contenedores (como por
    # ejemplo "/mnt/extra-addons" )
    # en un directorio local (como por ejemplo en un
    # directorio "./volumesOdoo/addons")
    # situado en el lugar donde ejecutemos "Docker compose"
    volumes:
      - ./volumesOdoo/addons:/mnt/extra-addons
      - ./volumesOdoo/odoo/filestore:/var/lib/odoo/filestore
      - ./volumesOdoo/odoo/sessions:/var/lib/odoo/sessions

    # Definimos variables de entorno de Odoo
    environment:
      - HOST=db
      - USER=odoo
      - PASSWORD=odoo
    # Indica que pasa ese parametro al arrancar el servicio Odoo
```

```
command: --dev=all
#Definimos el servicio de la base de datos
db:
  image: postgres:15
  # Definimos variables de entorno de PostgreSQL
  environment:
    - POSTGRES_PASSWORD=odoo
    - POSTGRES_USER=odoo
    - POSTGRES_DB=postgres
  # Mapeamos el directorio del contenedor "var/lib/postgresql/data" en un
  directorio "./volumesOdoo/dataPostgreSQL"
  # situado en el lugar donde ejecutemos "Docker compose"
  volumes:
    - ./volumesOdoo/dataPostgreSQL:/var/lib/postgresql/data
```

## 6. PUESTA EN MARCHA DE ODOO 17

Una vez realizada la instalación con cualquiera de las alternativas propuestas anteriormente, accederemos mediante nuestro navegador a Odoo mediante <http://localhost:8069> y deberemos realizar una configuración inicial. Aquí un ejemplo de dicha configuración:



Warning, your Odoo database manager is not protected. To secure it, we have generated the following master password for it:

**53cu-diye-asm4**

You can change it below but be sure to remember it, it will be asked for future operations on databases.

Master Password	<input type="password" value="••••••••••"/>
Database Name	<input type="text" value="postgre"/>
Email	<input type="text" value="correo.cuenta.odoo@gmail.com"/>
Password	<input type="password" value="••••"/>
Phone number	<input type="text"/>
Language	<input type="text" value="Spanish / Español"/>
Country	<input type="text" value="Spain"/>
Demo data	<input type="checkbox"/>
<input type="button" value="Create database"/> or restore a database	

A primera vista, se nos mostrará un “Password maestro” que podemos cambiar si queremos. Deberemos almacenar en un lugar seguro ese “Password maestro” para poder recuperar nuestro

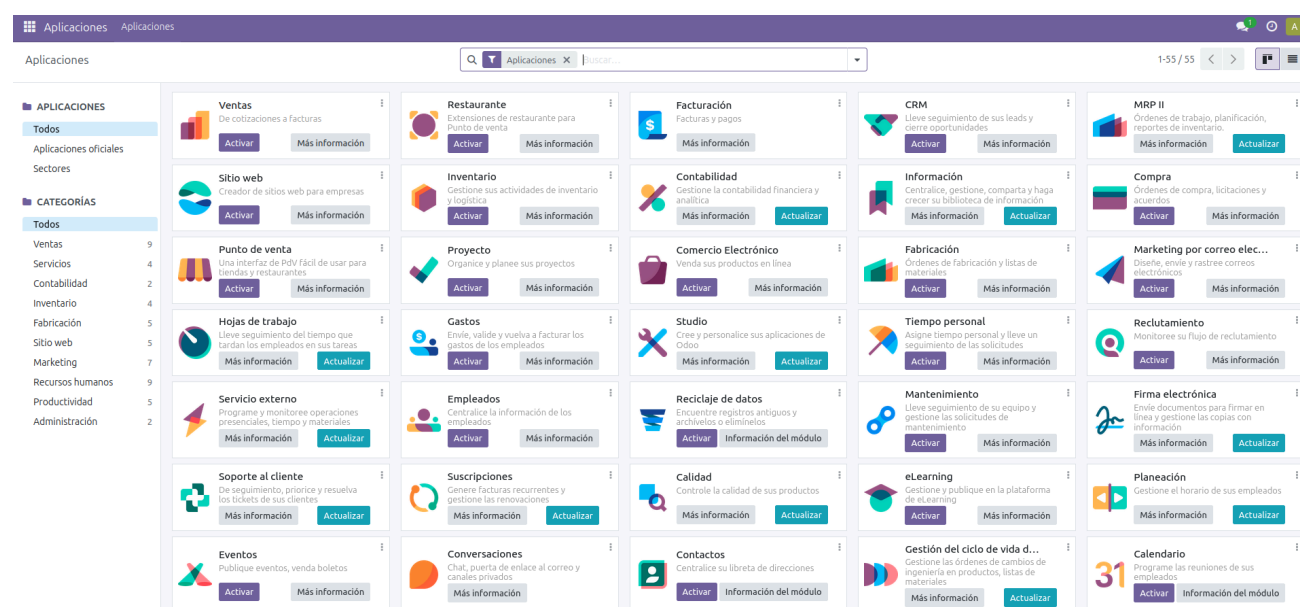


sistema ante problemas de usuarios.

! **Atención:** si perdemos el “Password maestro”, podemos ponerlo en blanco editando “/etc/odoo/odoo.conf” (o el fichero “.odoorc” dentro del “home”) y eliminando (o poner un comentario con #) el campo “admin\_password”. Si lo hacemos así y reiniciamos el servicio, Odoo nos dirá que no hay “Password maestro” y nos sugerirá que creemos un nuevo password.

Además, se nos pedirá configurar Odoo según los parámetros de nuestra instalación. En esta configuración crearemos un usuario administrador, e indicaremos nuestro país (esto realizará algunas adaptaciones para empresas locales) e idioma de Odoo. Incluso nos permite cargar la instalación con datos de demostración (útil para realizar pruebas, conocer cómo funciona Odoo, etc.).

Una vez esté todo listo, al pulsar “Create database” se creará inicializará Odoo. Tened paciencia (tarda unos minutos). Si todo ha ido bien, llegaréis a una pantalla similar a la siguiente:



Esto indica que Odoo 17 se ha instalado correctamente y ya podemos trabajar con él.

## 7. AUTORES (EN ORDEN ALFABÉTICO)

A continuación ofrecemos en orden alfabético el listado de autores que han hecho aportaciones a este documento.

- Jose Castillo Aliaga
- Sergi García Barea