

# Puesta en Producción Segura

## Unidad 0. Herramientas Necesarias

### Virtualización. Contenedores: Docker



*"Docker no solo acelera el desarrollo, también acelera el riesgo si olvidas la seguridad."*

**— *Inspirado en prácticas DevSecOps***

# Objetivos

- Comprender la diferencia entre máquinas virtuales y contenedores.
- Conocer la arquitectura de Docker y sus componentes principales.
- Aprender a ejecutar contenedores y manejar su ciclo de vida.
- Entender qué son las imágenes Docker y cómo utilizarlas desde un Registro.
- Crear imágenes personalizadas mediante un Dockerfile.
- Gestionar almacenamiento en Docker (volúmenes y persistencia de datos).
- Configurar escenarios multicontenedor con Docker Compose.



# Contenidos

- 01** Virtualización y contenedores
- 02** Docker
- 03** Ejecución de contenedores en Docker
- 04** Imágenes en Docker
- 05** Almacenamiento en Docker
- 06** Redes en Docker
- 07** Escenarios Multicontenedor
- 08** Creación de Imágenes



01

# Virtualización y contenedores



# Arquitectura máquinas virtuales

- Creación o despliegue de algún tipo de recurso tecnológico a través de software (SO, almacenamiento, aplicaciones).
- Son capaces de ejecutar software como si se tratase de una máquina física real.
- Una máquina virtual o "guest" cuenta con un SO completo que se ejecuta de manera aislada al "host".
  - Los procesos del "guest" no tienen relación con los del "host" y están limitados en cuanto a recursos.
- Su uso proporciona ciertas ventajas: portabilidad, escalabilidad, optimización del uso de los recursos físicos, etc.

# ¿Para qué se utiliza la virtualización?

- Aislamiento e independencia de servicios y contenidos.
- Laboratorio de pruebas.
- Virtualización de arquitecturas de las que no se dispone.
- Creación de clúster de máquinas y sistemas distribuidos.
- Herramienta de aprendizajes

# Ventajas e inconvenientes de la virtualización

## Ventajas

- Importante ahorro económico.
- Seguridad.
- Mayor aprovechamiento de recursos.
- Migración en vivo.
- Importante ahorro energético.

## Inconvenientes

- Muchos sistemas dependen de un solo equipo físico.
- Penalizaciones en rendimiento.

# Máquina virtual vs contenedor



## Máquinas virtuales

- Se ejecutan sobre un hipervisor.
- Crean un SO completo → se consumen más recursos.
- El estado de la máquina depende de su ejecución y no es inmutable.
- Limitaciones para la creación de múltiples máquinas virtuales simultáneas.



## Contenedores

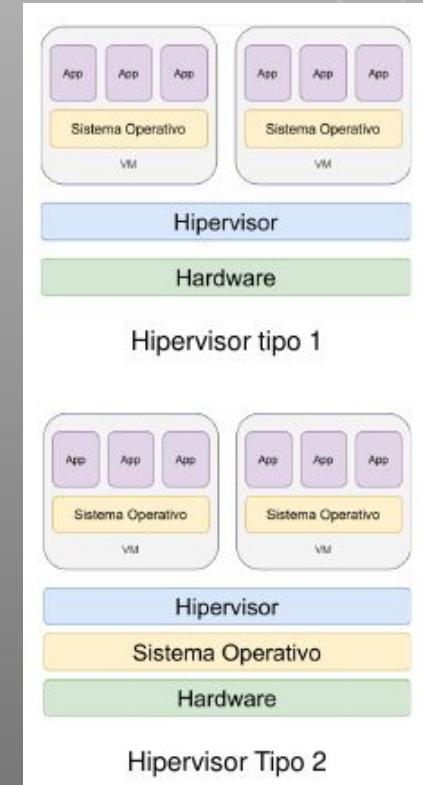
- Se ejecutan sobre un motor de contenedores.
- Sólo contienen lo necesario para ejecutar la aplicación.
- Son inmutables.
- Poca sobrecarga. En el "host" pueden ejecutarse miles de contenedores.

# Tipos de virtualización

- **Emulación.**  
Imitación o suplantación via software: Qemu, Wine ...
- **Virtualización por hardware de tipo 1.**
- **Virtualización por hardware de tipo 2.**
- **Virtualización ligera, a nivel de sistema operativo o basada en contenedores.**

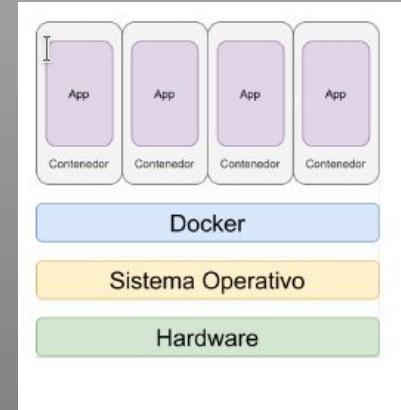
# Arquitectura máquinas virtuales

- Un hipervisor es un software específico encargado de gestionar máquinas virtuales dentro de una máquina real.
- Permite la abstracción del hardware físico y la asignación de recursos a cada máquina virtual.
- En general, existen dos tipos de hipervisores:
  - Tipo 1 (bare metal): se ejecutan directamente sobre el hardware del sistema → KVM, Hyper-V, ESXi...
  - Tipo 2 (hosted): se ejecutan sobre un SO "host" que a su vez se ejecuta sobre el hardware → VirtualBox, QEMU, VMware Workstation...



# Arquitectura contenedores

- Proporcionan un entorno de ejecución ligero para la ejecución de aplicaciones → Docker, LXD, CRI-O...
- Sólo almacenan las aplicaciones, bibliotecas, ficheros, etc. que son necesarios para el correcto funcionamiento de la aplicación.
  - No contienen un SO completo.
- Comparten/utilizan los recursos del propio SO "host".
- Permiten la creación de entornos replicables, consistencia entre entornos de prueba y de producción, reducir costes, etc.

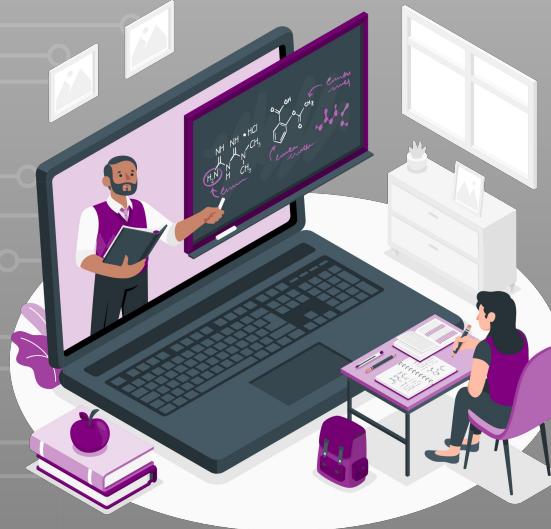


# Clasificación de contenedores

- **Contenedores de Sistemas:** El uso que se hace de ellos es muy similar al que hacemos sobre una máquina virtual: se accede a ellos (normalmente por ssh), se instalan servicios, se actualizan, ejecutan un conjunto de procesos, ... Ejemplo: LXC(Linux Container).
- **Contenedores de Aplicación:** Se suelen usar para el despliegue de aplicaciones web. Ejemplo: Docker, Podman, ...

# Contenedores de aplicaciones: ventajas

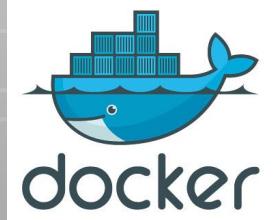
- Portabilidad: Los contenedores encapsulan una aplicación y todas sus dependencias de manera aislada.
- Aislamiento: Los contenedores proporcionan un nivel de aislamiento entre la aplicación y el sistema operativo del anfitrión.
- Eficiencia en el uso de recursos: Al compartir el núcleo del sistema operativo y solo incluir las bibliotecas y dependencias necesarias, los contenedores son más eficientes en términos de recursos en comparación con las máquinas virtuales.
- Despliegue rápido: Los contenedores permiten la creación, el despliegue y la escalabilidad rápida de aplicaciones. La capacidad de implementar contenedores en cuestión de segundos o minutos facilita la entrega continua y el despliegue ágil de aplicaciones.



02

# Introducción a Docker





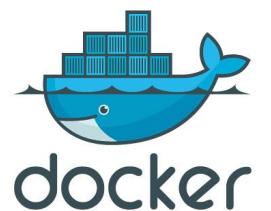
# Docker

- Es un proyecto open source que permite la creación y despliegue de contenedores.
- Está escrito en el lenguaje de programación Go y se sirve de características del kernel de Linux para proporcionar su funcionalidad.
- Principalmente está formado por los siguientes elementos:
  - Namespaces: utiliza los nombres de espacio del kernel para proporcionar un entorno aislado de procesos → contenedor.
  - Cgroups: hace uso de los grupos de control del kernel para la asignación compartida de recursos y aislamiento de los contenedores.
  - UnionFS/OverlayFS: permite combinar distintos sistemas de archivos formando un único sistema (ejecución de múltiples instancias de contenedor).
- Hace uso de AppArmor, SELinux y de las capacidades del kernel para añadir seguridad a los contenedores.

<https://docs.docker.com/engine/security/apparmor>

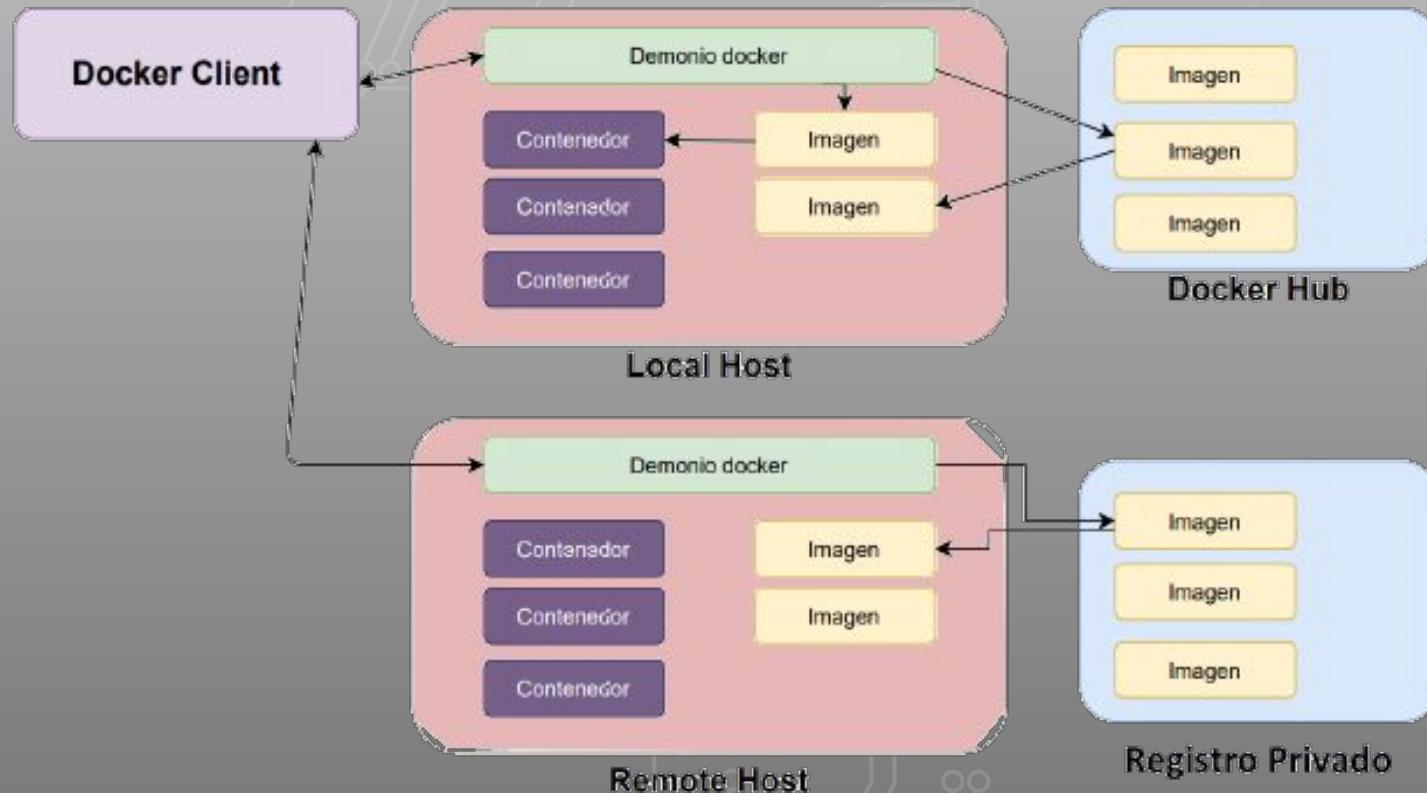
# Objetos Docker

- Contenedor
  - Entorno aislado donde se ejecuta la aplicación.
  - Tiene su propio sistema de ficheros con todas las dependencias.
  - Puede estar conectado a una red virtual y utilizar almacenamiento adicional.
  - Utiliza los recursos del servidor donde se está ejecutando
- Imagen
  - Una imagen es una plantilla de sólo lectura con instrucciones para crear un contenedor Docker.
  - Contiene el sistema de fichero que tendrá el contenedor.
- Volumenes.
- Redes
- Plugins



# Arquitectura de docker

- **Docker daemon** (dockerd): motor encargado de gestionar los componentes relacionados con Docker. Permanece a la escucha de peticiones a la Docker API enviadas desde el cliente.
- **Docker client** (CLI): interfaz de línea de comandos que nos permite interactuar con el motor de Docker.
- **Registros de imágenes**: servicio de almacenamiento de imágenes (i.e. repositorios). Pueden ser tanto públicos como privados.
- **Docker Desktop**: Aplicación gráfica que permite la gestión sencilla de objetos Docker. Incluye el demonio y el cliente Docker.



# Instalación de Docker

Instalación de **Docker Engine** (Incluye el cliente y el demonio Docker):

- Distribuciones Linux:

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

Instalación de **Docker Desktop** (Incluye el cliente, el demonio Docker, aplicación gráfica para gestionar objetos Docker y otras funcionalidades):

- Docker Desktop (Windows):

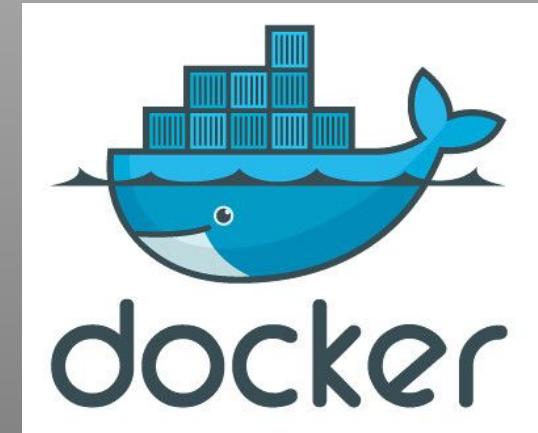
<https://docs.docker.com/desktop/setup/install/windows-install/>

- Docker Desktop (Linux):

<https://docs.docker.com/desktop/setup/install/linux/>

# Instalación de Docker

- Añadir a nuestro usuario el grupo docker para poder utilizar Docker con usuarios sin privilegios:
  - sudo usermod -aG docker \$USER
- Verificar la instalación:
  - docker
  - docker version
  - docker info



Fuente de la imagen:  
<https://www.flickr.com/photos/xmodulo/1409888813>



03

# Ejecución de contenedores en Docker



# Docker CLI

Desde terminal de comandos podemos invocar a los comandos de Docker

Terminal



\$ docker

Usage: docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:

- run Create and run a new container from an image
- exec Execute a command in a running container
- ps List containers
- build Build an image from a Dockerfile
- pull Download an image from a registry
- push Upload an image to a registry
- images List images

# docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Crea e inicia un contenedor con la imagen especificada y la ejecución de la órdenes indicadas.

- docker run ubuntu echo 'Hello world'

Imagen: ubuntu

Ejecución: "echo 'Hello world'"

Terminal



```
docker run ubuntu echo 'Hello world'  
Hello world
```

# El “Hola Mundo” de docker

- docker run hello-world

Terminal



```
$ docker run hello-world
```

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest:
sha256:a13ec89cdf897b3e551bd9f89d499db6ff3a7f44c5b9eb
8bca40da20eb4ea1fa
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.



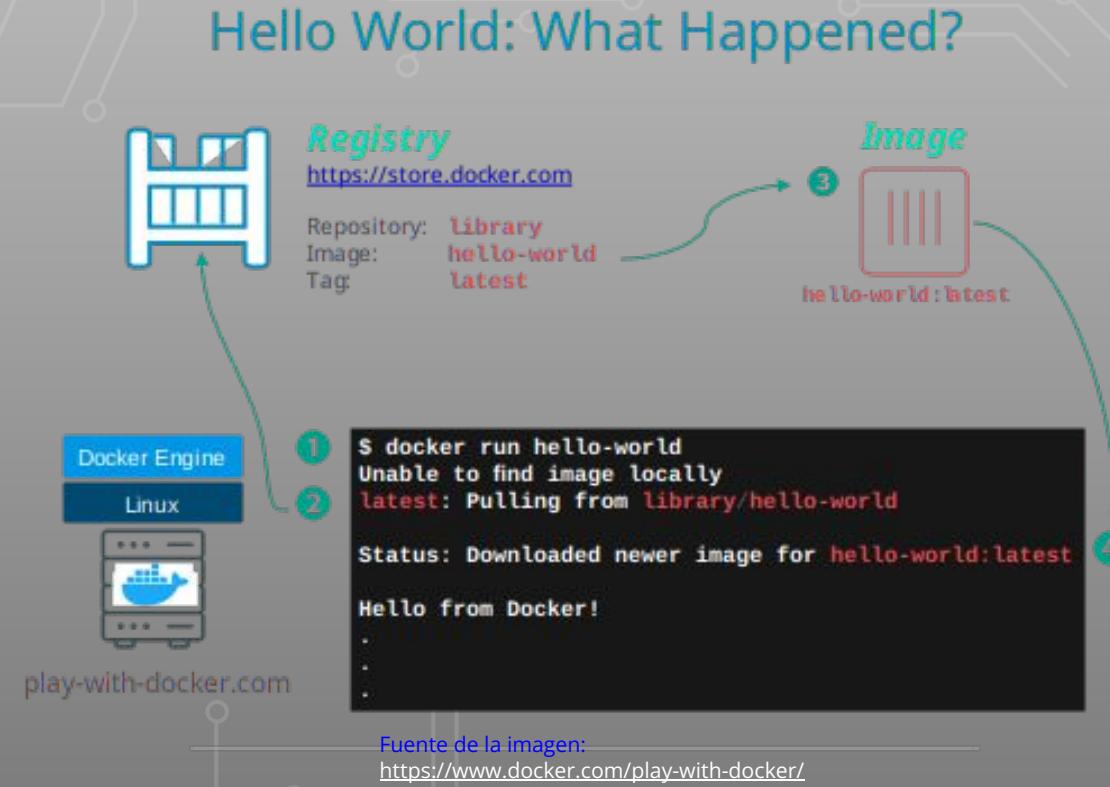
Financiado por  
la Unión Europea  
NextGenerationEU



Para ver qué ocurre cuando ejecutamos docker run, podemos abrir un terminal y ejecutar en él docker event

### ¿Que pasa cuando ejecutarmos docker run hello-world?

- 1 Ejecución de orden en Docker cli
2. Búsqueda de la imagen en el repositorio local.
3. Descarga desde repositorio remoto de la imagen.
4. Creación de contenedor a partir de la imagen en demonio docker.



# docker ps [OPTIONS]

Muestra información de los contenedores en ejecución.

- docker ps

Muestra información de todos los contenedores.

- docker ps -a

Terminal



```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND
CREATED	STATUS	PORTS NAMES
e365106f39fe	wordpress:5.4	"docker-entrypoint.s..."
8 weeks ago	Exited (0) 8 weeks ago	files-WPweb-1

# docker images [OPTIONS] [REPOSITORY[:TAG]]

Muestra información de las imágenes docker presentes en repositorio local/otro repositorio

- docker images
- docker image ls

Terminal



```
$ docker images
REPOSITORY          TAG
IMAGE ID            CREATED        SIZE
kalilinux/kali-rolling    latest
168a2513f74f   2 months ago  123MB
```

# Opciones en ejecución de contenedores

- Indicar el nombre que va a tener un contenedor
- name <Contenedor>
- Poner hostname
- hostname
- Eliminación automática una vez finalizada ejecución
- rm

Terminal



```
$ docker run --name miContenedor  
--hostname ContenedorHello --rm  
hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
e6590344b1a5: Pull complete  
Digest:  
sha256:ec153840d1e635ac434fab5e377081f17e0e15  
afab27beb3f726c3265039cff  
Status: Downloaded newer image for  
hello-world:latest
```

Hello from Docker!

# Contenedor Interactivo

- Con **-i** Abrimos una sesión interactiva
- Con **-t** creamos un pseudo-terminal con el que nos conectamos al contenedor creado.

**docker run -it --name  
contenedor2 -h cont1  
ubuntu bash**

El contenedor se para cuando salimos de él.

Terminal



```
$ docker run -it --name contenedor2  
-h cont1 ubuntu bash  
root@cont1:/#
```

# Contenedor demonio

- Con **-d** el contenedor es **daemon** o **demonio**. La ejecución se hace en segundo plano y su ejecución es de forma desatendida.

```
docker run -d --name  
contenedor4 ubuntu bash -c  
"while true; do echo hello world;  
sleep 1; done"
```

El contenedor queda en ejecución en segundo plano .

Terminal



```
$ docker logs contenedor2  
root@cont1:/# exit  
exit  
root@cont1:/#
```

# docker logs [OPTIONS] CONTAINER

Nos muestra los **logs** o **eventos** del contenedor.

- docker logs contenedor2

Terminal



```
$ docker logs contenedor2
```

# docker rm [-f | -l | -v] CONTAINER [CONTAINER...]

Borra un contenedor que se encuentra parado.

- docker rm contenedor2  
Forzado (-f) si está en ejecución lo para.
- \$ docker rm -f contenedor2  
Si el contenedor tiene asociado un volumen, lo elimina también.
- \$ docker rm -v contenedor2

Terminal



```
$ docker rm contenedor 2  
contenedor2
```

# Variables de entorno -e Nombre\_variable=valor

- Para crear una variable de entorno al crear un contenedor usamos el flag -e o --env:
  - docker run -it --name contenedor5 -e USUARIO=prueba ubuntu bash
- El proceso que inicializa el contenedor crea esa variable en el contenedor.

# docker start [OPTIONS] CONTAINER [CONTAINER...]

Inicia un contenedor parado  
- docker start contenedor2

Terminal



```
$ docker start contenedor2  
contenedor2
```

# docker attach [OPTIONS] CONTAINER

Conecta al usuario al proceso en ejecución de un contenedor.

Suele usarse con contenedores interactivos

- docker attach  
contenedor2

Terminal



```
$ docker start contenedor2  
contenedor2
```

# docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

**Ejecuta comandos o scripts**

en un contendor

Suele usarse con  
contenedores interactivos

- docker exec -it  
contenedor2 /bin/bash

En este caso el comando  
**/bin/bash** nos abre un  
terminal interactivo en el  
contenedor.

Terminal



```
$ docker exec -it contenedor2  
/bin/bash  
contenedor2
```

# docker stop [OPTIONS] CONTAINER [CONTAINER...]

Parar un contenedor en ejecución

- docker stop contenedor2

Reiniciar un contenedor

- docker restart  
contenedor2

Terminal



```
$ docker stop contenedor2  
contenedor2
```

# docker pause/unpuase [OPTIONS] CONTAINER

Pausar un contenedor en ejecución

- docker pause  
contenedor2

Reanudar un contenedor pausado

- docker unpause  
contenedor2

Terminal



```
$ docker stop contenedor2  
contenedor2
```

# docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Permite ejecutar comandos en contenedores iniciados.

- docker exec hora-container2  
ls

Terminal



```
$ docker run -d --name hora-container2 ubuntu bash -c 'while true; do date +"%T" >> hora.txt; sleep 1; done'  
hora.txt  
$ docker exec hora-container2 ls  
$ docker exec hora-container2 cat hora.txt  
19:31:34
```

# docker cp [OPTIONS] CONTAINER:SRC\_PATH DEST\_PATH|- docker cp [OPTIONS] SRC\_PATH|- CONTAINER:DEST\_PATH

Copia archivos entre nuestro equipo anfitrión y el contenedor o viceversa

- echo "Curso Docker">>docker.txt
- \$ docker cp docker.txt hora-container2:/tmp

Terminal



```
$ echo "Curso Docker">>docker.txt
$ docker cp docker.txt
hora-container2:/tmp
Successfully copied 2.05kB to
hora-container2:/tmp
```

# docker inspect [OPTIONS] NAME|ID [NAME|ID...]

Nos proporciona información de cualquier objeto docker.

- docker inspect [OPTIONS] NAME|ID [NAME|ID...]
  - Id del contenedor.
  - Puertos abiertos y sus redirecciones.
  - bind mounts y volúmenes usados.
  - El tamaño del contenedor
  - El comando que se esta ejecutando en el contenedor.
  - El valor de las variables de entorno.
- Y muchas más cosas....

Terminal



```
$ docker inspect hora-container2
```

```
[  
 {  
   "Id":  
     "7b45b0ae4cf453696169382e4a5961897adbd6dbd7f6f  
     0ba5469fee3feeb2dca",  
   "Created": "2025-08-01T17:37:59.321773814Z",  
   "Path": "bash",  
   "Args": [  
     "-c",  
     "while true; do date +\"%T\" >> hora.txt; sleep 1;  
   done"  
 ],  
   "State": {  
     "Status": "running",
```



04

# Imágenes en Docker



# ¿Qué es una imagen Docker?

- Una imagen es una plantilla de sólo lectura con instrucciones para crear un contenedor Docker.
- Contiene el sistema de fichero que tendrá el contenedor.
- Además establece el comando que ejecutará el contenedor por defecto.
- Podemos crear nuestras propias imágenes o utilizar las creadas por otros y publicadas en un registro.
- Un contenedor es una instancia ejecutable de una imagen.

# Registro de imágenes

- El Registro Docker es un componente donde se almacena las imágenes Docker.
- Tenemos un registro local donde se almacenan las imágenes desde las que vamos a crear contenedores.
- Existen registros remotos que nos permiten distribuir las imágenes.
- Los registros pueden ser públicos o privados.
- El registro público que nos ofrece Docker se llama Docker Hub.

# Nombre de las imágenes

El **nombre** de una imagen suele estar formado por tres partes:

**usuario/nombre:etiqueta**

- **usuario**: El nombre del usuario que la ha generado. Las imágenes oficiales en Docker Hub no tienen nombre de usuario.
- **nombre**: Nombre significativo de la imagen.
- **etiqueta**: Nos permite versionar las imágenes. De esta manera controlamos los cambios que se van produciendo en ella. Si no indicamos etiqueta, por defecto se usa la etiqueta latest, por lo que la mayoría de las imágenes tienen una versión con este nombre.

**jmmedinac03/nessus\_plugins:latest**

MANIFEST DIGEST

sha256:f1d8b44fc1c612214d028e7b5c5d672dfc34381c1a4d0b2c51d5538519379bac



# Ejemplo de etiquetas

Ejemplo de etiquetas para la imagen [nginx](#)

Cada imagen está identificada por un identificador. Una misma imagen, puede estar etiquetada por etiquetas diferentes.

Por ejemplo, en la imagen [nginx](#) las etiquetas *1.29.0*, *1.29*, *1*, *mainline*, *latest* y otras corresponden a la misma imagen.

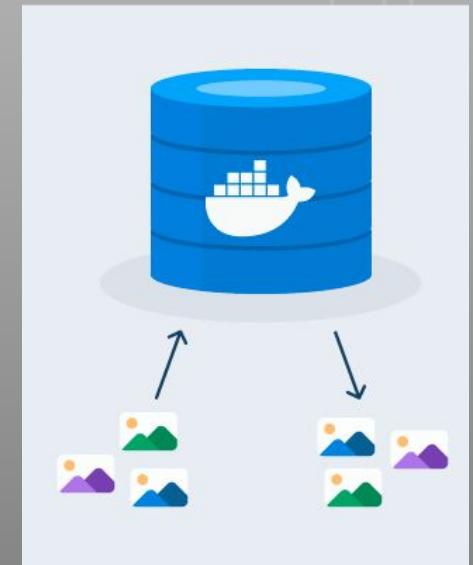
## Supported tags and respective Dockerfile links

- [1.29.0](#) , [mainline](#) , [1](#) , [1.29](#) , [latest](#) , [1.29.0-bookworm](#) , [mainline-bookworm](#) , [1-bookworm](#) , [1.29-bookworm](#) , [bookworm](#) ↗
- [1.29.0-perl](#) , [mainline-perl](#) , [1-perl](#) , [1.29-perl](#) , [perl](#) , [1.29.0-bookworm-perl](#) , [mainline-bookworm-perl](#) , [1-bookworm-perl](#) , [1.29-bookworm-perl](#) , [bookworm-perl](#) ↗
- [1.29.0-otel](#) , [mainline-otel](#) , [1-otel](#) , [1.29-otel](#) , [otel](#) , [1.29.0-bookworm-otel](#) , [mainline-bookworm-otel](#) , [1-bookworm-otel](#) , [1.29-bookworm-otel](#) , [bookworm-otel](#) ↗
- [1.29.0-alpine](#) , [mainline-alpine](#) , [1-alpine](#) , [1.29-alpine](#) , [alpine](#) , [1.29.0-alpine3.22](#) , [mainline-alpine3.22](#) , [1-alpine3.22](#) , [1.29-alpine3.22](#) , [alpine3.22](#) ↗
- [1.29.0-alpine-perl](#) , [mainline-alpine-perl](#) , [1-alpine-perl](#) , [1.29-alpine-perl](#) , [alpine-perl](#) , [1.29.0-alpine3.22-perl](#) , [mainline-alpine3.22-perl](#) , [1-alpine3.22-perl](#) , [1.29-alpine3.22-perl](#) , [alpine3.22-perl](#) ↗
- [1.29.0-alpine-slim](#) , [mainline-alpine-slim](#) , [1-alpine-slim](#) , [1.29-alpine-slim](#) , [alpine-slim](#) , [1.29.0-alpine3.22-slim](#) , [mainline-alpine3.22-slim](#) , [1-alpine3.22-slim](#) , [1.29-alpine3.22-slim](#) , [alpine3.22-slim](#) ↗
- [1.29.0-alpine-otel](#) , [mainline-alpine-otel](#) , [1-alpine-otel](#) , [1.29-alpine-otel](#) , [alpine-otel](#) , [1.29.0-alpine3.22-otel](#) , [mainline-alpine3.22-otel](#) , [1-alpine3.22-otel](#) , [1.29-alpine3.22-otel](#) , [alpine3.22-otel](#) ↗

# ¿Para que sirven las etiquetas de las imágenes?

Si utilizamos el nombre de una imagen sin indicar la etiqueta, se toma por defecto la etiqueta **latest** que suele corresponder a la última versión de la aplicación.

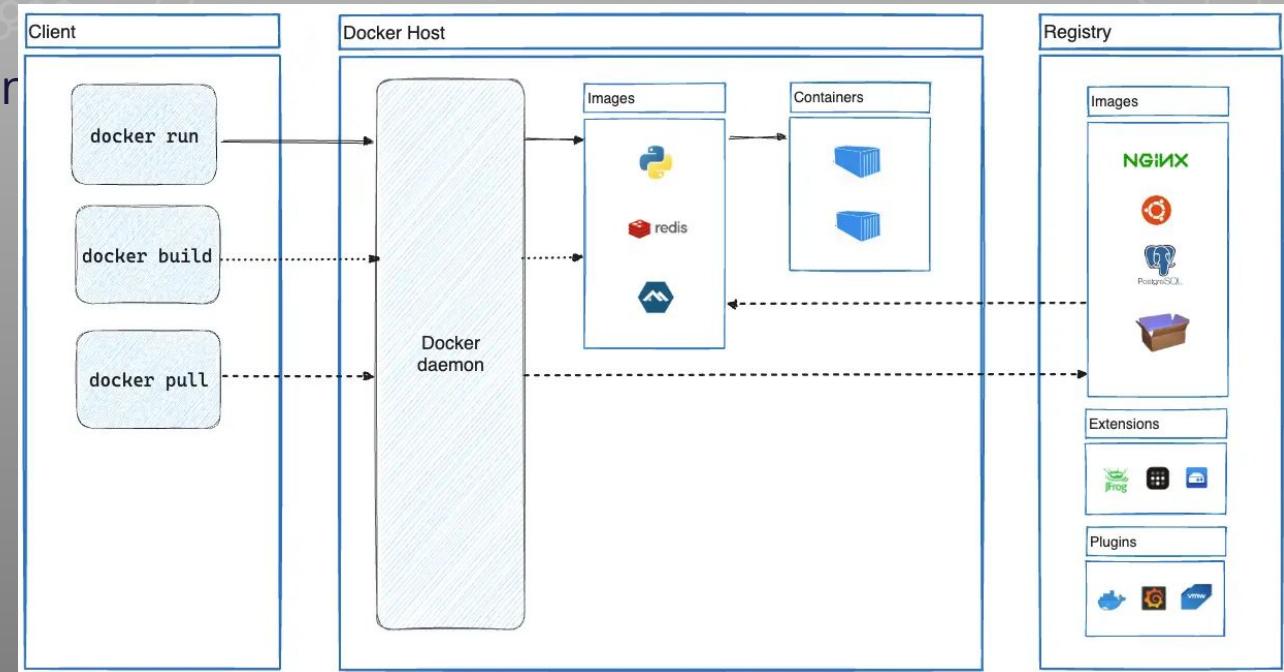
- Normalmente las etiquetas nos permiten versionar las imágenes.
- Podemos seguir observando que algunas etiquetas, nos indican además de la versión, los servicios que tienen instalada la imagen, por ejemplo si usamos la etiqueta **slim** es una versión minimizada, las **alpine** están basadas en linux alpine y la **latest** o **bookworm** están basadas en debian.



Fuente de la imagen:  
<https://octopus.com/blog/custom-docker-registry>

# Registro de imágenes: Docker Hub

- Docker Hub es un registro público de imágenes Docker.
- En nuestro equipo guardamos un registro privado de imágenes.



Fuente de la imagen: <https://docs.docker.com/>

# Registro de imágenes: Docker Hub

- **Repositorio:** Un repositorio nos permite guardar una imagen. De cada imagen podemos tener distintas versiones. Las etiquetas nos permiten identificar cada versión.
- **Usuarios:** Nos podemos dar de alta en Docker Hub para subir y distribuir nuestras imágenes.
- El nombre de una imagen tiene el siguiente formato:  
usuario/nombre:etiqueta.
- Las imágenes oficiales en Docker Hub no tienen nombre de usuario.
- Imagenes de confianza:



Fuente de la imagen: <https://docs.docker.com/>

# docker pull [OPTIONS] NAME[:TAG|@DIGEST]

Descargar imagen a repositorio local

- docker pull nginx

**Recordamos**, para listar las imágenes presentes en repositorio local:

- docker images
- docker image ls

**Recordamos**, para mostrar características de una imagen del repositorio local:

- docker inspect nginx

# docker search [OPTIONS] Termino-a-buscar

Para buscar imágenes presentes en **Docker Hub** desde la línea de comandos.

- docker search nessus

Obtenemos información de

- Nombre de la imagen
- Descripción
- Puntuación
- Si es oficial

## docker rmi

Borramos imagen de repositorio local

- **docker rmi nginx**
- **docker image rm nginx**

No podemos eliminar una imagen si tenemos algún contenedor creado a partir de ella.

- Podemos forzar borrado con:
- **docker image rm -f nginx**

Terminal



```
$ docker rmi nginx
```

Untagged: nginx:latest

Untagged:

nginx@sha256:84ec966e61a8c7846f509da7eb081c55c1d568  
17448728924a87ab32f12a72fb

Deleted:

sha256:2cd1d97f893f70cee86a38b7160c30e5750f3ed6ad86c  
598884ca9c6a563a501

Deleted:

sha256:3c1159cd77f83ede793fc21502ae30b39b04378b6b1b  
625451d701d555cc1cb9

Deleted:

sha256:221aaeee6035b28a744a32189b7ff7853e07f65741853  
66a2272fb5e525e4e99e

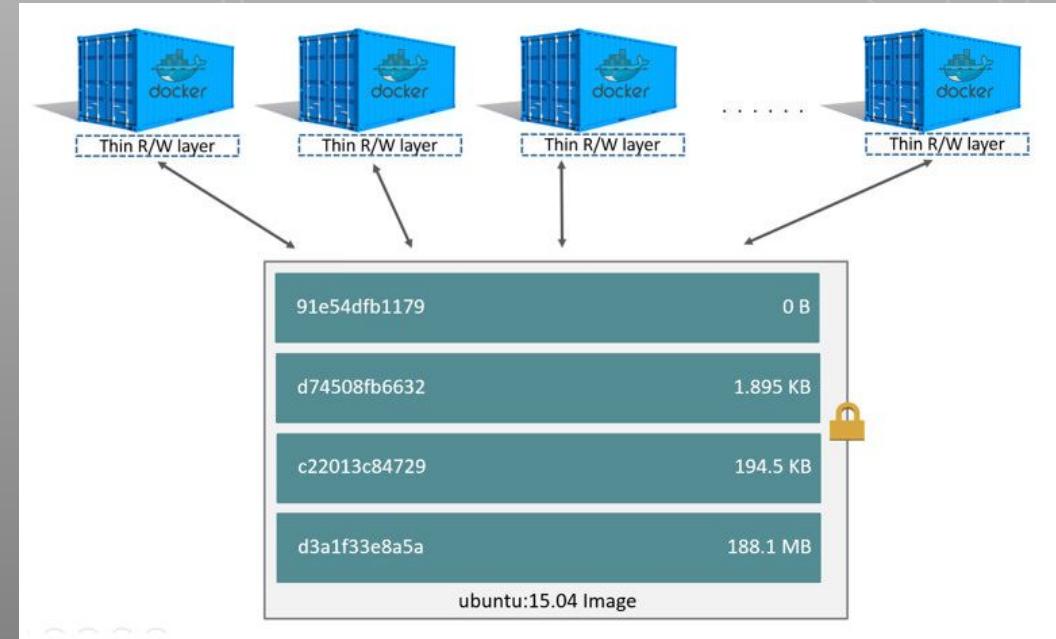
# Imagenes en Docker: Capas

Las imágenes se construyen a partir de **capas ordenadas**.

- Cada capa es sólo un conjunto de diferencias en el **Sistema de Archivos** con respecto a la capa anterior.
- En el proceso de creación de las imágenes, los comandos que cambian el sistema de archivos (instalaciones, modificación de ficheros, copiar ficheros,...) producen una nueva capa.
- Cuando tomas todas las capas y las apilas, obtienes una nueva imagen que contiene todos los cambios acumulados.
- Hay muchas imágenes que contienen capas comunes( p.e. Basadas en debian). Esas capas comunes sólo se almacenan una vez.

# Imagenes en Docker: Capas

- Las capas de la imagen son únicamente de lectura, por lo que se añade una nueva capa de lectura-escritura.
- Todos los cambios efectuados al contenedor específico son almacenados en esa capa.
- Los contenedores son efímeros, por que cuando lo borramos, se borra la capa del contenedor, por lo que se pierde todos sus datos.



Fuente de la imagen: <https://iesgn.github.io/>



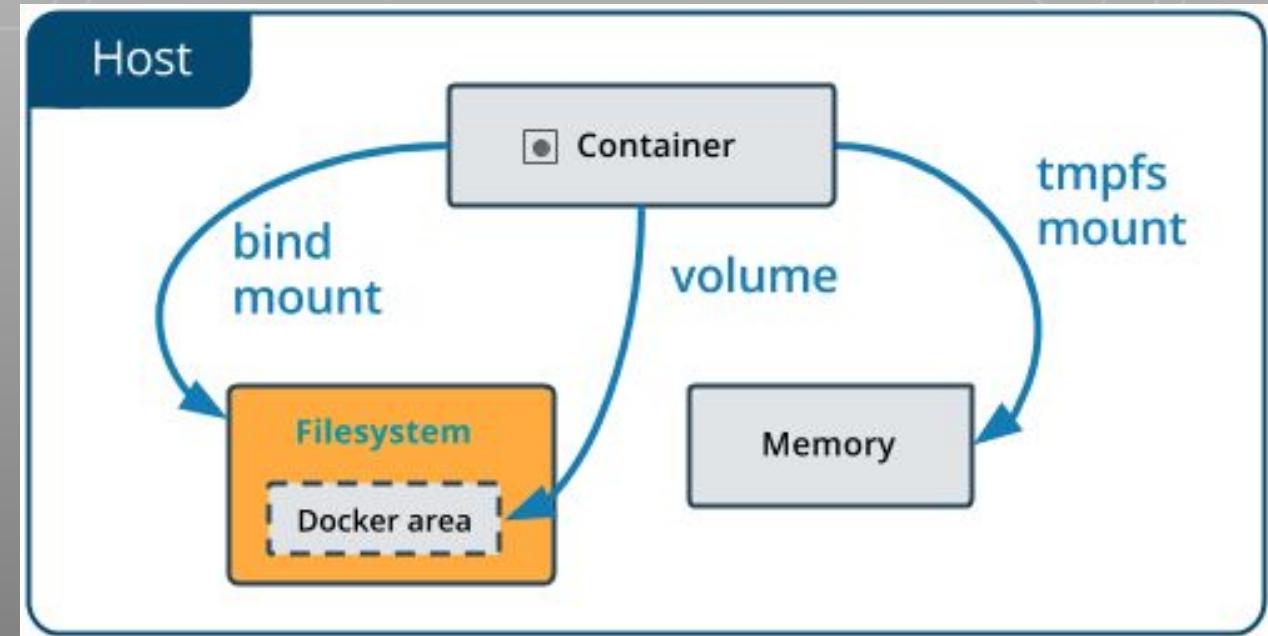
05

# Almacenamiento en Docker



# Almacenamiento en docker

- Problema:
  - Al borrar contenedor perdemos los datos
- Solución:
  - Volúmenes Docker
  - Bind Mount
  - Tmpfs mounts



Fuente de la imagen:  
<https://docker-docs.uclv.cu/storage/bind-mounts/>

# Volúmenes Docker

- Los volúmenes son creados y gestionados por Docker.
- Un volumen corresponde a un directorio en el Host Docker. La información se almacena **/var/lib/docker/volumes** y es gestionada directamente por Docker.
- Un volumen dado puede ser montado en múltiples contenedores simultáneamente.
- Cuando ningún contenedor en ejecución está utilizando un volumen, el volumen sigue estando disponible para Docker y no se elimina automáticamente.
- Cuando creas un volumen, puede tener nombre o ser anónimo.
- A los volúmenes anónimos se les da un nombre aleatorio que se garantiza que sea único dentro de un Host Docker dado.

# Bind Mount

- Con **Bind Mount** podemos hacer que un archivo o directorio de la máquina anfitriona se monte en un contenedor.
- No es necesario que el archivo o directorio ya exista en el Host Docker.  
Se crea bajo demanda si aún no existe.
- La gestión la realizamos con el SO anfitrión, no utilizamos el cliente Docker para esa gestión.
- Al realizar cambios sobre los ficheros del bind mount en el anfitrión, se cambian directamente en el contenedor..

# Cuando usar Volume Docker y Bind Mount

- Volume Docker:

- Compartir datos entre múltiples contenedores en ejecución.
- Almacenar los datos de su contenedor en un servidor remoto.
- Hacer copias de seguridad, restaurar o migrar datos de un Host Docker a otro: Copiar el directorio /var/lib/docker/volumes/<nombre-volumen>.

- Bind Mount:

- Compartir archivos entre máquina anfitriona y contenedores.
- Compartir código fuente o artefactos de construcción entre un entorno de desarrollo en el Host Docker y un contenedor.
- Compartir accesos entre contenedores Docker y otras aplicaciones.

# Operaciones con volúmenes

Crea un volumen docker

- docker volume create miVolumen

Elimina un volumen docker

- docker volume rm miVolumen

Fuerza la eliminación de un volumen docker aunque esté siendo utilizado.

- docker volume rm -f miVolumen

Lista los volúmenes docker de nuestro equipo

- docker volume ls

Ver detalles de un volumen docker

- docker inspect miVolumen

Terminal



```
$ docker volume create  
miVolumen  
miVolumen  
$ sudo ls  
/var/lib/docker/volumes/  
miVolumen
```

# Creación de contenedores con volúmenes

Asociamos los volúmenes a los contenedores en la creación de éstos

```
- docker run -d -v --name mywwwserver  
miweb:/usr/local/apache2/htdocs -p  
8080:80 httpd:2.4
```

De la siguiente manera creamos un archivo con contenido html en nuestro servidor:

```
- $ docker exec my-apache-app bash -c  
'echo "<h1>Hola</h1>" >  
/usr/local/apache2/htdocs/index.html'
```

Terminal



```
$ docker volume create miweb  
miweb  
$ docker run -d -v --name  
mywwwserver  
miweb:/usr/local/apache2/htdocs  
-p 8080:80 httpd:2.4  
531171c3da210001a7b4fb6b8273  
941231003f771d85f2b21eb5357e  
3edbcf2
```

# Asociar volúmenes Bind Mount

Creamos directorio en nuestro equipo que contendrá los archivos del servidor web:

- mkdir web
  - cd web
  - echo "<h1>Hola</h1>" > index.html
- docker

Creamos el contenedor:

- docker run -d --name my-apache-app -v /home/usuario/web:/usr/local/apache2/htdocs -p 8080:80 httpd:2.4



06

# Redes en Docker



# Tipos de redes en Docker

- **Red bridge:** Nos permite que los contenedores estén conectados a una red privada, con un direccionamiento privado conectado a la Host Docker mediante un Linux Bridge.
  - Nos permiten aislar los contenedores del acceso exterior.
  - Los contenedores conectados a un red bridge tienen acceso a internet por medio de una regla SNAT.
  - Usamos el parámetro `-p` en docker run para exponer algún puerto. Se crea una regla DNAT para tener acceso al puerto.
- **Red host:** No tiene dirección IP propia, sino es como si tuviera la dirección IP del Host Docker. Por lo tanto, los puertos son accesibles directamente desde el Host Docker.
- **Red none:** La red none no configurará ninguna IP para el contenedor y no tiene acceso a la red externa ni a otros contenedores. Tiene la dirección loopback y se puede usar para ejecutar trabajos por lotes.

# Redes bridge

Existen dos tipos de redes bridge:

- La **red bridge creada por defecto** por Docker para que de forma predeterminada los contenedores tengan conectividad.
- Y las redes bridge **definidas por el usuario**.

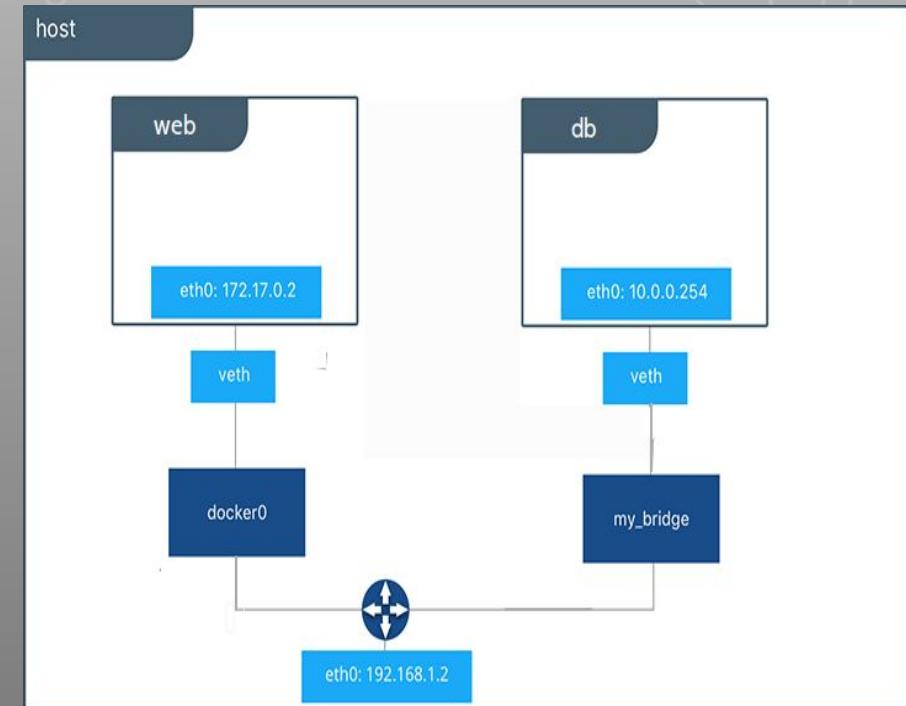
## Red bridge por defecto

Por defecto los contenedores que creamos se conectan a la red de tipo bridge llamada bridge.

- Se crea en el Host Docker un Linux Bridge llamado docker0.
- El direccionamiento de esta red es 172.17.0.0/16.

# Red bridge definida por el usuario

- Permite aislar diferentes contenedores o grupos de contenedores en distintas redes Docker.
- Proporcionan **DNS**.
- Más control sobre la configuración de las redes y mejor gestión de aislamiento de los contenedores.
- Tipo que deben tener contenedores en producción.



Fuente de la imagen: <https://iesgn.github.io/>



Financiado por  
la Unión Europea  
NextGenerationEU



# Gestión de redes bridge definidas por el usuario

Crear una red definida por el usuario

- **docker network create -d bridge red1**

Listar todas las redes

- **docker network ls**

Mostrar detalles de una red

- **docker network inspect red1**

Eliminar red

- **docker rm red1**

# Creación avanzada y uso de redes

Creamos red especificando rango de ips, puerta de enlace, etc.

- **docker network create --subnet 192.168.0.0/24 --gateway 192.168.0.100 red2**

- Creación de contenedor especificando red y configuración:

```
docker run -it --name contenedor  
--network red2 --ip 192.168.0.10  
--add-host=testing.example.com:192.1  
68.0.20 --dns 1.1.1.1 --hostname  
servidor1 alpine
```



07

# Escenarios multicontenedor



- datos.

JUNTA DE EXTREMADURA



Plan de  
Recuperación,  
Transformación  
y Resiliencia

Fuente de la imagen: <https://iesgn.github.io/>



Financiado por  
la Unión Europea  
NextGenerationEU

docker

N  
- docker  
- |

JUNTA DE EXTREMADURA

Terminal



\$  
[



Plan de  
Recuperación,  
Transformación  
y Resiliencia



Financiado por  
la Unión Europea  
NextGenerationEU



MINISTERIO  
DE EDUCACIÓN,  
FORMACIÓN PROFESIONAL  
Y DEPORTES



docker

N  
- docker  
- |

JUNTA DE EXTREMADURA

Terminal



\$  
[



Plan de  
Recuperación,  
Transformación  
y Resiliencia



Financiado por  
la Unión Europea  
NextGenerationEU



MINISTERIO  
DE EDUCACIÓN,  
FORMACIÓN PROFESIONAL  
Y DEPORTES





08

# Creación de imágenes en Docker



docker

N  
- docker  
- |

JUNTA DE EXTREMADURA

Terminal



\$  
[



Plan de  
Recuperación,  
Transformación  
y Resiliencia



Financiado por  
la Unión Europea  
NextGenerationEU



MINISTERIO  
DE EDUCACIÓN,  
FORMACIÓN PROFESIONAL  
Y DEPORTES



# Two columns

## Mercury

Mercury is the closest planet to the Sun and the smallest one in the Solar System—it's only a bit larger than the Moon. The planet's name has nothing to do with the liquid metal, since Mercury was named after the Roman messenger god

## Venus

Venus has a beautiful name and is the second planet from the Sun. It's terribly hot—even hotter than Mercury—and its atmosphere is extremely poisonous. It's the second-brightest natural object in the night sky

# One column



# Contents of this template

You can delete this slide when you're done editing the presentation

Fonts	To view this template correctly in PowerPoint, download and install the fonts we used
Used and alternative resources	An assortment of graphic resources that are suitable for use in this presentation
Thanks slide	You must keep it so that proper credits for our design are given
Colors	All the colors used in this presentation
Icons and infographic resources	These can be used in the template, and their size and color can be edited
Editable presentation theme	You can edit the master slides easily. For more info, click <a href="#">here</a>

# EJEMPLO GRÁFICO COLUMNAS



**PROYECTO PRIMARIO 1  
(PP1)**



**PROYECTO PRIMARIO 2  
(PP2)**



**PROYECTO PRIMARIO 3  
(PP3)**



**PROYECTO PRIMARIO 4  
(PP4)**



**PROYECTO PRIMARIO 5  
(PP5)**

Para modificar el gráfico asociado haz [click en el enlace](#)

# Three columns



## Jupiter

Jupiter is a gas giant and the biggest planet in the Solar System. It's the fourth-brightest object in the night sky and it was named after the Roman god of the skies



## Venus

Venus has a beautiful name and is the second planet from the Sun. It's hot and its atmosphere is poisonous. It's the second-brightest natural object in the night sky



## Saturn

Saturn is a gas giant and has several rings and it's composed mostly of hydrogen and helium. It was named after the Roman god of wealth and agriculture

# Four columns

## Mars

Despite being red, mars is actually a cold place. It's full of iron oxide dust, which gives the planet its reddish cast, and it's made of basalt



## Jupiter

Jupiter is a gas giant and the biggest planet in the solar system. It's the fourth-brightest object in the night sky and named after a roman god



## Venus

Venus has a beautiful name and is the second planet from the sun. It's terribly hot, even hotter than mercury. It has a toxic atmosphere



## Saturn

Saturn is a gas giant and has several rings. It's composed mostly of hydrogen and helium. It has beautiful rings



# Six columns



## Mars

Despite being red, Mars is actually a cold place



## Venus

Venus has a beautiful name and is terribly hot



## Neptune

Neptune is the farthest planet from the Sun



## Earth

Earth is the third planet from the Sun



## Saturn

Saturn is a gas giant and has several rings



## Jupiter

Jupiter is a gas giant and the biggest planet



**9h 55m 23s**

Jupiter's rotation period

**386,000 km**

Distance between Earth and the Moon

**333,000**

The Sun's mass compared to Earth's

# Ejemplo de Tabla

Proyectos Primarios	Descripción	Prioridad
PP1	Earth is the third planet from the Sun	
PP2	Saturn is a gas giant and has several rings	
PP3	Jupiter is a gas giant and the biggest planet	
PP4	Despite being red, Mars is actually a cold place	
PP5	Venus has a beautiful name and is terribly hot	

# Timeline

2XXX

Despite being red, Mars  
is actually a cold place



2XXX

Earth is the only planet  
known to harbor life



2XXX

Venus is the second  
planet from the Sun



2XXX

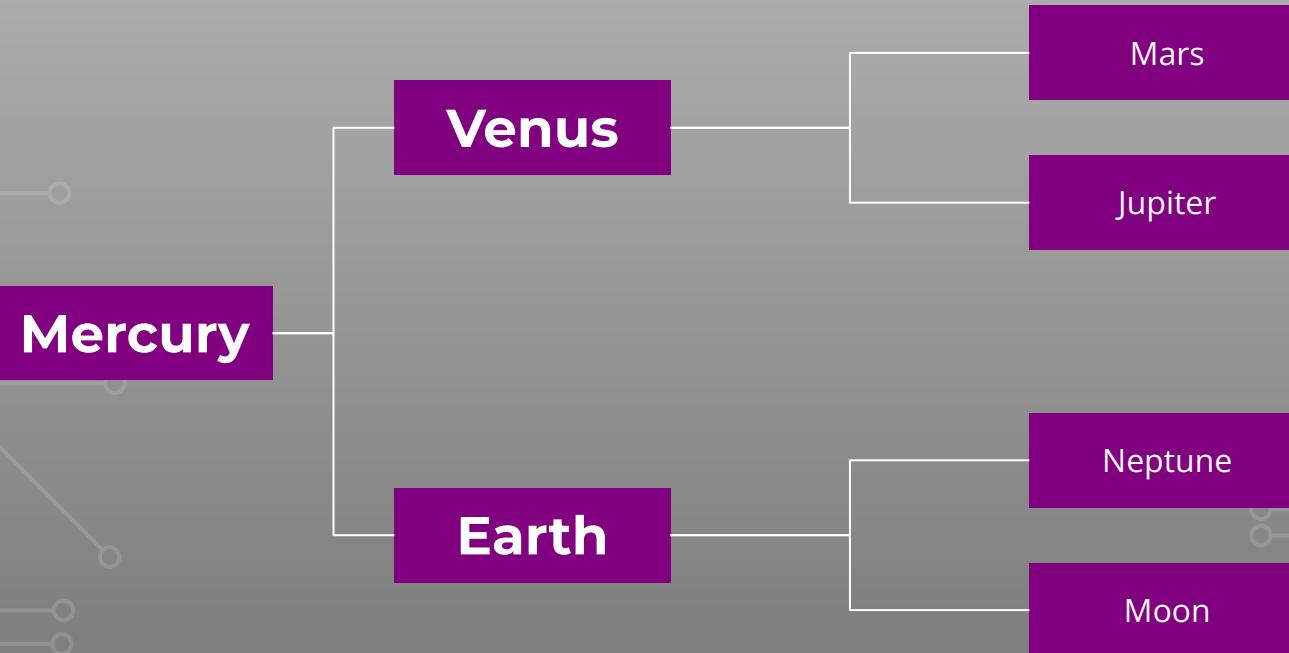
Saturn is one of the  
planets with rings



2XXX

Neptune is the farthest  
planet from the Sun

# Organizational chart



# Roadmap infographic

Task	Week 1	Week 2	Week 3	Week 4
Mercury				
Mars				
Saturn				
Venus				
Jupiter				
Neptune				

# Photo showcase

You can replace the images on the screen with your own work. Just right-click on one and select "Replace image"



# 750.000€

Destinado al Proyecto de Excelencia (2022-2024)

# Bibliografía y Webgrafía

- [Curso Docker Jose Domingo Muñoz](#)
- Presentación "Escalado de Servidores – Virtualización – Contenedores" de *Rafael López García*
- Presentación de "Implantación de sistemas seguros de despliegue de software" de *Carlos Nuñez*



# Gracias!

¿Alguna pregunta?



[informatica.iesvalledeljerteplasencia.es](http://informatica.iesvalledeljerteplasencia.es)



[coordinacion.cenfp@iesvp.es](mailto:coordinacion.cenfp@iesvp.es)



C/ Pedro y Francisco González, s/n  
10600, Plasencia (Cáceres)



927 01 77 74

