



## 3.2 – Contenedores y Orquestadores

# Tema 1 - Docker



Universidad  
Rey Juan Carlos

Micael Gallego  
micael.gallego@urjc.es  
@micael\_gallego



# Docker

- Introducción
- Volúmenes
- Configuración de contenedores
- Aplicaciones de consola
- Redes
- Creación de imágenes
- Herramientas para la creación de imágenes
- Desarrollo con contenedores

# Docker

- Introducción
- Volúmenes
- Configuración de contenedores
- Aplicaciones de consola
- Redes
- Creación de imágenes
- Herramientas para la creación de imágenes
- Desarrollo con contenedores

# Docker

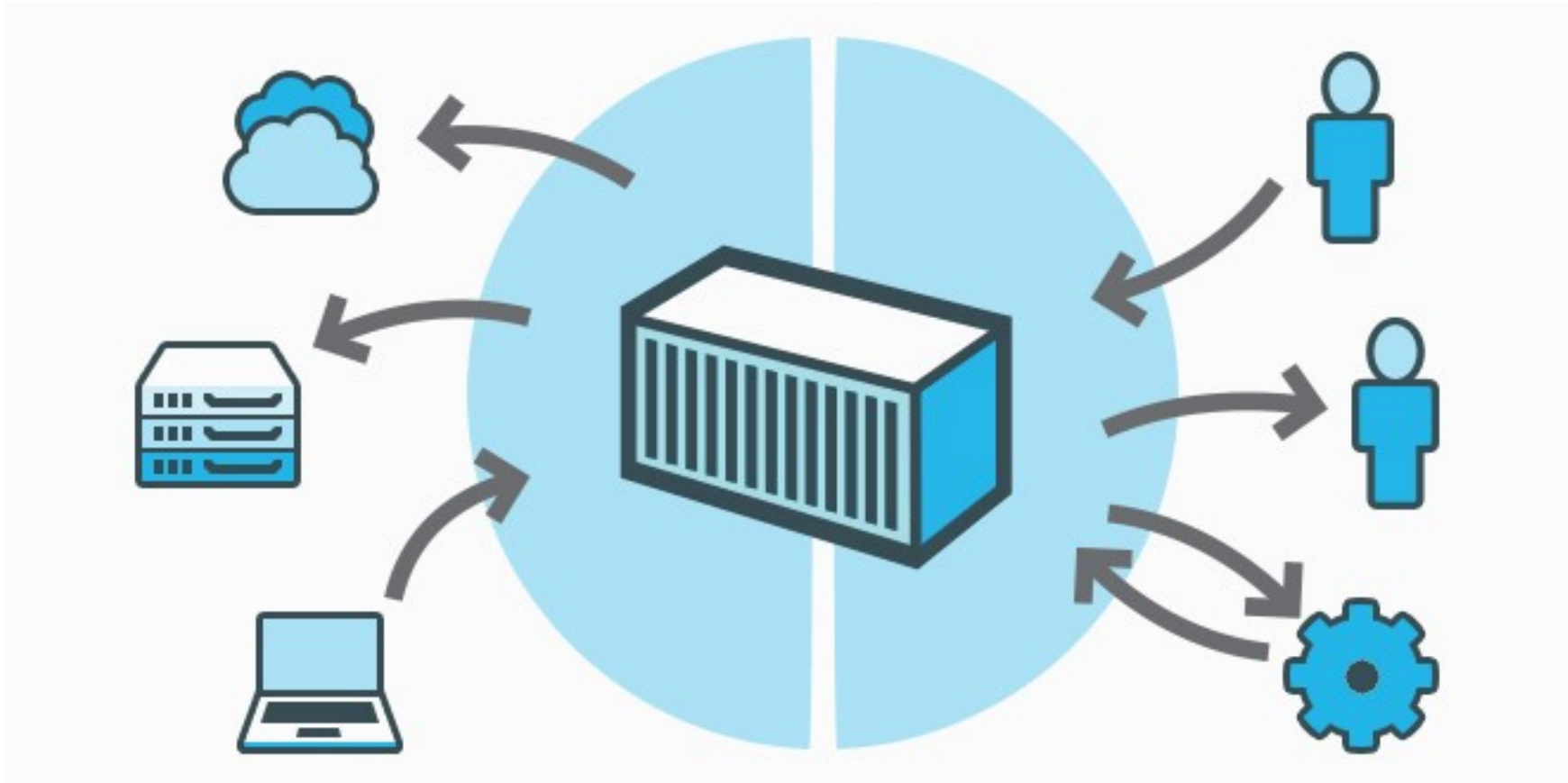


Los **contenedores Docker** permiten empaquetar, distribuir y ejecutar servicios de red, con un formato **estándar**

# Docker



# Docker



# Docker



- Es la tecnología de contenedores **más popular** (aunque existen otras tecnologías de contenedores)
- Muy utilizada en sistemas **linux**, aunque dispone de herramientas para **desarrolladores en windows y mac**
- Con **repositorio de imágenes** (DockerHub) con imágenes públicas de contenedores
- Creada en **2013**

<https://www.docker.com/>

- ¿Qué son los contenedores Docker?
  - Son aplicaciones empaquetadas con **todas sus dependencias**
  - Se pueden ejecutar en **cualquier sistema operativo**
    - En linux de forma **óptima**
    - En windows y mac con **virtualización ligera**
  - Se **descargan de forma automática** si no están disponibles en el sistema
  - Por defecto están aisladas del host (mayor seguridad)
  - Sólo es necesario tener instalado **Docker**

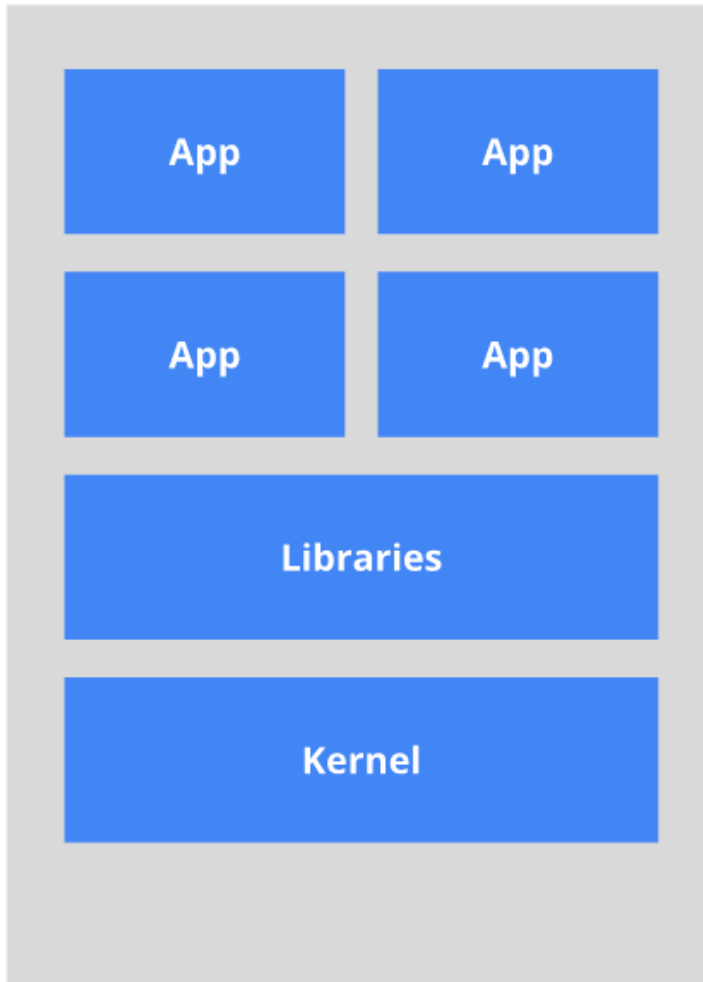


- **Formato de distribución y ejecución de servicios en linux**
  - Cada sistema linux tiene su **propio** sistema de **distribución y ejecución de servicios**
  - Los servicios **comparten recursos del servidor** sin ningún tipo de aislamiento entre ellas
  - Un **servicio** depende de las **versiones concretas de librerías** instaladas
  - Pueden aparecer problemas cuando varios servicios necesitan **versiones diferentes de las mismas librerías**

- **Formato de distribución y ejecución de servicios con Docker**
  - Los contenedores son un nuevo estándar de **empaquetado, distribución y ejecución de servicios en linux**
  - Cada paquete contiene el **binario del servicio** y todas las **librerías y dependencias** para que ese servicio se pueda ejecutar en un **kernel linux**
  - Se prefiere la potencial **duplicación de librerías** frente a los potenciales **problemas de compatibilidad** entre servicios

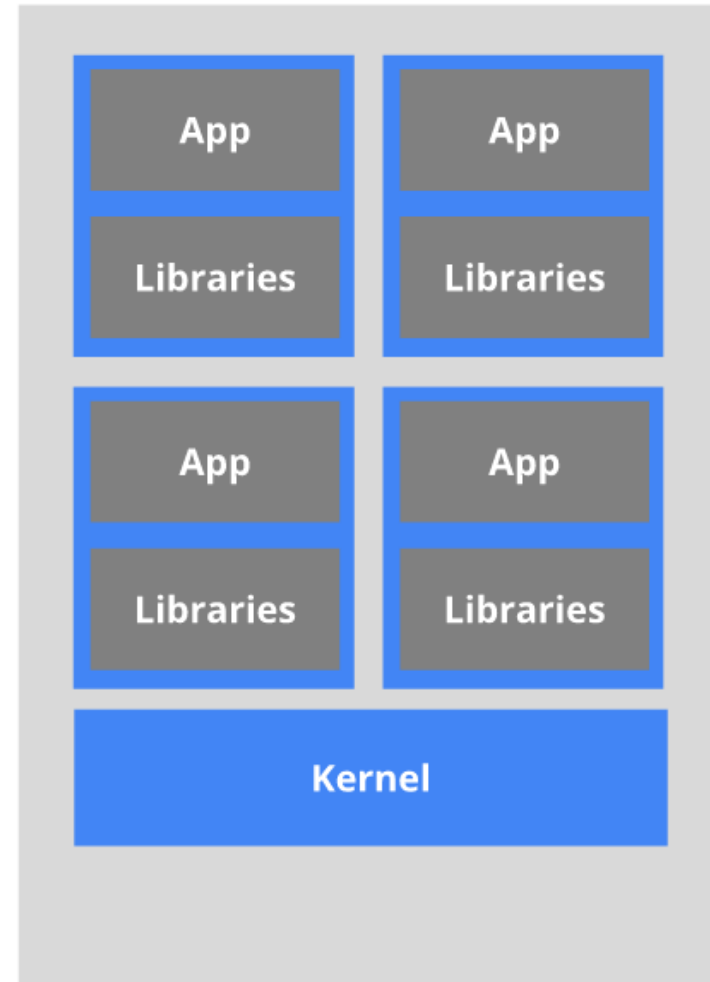
# Docker

**The old way:** Applications on host



*Heavyweight, non-portable  
Relies on OS package manager*

**The new way:** Deploy containers



*Small and fast, portable  
Uses OS-level virtualization*

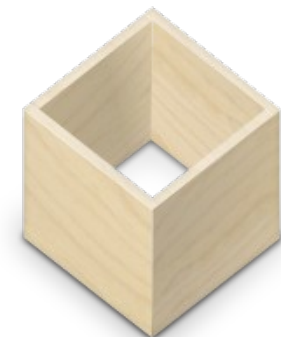
- **Tipos de aplicaciones:**
  - Aplicaciones **de red:**
    - Web, bbdd, colas de mensajes, cachés, etc.
  - Aplicaciones **de línea de comandos:**
    - Compiladores, generadores de sitios web, conversores de vídeo, generadores de informes...

# Docker

- Tipos de aplicaciones:

- Aplicaciones **gráficas**:

- Es posible pero no está diseñado para ello
    - Alternativas en linux



FLATPAK

<https://snapcraft.io/>

<https://flatpak.org/>

- **Sistemas operativos soportados**
  - **Contenedores linux**
    - Más usados y más maduros
    - En linux se ejecutan directamente por el kernel
    - En win y mac se ejecutan en máquinas virtuales ligeras gestionadas por docker
  - **Contenedores windows**
    - Menos usados y menos maduros
    - Sólo se pueden ejecutar en windows server

# ¿Cómo funciona Docker?

- Ejecuta procesos linux de forma aislada del resto con **namespaces** y **cgroups**
- Permiten definir un sistema de ficheros “virtual” específico para el proceso
- Cuando se ejecuta el servicio tiene un entorno similar al que tendría si estuviera en su **propia máquina**
- Se tienen ventajas similares a las **máquinas virtuales** pero de forma mucho **más eficiente**

# ¿Cómo funciona Docker?

- Los **contenedores** son una tecnología que ofrece unas **ventajas similares** a las VMs pero **aprovechando** mejor los recursos:
  - Los contenedores tardan **milisegundos** en arrancar
  - Consumen únicamente la **memoria** que necesita la app ejecutada en el contenedor.
    - Una VMs **reserva la memoria completa** y es usada por el sistema operativo huésped (*guest*) y la aplicación

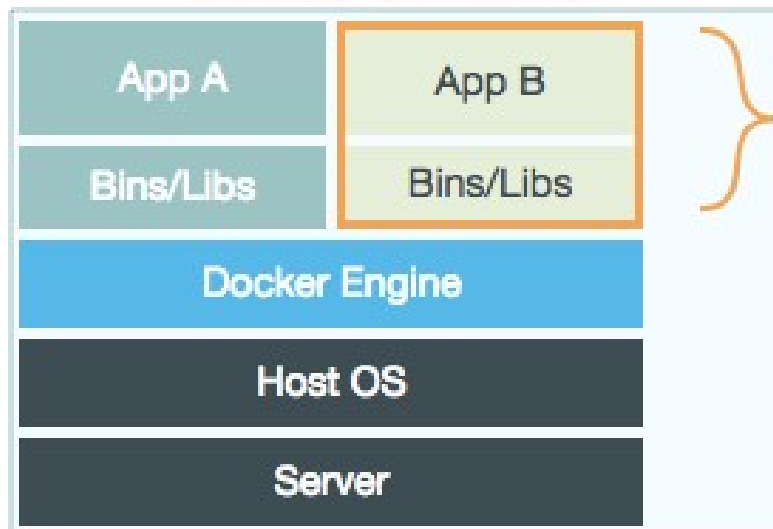


# ¿Cómo funciona Docker?

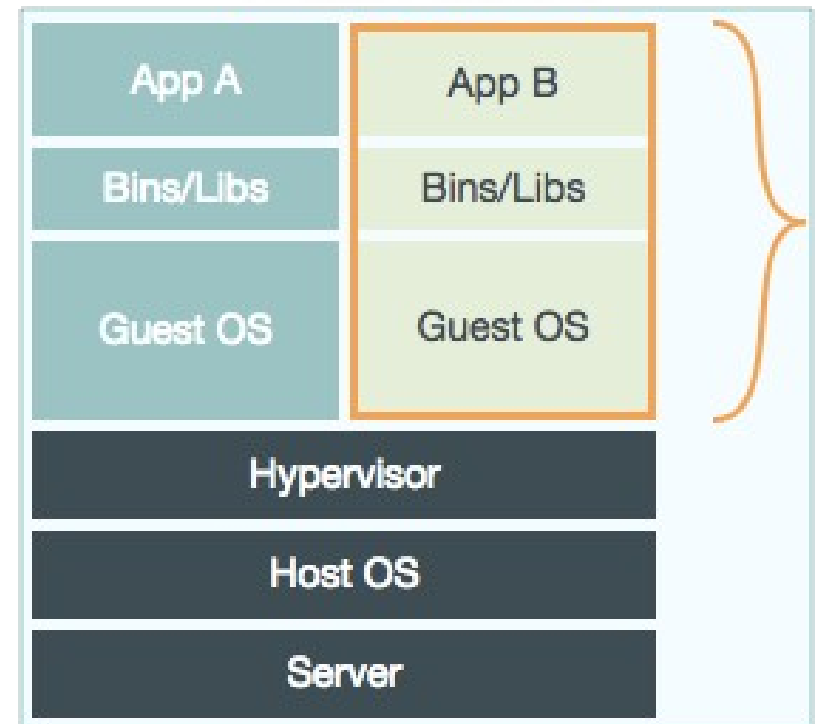
- ¿Por qué son tan eficientes los contenedores?
  - Para ejecutar un contenedor no se necesita **hypervisor** porque **no se ejecuta** un sistema operativo invitado y no hay que simular HW
  - Un contenedor es un **paquete** que contiene una **app** y todo el sw necesario para que se ejecute (python, Java, gcc, libs....)
  - El contenedor es ejecutado directamente por el **kernel del sistema operativo** como si fuera una aplicación normal pero de forma **aislada del resto**

# ¿Cómo funciona Docker?

Docker Container



Virtual Machine

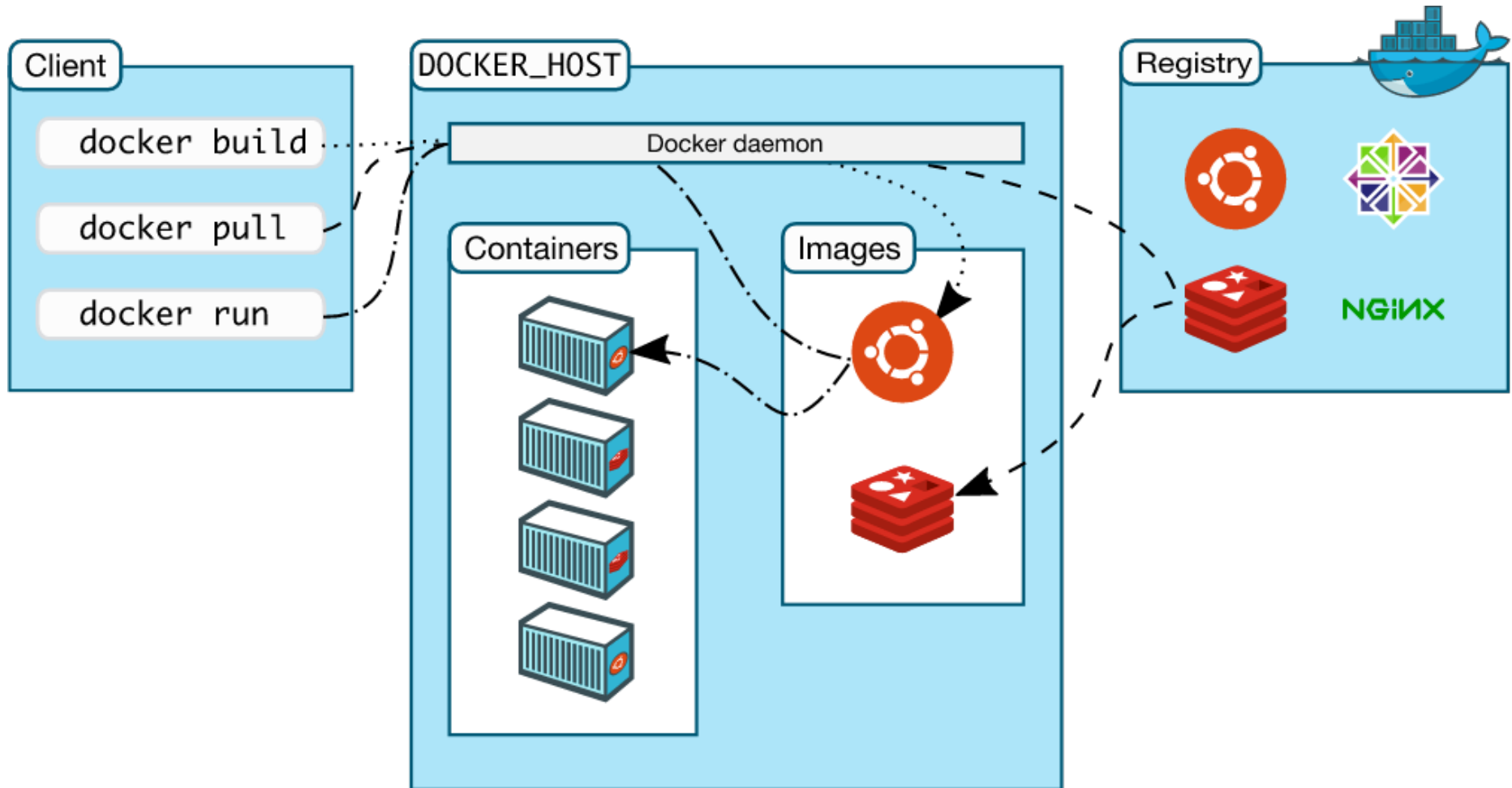


# ¿Cómo funciona Docker?

## Principales diferencias

Máquinas Virtuales	Contenedores
Más pesadas	Más ligeras
Varios procesos	Optimizadas para un único proceso (aunque pueden tener varios)
Conexión por ssh (aunque esté en local)	Acceso directo al contenedor
Más seguridad porque están más aisladas del host	Potencialmente menor seguridad porque se ejecutan como procesos en el host

# Conceptos Docker



# Conceptos Docker

- **Imagen docker**

- Ficheros a los que tendrá acceso el contenedor cuando se ejecute.
  - Herramientas/librerías de una distribución linux menos el kernel (**ubuntu, alpine**)
  - Runtime de ejecución (**Java**)
  - La aplicación en sí (**webapp.jar**)

# Conceptos Docker

- **Imagen docker**

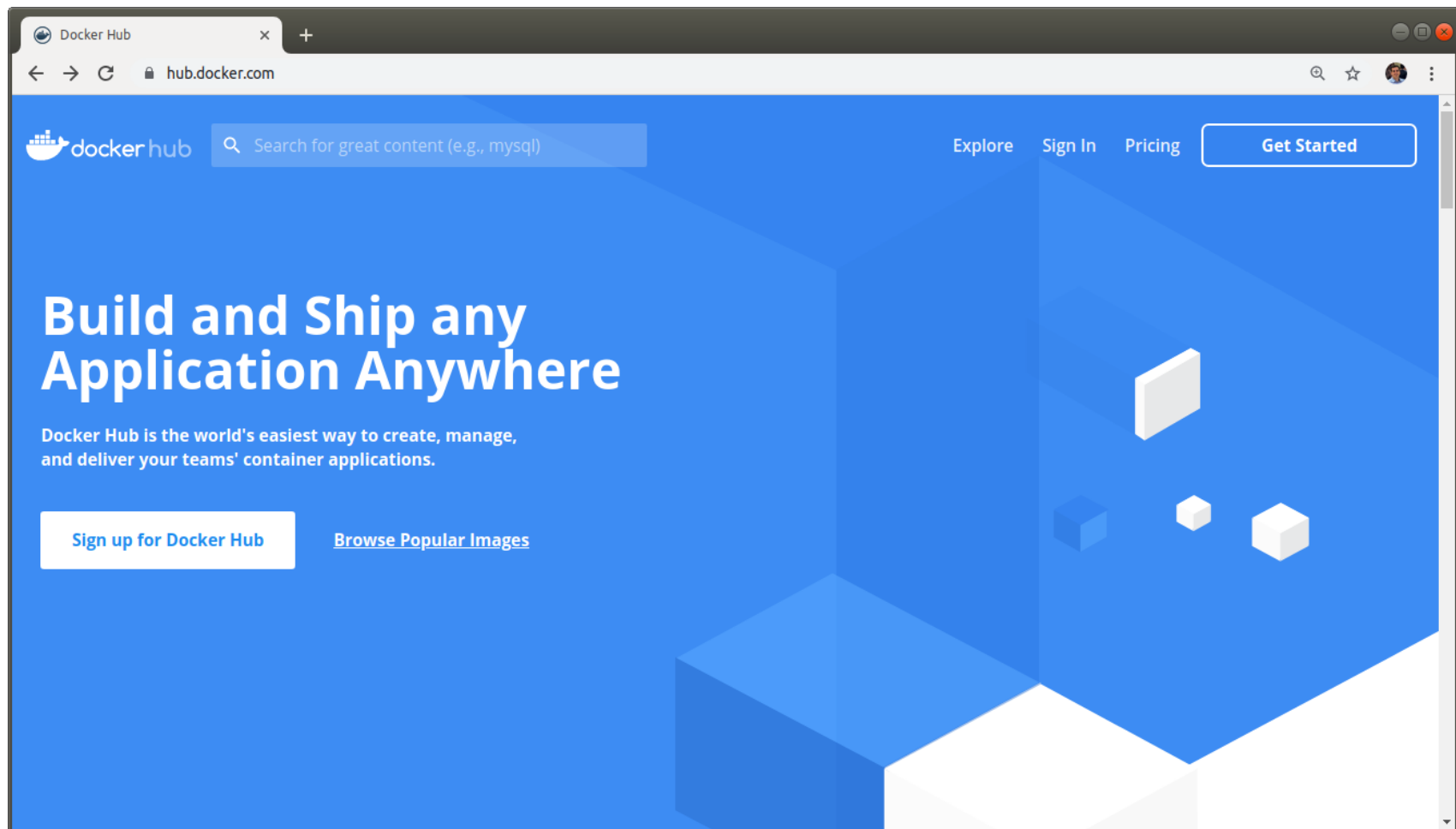
- Un contenedor siempre se inicia desde una **imagen**
- Si se quiere arrancar un contenedor partiendo de una imagen que no está disponible, se **descarga automáticamente** de Internet

# Conceptos Docker

- **Docker Registry**
  - **Servicio remoto** para subir y descargar imágenes
  - Puede guardar varias “versiones” (**tags**) de la misma imagen
  - Las diferentes versiones de una misma imagen se almacenan en un **repositorio (mysql, drupal...)**
  - **Docker Hub** es un registro público y gratuito gestionado por Docker Inc.
  - Puedes instalar un **registro privado**

# Conceptos Docker

- Docker Hub






# Conceptos Docker

- Docker Hub: Algunos repositorios oficiales



ubuntu  The Official Ubuntu base image



WordPress is a free and open source blogging tool and a content management system



Popular open-source relational database management system



Document-oriented NoSQL database



Official CentOS base image



High performance reverse proxy server

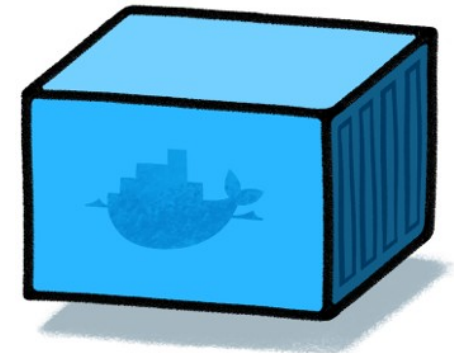


Relational database management system



Node.js is a platform for scalable server-side and networking applications

# Conceptos Docker



- **Contenedor Docker**
  - Representa la **aplicación en ejecución**
  - Un contenedor se crea desde **una imagen**
  - Si la aplicación escribe un fichero, el fichero queda dentro del contenedor, no se **modifica la imagen**
  - Los contenedores se pueden **arrancar, pausar y parar**
  - Puede haber **varios contenedores ejecutandose** a la vez partiendo desde la **misma imagen**

# Conceptos Docker

- **Docker Engine**
  - **Proceso encargado** de gestionar docker
  - Gestiona las **imágenes** (descarga, creación, subida, etc...)
  - Gestiona los **contenedores** (arranque, parada, etc..)
  - Habitualmente se controla desde el **cliente docker** por **línea de comandos** (aunque también se puede controlar por **API REST**)

# Conceptos Docker

- **Docker client**
  - Herramienta por línea de comandos (*Command line interface*, CLI) para controlar las imágenes y los contenedores

```
$ docker <params>
```

# Instalación de Docker

- **Linux:**
  - Docker Engine
    - Ejecución nativa
    - Funcionalidad completa
  - Docker Desktop for Linux
    - Usa virtualización ligera
    - Funcionalidad algo más limitada
- **Mac**
  - Docker Desktop for Mac
    - Usa virtualización ligera para ejecutar el kernel de linux
    - Funcionalidad algo más limitada que linux
- **Windows**
  - Docker Desktop for Windows
    - Usa virtualización ligera para ejecutar el kernel de linux
    - Funcionalidad algo más limitada que linux

# Instalación de Docker

## •Windows:

- Microsoft Windows 10 y 11 Home, Education, Enterprise y Pro:  
<https://docs.docker.com/desktop/install/windows-install/>

## •Linux:

- Ubuntu: <https://docs.docker.com/engine/install/ubuntu/>
- Fedora: <https://docs.docker.com/engine/install/fedora/>
- Debian: <https://docs.docker.com/engine/install/debian/>
- CentOS: <https://docs.docker.com/engine/install/centos/>
- Docker for Desktop (Ubuntu, Debian y Fedora):  
<https://docs.docker.com/desktop/install/linux-install/>

## •Mac:


- Apple macOS Big Sur (11) o superior con Intel o Apple Silicon:  
<https://docs.docker.com/desktop/install/mac-install/>

# Ejecución de contenedores

## Ejecutar “hello-world” en un contendor

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
03f4658f8b78: Pull complete
a3ed95caeb02: Pull complete
Digest:
sha256:8be990ef2aeb16dbcb9271ddfe2610fa6658d13f6dfb8bc72074c
c1ca36966a7
Status: Downloaded newer image for hello-world:latest

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

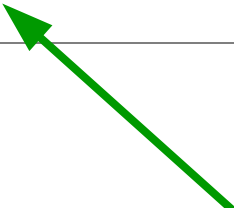


La primera vez la imagen se descarga

# Ejecución de contenedores

## Ejecutar “hello-world” por segunda vez

```
$ docker run hello-world  
Hello from Docker.  
This message shows that your installation appears to be  
working correctly.  
...
```



La segunda vez se usa  
la vez la imagen se  
descarga



# Ejecución de contenedores

## Inspeccionar los contenedores existentes

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a6a9d46d0b2f	alpine	"echo 'hello'"	6 minutes ago	Exited (0)	6 minutes ago	lonely_kilby
ff0a5c3750b9	alpine	"ls -l"	8 minutes ago	Exited (0)	8 minutes ago	elated_ramanujan
c317d0a9e3d2	hello-world	"/hello"	34 seconds ago	Exited (0)	34 seconds ago	stupefied_mcclintock

Muestra los contenedores del sistema.  
 Todos ellos tienen el estado STATUS Exited. Estos  
 contenedores no se están ejecutando  
 (pero consumen espacio en disco)

# Imágenes docker

- Para ejecutar un contenedor es necesario tener una **imagen** en la máquina
- Las imágenes se descargan de un docker registry (**registro**)
- Cada registro tiene un repositorio por cada imagen con múltiples versiones (**tags**)
- **DockerHub** es un registro gratuito en el que cualquiera puede subir imágenes públicas

# Imágenes docker

- **Imágenes oficiales vs de usuario**
  - **Oficiales (nombre)**
    - Creadas por compañías o comunidades de confianza
  - **De usuario (usuario/nombre)**
    - Cualquier usuario puede crear una cuenta y subir sus propias imágenes

# Imágenes docker

## Inspección de las imágenes descargadas

```
$ docker image ls
```

```
$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
sequence/static-site	latest	92a386b6e686	2 hours ago	190.5 MB
nginx	latest	af4b3d7d5401	3 hours ago	190.5 MB
python	2.7	1c32174fd534	14 hours ago	676.8 MB
postgres	9.4	88d845ac7a88	14 hours ago	263.6 MB
containous/traefik	latest	27b4e0c6b2fd	4 days ago	20.75 MB
...				

# Imágenes docker

## • Tags

```
$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
sequence/static-site	latest	92a386b6e686	2 hours ago	190.5 MB
nginx	latest	af4b3d7d5401	3 hours ago	190.5 MB
python	2.7	1c32174fd534	14 hours ago	676.8 MB
postgres	9.4	88d845ac7a88	14 hours ago	263.6 MB
Containous/traefik	latest	27b4e0c6b2fd	4 days ago	20.75 MB
...				

```
$ docker run hello-world:linux
```

tag

- El “tag” de una imagen es como su **versión**
- El nombre está inspirado en los tags de git. **Es una etiqueta**
- “**latest**” es la versión por defecto que se descarga si no se especifica versión. Normalmente apunta a la **última versión estable** de la imagen

# Servicios de red

- Servidor web en un contenedor

```
docker run --name static-site \  
  -e AUTHOR="Your Name" -d \  
  -p 9000:80 sequence/static-site
```

# Servicios de red

- Servidor web en un contenedor

```
docker run --name static-site \
  -e AUTHOR="Your Name" -d \
  -p 9000:80 sequence/static-site
```

**--name static-site**

Nombre del contenedor

# Servicios de red

- Servidor web en un contenedor

```
docker run --name static-site \  
  -e AUTHOR="Your Name" -d \  
  -p 9000:80 sequence/static-site
```

**-e AUTHOR="Your Name"**

Pasar variables de entorno a la aplicación  
que se ejecuta en el contenedor



# Servicios de red

- Servidor web en un contenedor

```
docker run --name static-site \  
  -e AUTHOR="Your Name" -d \  
  -p 9000:80 sequence/static-site
```

**-d**

Ejecuta el contenedor en segundo plano  
(no bloquea la shell durante la ejecución)

# Servicios de red

- Servidor web en un contenedor

```
docker run --name static-site \  
  -e AUTHOR="Your Name" -d \  
  -p 9000:80 sequence/static-site
```

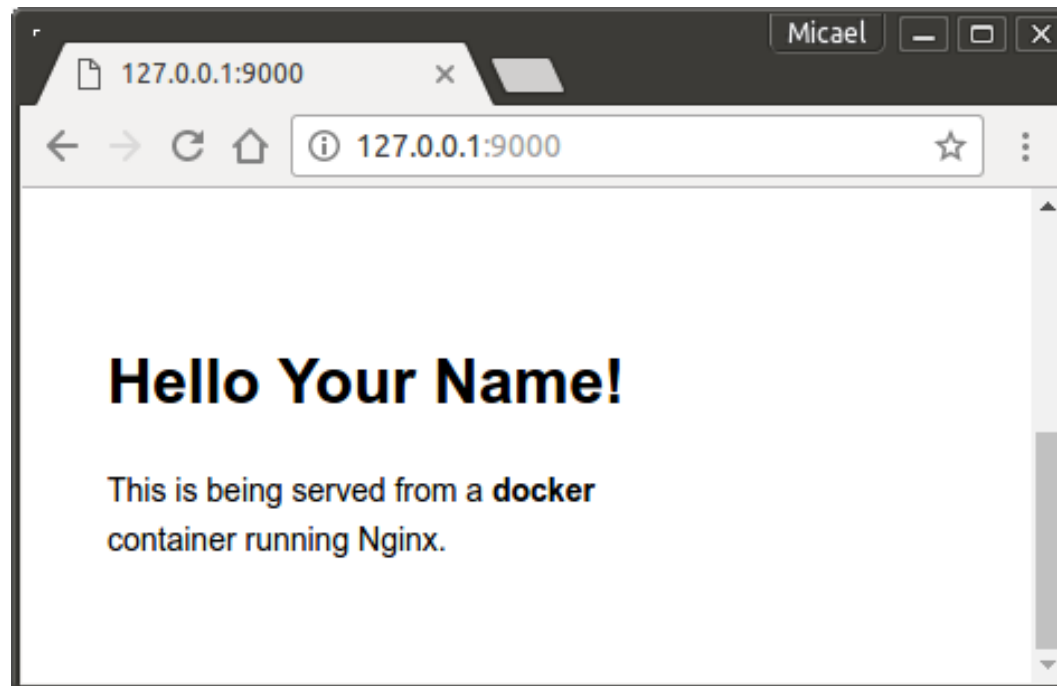
**-p 9000:80**

Conecta el puerto 9000 del host  
al puerto 80 del contenedor

# Servicios de red

- Usar el servicio

- Abre la URL <http://127.0.0.1:9000> en un browser accede al puerto 80 de la aplicación en el contenedor



# Gestión de contenedores

- Contenedores en ejecución

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             NAMES
CREATED            STATUS             PORTS              NAMES
a7a0e504ca3e       sequence/static-site  "/bin/sh -c 'cd /usr/"  28
seconds ago        Up 26 seconds
```

Container id es  
**a7a0e504ca3e**  
Este id se usa para  
referirte al contenedor

STATUS es UP

# Gestión de contenedores

- **Logs**

- Obtener la salida estándar de un contenedor

```
$ docker logs static-site
```

- Útil para contenedores arrancados en segundo plano

# Gestión de contenedores

- Parar y borrar contenedores

- Parar un contenedor en ejecución

```
$ docker stop static-site
```

- Borrar los ficheros del contenedor parado

```
$ docker rm static-site
```

# Gestión de contenedores

- Parar y borrar contenedores

- Parar y borrar en un comando

```
$ docker rm -f static-site
```

- Parar y borrar todos los contenedores

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

# Servicios de red

- Base de datos MySQL dockerizada

- Arranque contenedor:

```
$ docker run -d --name some-mysql \
  -e MYSQL_ROOT_PASSWORD=my-secret-pw \
  mysql:8.0
```

- Configuración con variables de entorno:

- **MYSQL\_DATABASE, MYSQL\_USER, MYSQL\_PASSWORD**

[https://hub.docker.com/\\_/mysql/](https://hub.docker.com/_/mysql/)



# Ejercicio 1

- **Ejecuta una web con Drupal en un contenedor docker**
  - Revisa la documentación de la página de DockerHub de Drupal
  - Accede al drupal desde un navegador web

# Docker

- Introducción
- **Volúmenes**
- Configuración de contenedores
- Aplicaciones de consola
- Redes
- Creación de imágenes
- Herramientas para la creación de imágenes
- Desarrollo con contenedores

# Volúmenes

- Los contenedores pueden acceder a **carpetas y ficheros del host**
- Ejemplos de uso:
  - Ficheros de **configuración**
  - Ficheros de **entrada y salida** (compiladores, servidores web...)
  - Carpetas para guardar los datos de una **BBDD**
- Por cada carpeta del host a la que se quiere acceder, se configura un **volumen**, indicando qué **carpeta del host** es visible en qué **carpeta del contendor**

# Volúmenes

- Configuración de volúmenes al ejecutar un contenedor

```
$ docker run -v <host_dir>:<container_dir> <image>
```

- Configurar la carpeta en la que se ejecuta el comando (variable de entorno PWD)

```
$ docker run -v "$PWD":<container_dir> <image>
```

# Volúmenes

## • Contenedor NGINX

- La imagen oficial del servidor web NGINX puede servir por http (web) ficheros del host

```
$ docker run -d -p 9000:80 \
  -v "$PWD":/usr/share/nginx/html:ro \
  nginx:1.23
```

- Carpeta del contenedor: **/usr/share/nginx/html**
- Modo de solo lectura (:ro)
- Abre el navegador en [http://127.0.0.1:9000/some\\_file](http://127.0.0.1:9000/some_file)
- “some\_file” es un fichero de la carpeta en la que se ejecuta el comando

[https://hub.docker.com/\\_/nginx/](https://hub.docker.com/_/nginx/)

# Volúmenes

## • Limitaciones Docker Desktop

- Docker Desktop para Windows (Hiper-V), Linux y Mac sólo permite montar por defecto algunos directorios como “C:\Users” en Windows, “/home/<user>” en Linux y “/Users” en Mac
- Los directorios permitidos se pueden configurar desde Docker Desktop en la pestaña File sharing



<https://docs.docker.com/desktop/settings/windows/#file-sharing>

<https://docs.docker.com/desktop/settings/linux/#file-sharing>

<https://docs.docker.com/desktop/settings/mac/#file-sharing>

# Docker

- Introducción
- Volúmenes
- **Configuración de contenedores**
- Aplicaciones de consola
- Redes
- Creación de imágenes
- Herramientas para la creación de imágenes
- Desarrollo con contenedores

# Configuración de contenedores

- Dependiendo de cómo se haya creado la imagen, un contenedor puede configurarse de diferentes formas cuando se ejecuta:
  - Sin configuración (ejecución por defecto)

```
$ docker run <imagen>
```

- Configuración con variables de entorno

```
$ docker run -e <NAME>=<value> <imagen>
```



# Configuración de contenedores

- Dependiendo de cómo se haya creado la imagen, un contenedor puede configurarse de diferentes formas cuando se ejecuta:
- **Parámetros del comando por defecto**

```
$ docker run <imagen> <params>
```

- **Comando y parámetros (cuando no hay comando por defecto)**

```
$ docker run <imagen> <comando> <params>
```

# Docker

- Introducción
- Volúmenes
- Configuración de contenedores
- **Aplicaciones de consola**
- Redes
- Creación de imágenes
- Herramientas para la creación de imágenes
- Desarrollo con contenedores

# Aplicaciones de consola

- Jekyll

- Jekyll es una herramienta que genera un sitio web partiendo de ficheros de texto (**Markdown**)
- Es el sistema que usa GitHub para sus páginas
- Jekyll se puede usar desde un **contenedor** sin tener que instalarlo en el host



# Aplicaciones de consola

## • Jekyll

- Descargar contenido de ejemplo

```
$ git clone https://github.com/pages-themes/minimal
$ cd minimal
```

- Ejecutar el contenedor para generar el sitio

```
$ docker run --rm -it -v "${PWD}":/srv/jekyll \
  jekyll/jekyll:4.2.2 jekyll build
```

- **jekyll build** es el comando del contenedor
- **--rm** borra el contenedor al terminar su ejecución
- El resultado se crea en la carpeta **\_site**

# Aplicaciones de consola

- **Jekyll**

- También podremos servir el sitio utilizando la misma imagen de **Jekyll**

```
$ docker run --rm -it -v "${PWD}":/srv/jekyll \  
  -p 4000:4000 jekyll/jekyll:4.2.2 jekyll serve
```

- **jekyll serve** es el comando del contenedor
- **--rm** borra el contenedor al terminar su ejecución
- Abrir <http://localhost:4000/> en el navegador web

# Aplicaciones de consola

- **Jekyll**

- Servir el contenido de la carpeta **\_site** con un servidor web

```
$ cd _site
$ docker run --rm -it -v "$PWD":/usr/share/nginx/html:ro \
  -p 9000:80 nginx:1.23
```

- Abrir <http://localhost:9000/> en el navegador web

[https://hub.docker.com/\\_/nginx/](https://hub.docker.com/_/nginx/)  
<https://hub.docker.com/r/jekyll/jekyll>

# Aplicaciones de consola

## • Jekyll

- Los ficheros generados tienen los **permisos del usuario** que se ejecuta dentro del contenedor
- Por defecto, los contenedores se ejecutan como **root**
- Algunos contenedores como el de **Jekyll** se ejecutan con los permisos **1000:1000**
- En linux el usuario que tiene los permisos **1000:1000** coincide con el primer usuario creado en el sistema

```
$ cat /etc/passwd | grep oscaroto
oscaroto:x:1000:1000:oscaroto,,,:/home/oscaroto:/usr/bin/zsh
```

# Ejercicio 2

- Genera el .jar de una aplicación con un contenedor docker
- Utiliza la aplicación “**application-java-enunciado**”
- Busca una imagen adecuada en Docker Hub (tiene que tener Maven y un JDK de Java)
- Monta las carpetas adecuadas (para que el compilador pueda acceder al fuente y para que pueda generar el binario)



# Imagen con Java

- ¿Qué imagen con Java utilizar?

# OpenJDK

- OpenJDK
  - En 2022 Red Hat descontinúa OpenJDK Project Builds

## Discontinued OpenJDK Project Builds

### What are these binaries?

Upstream binaries were built by Red Hat on their infrastructure and distributed by AdoptOpenJDK on behalf of the OpenJDK jdk8u and jdk11u community projects. The last releases produced by this method were 8u342 and 11.0.16 (July 2022 CPU update). This service is now provided by the [Adoptium](#) project, and users of upstream builds should now migrate to the equivalent [Eclipse Temurin](#) releases.

# Imagen con Java

- ¿Qué imagen con Java utilizar?
  - Eclipse Temurin
    - Continuación de OpenJDK
    - El objetivo de Eclipse Temurin es proporcionar los binarios de Java SE
    - Nace en Octubre 2021



**TEMURIN**  
by ADOPTIUM

# Una shell dentro del contenedor

- El uso principal de un contenedor es empaquetar aplicaciones (de consola o servicio de red)
- En ocasiones queremos ejecutar comandos de forma interactiva “**dentro del contenedor**”
- La mayoría de las imágenes tienen el binario de una consola (shell): **/bin/sh** o **/bin/bash** (dependiendo de la imagen)

# Una shell dentro del contenedor

- Shell en un contenedor con ubuntu

```
$ docker run -it ubuntu:20.04 /bin/sh
# ls
bin    dev    home  lib64 mnt    proc  run    srv    tmp    var
boot  etc    lib   media opt    root  sbin   sys    usr
# exit
```

- La opción **-it** se usa para que se conecte la terminal al proceso del contenedor de forma **interactiva**
- Usados de esta forma los contenedores se parecen a una **máquina virtual ligera** a la que se accede por **ssh**

# Una shell dentro del contenedor

- Shell en un contenedor con ubuntu

```
$ docker exec -it <container_name> /bin/sh
# ls
bin    dev    home   lib64  mnt    proc   run    srv    tmp    var
boot   etc    lib    media  opt    root   sbin   sys    usr
# exit
```

- Es posible ejecutar un **comando** dentro de un contenedor en ejecución
- Se suele ejecutar una shell para poder inspeccionar el **sistema de ficheros** del contenedor para **depurar** problemas

# Ejercicio 3

- **Usa Maven dockerizado del ejercicio 2 como si fuera una mini máquina virtual**
  - Ejecuta una shell en el contenedor
  - Ejecuta los comandos de compilación dentro de la shell cada vez que quieras compilar

# Docker

- Introducción
- Volúmenes
- Configuración de contenedores
- Aplicaciones de consola
- **Redes**
- Creación de imágenes
- Herramientas para la creación de imágenes
- Desarrollo con contenedores

# Redes en Docker

- Interfaces de red por defecto en Docker



bridge



host



none

- **Bridge:** Red por defecto al lanzar un contenedor, proporciona aislamiento al utilizar una red privada diferente a la red del Host. Permite la comunicación entre contenedores en la misma red pero sin poder utilizar el nombre de los contenedores
- **Host:** Esta red elimina el aislamiento entre el Host y los contenedores. Utiliza la red del Host y su IP, esto implica que los puertos del contenedor estarán directamente disponibles en el Host
- **None:** Esta red no permitirá la salida a internet del container ni la comunicación entre contenedores



# Redes en Docker

- Si creamos una red nueva los contenedores de esa red podrán **verse** utilizando su **nombre de servicio (DNS)**
- Los contenedores de la red por defecto (**bridge**) también pueden verse, pero solo utilizando las **IPs** de los servicios
- En **Linux** podremos conectarnos a los contenedores levantados utilizando su **IP**, no es necesario **bindear** los puertos. En **Windows y Mac** siempre será necesario bindear los puertos que queramos atacar desde fuera del contenedor

# Redes en Docker

- Ejecución de un servidor web sin **bindear** puertos

```
$ docker run --rm -it -v "$PWD":/usr/share/nginx/html:ro \
  --name nginx nginx:1.23
```

- Obtener la **IP** de un contenedor en ejecución

```
$ docker inspect -f \
  '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' \
  nginx
172.17.0.2
```

- Abrir <http://172.17.0.2/> en el navegador web



Esta funcionalidad solo está disponible en sistemas Linux

# Redes en Docker

- Listado de las redes disponibles

```
$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
7c2f22e31df5       bridge             bridge              local
77f9d63ab030       host               host                local
a7f76a78cc29       none              null                local
```

- Creación de una nueva red

```
$ docker network create some-network
```

- Conectar un container a una determinada red

```
$ docker run --network some-network alpine
```

# Redes en Docker

## • Conexión entre 2 contenedores

- Creamos una nueva red

```
$ docker network create drupal-network
```

- Ejecutamos un contenedor MySQL y lo conectamos en la red creada

```
$ docker run -d --name mysql --network drupal-network \
  -e MYSQL_DATABASE=drupal -e MYSQL_USER=user \
  -e MYSQL_PASSWORD=pass -e MYSQL_ROOT_PASSWORD=pass \
  mysql:8.0
```

- Ejecutamos un contenedor Drupal y lo conectamos en la red creada

```
$ docker run -d --network drupal-network --name drupal \
  -p 8080:80 drupal:10.0
```

# Redes en Docker

## Conectar Drupal con la base de datos

Utilizaremos los datos de usuario, contraseña y nombre de la base de datos definidos en el contenedor de “mysql”

*Database name: drupal*  
*Username: user*  
*Database password: pass*

### Database configuration

#### Database type \*

- ☒ MySQL, MariaDB, Percona Server, or equivalent  
☐ PostgreSQL  
☐ SQLite

#### Database name \*

drupal

#### Database username \*

user

#### Database password

....

#### ADVANCED OPTIONS

##### Host \*

mysql

##### Port number

3306

##### Table name prefix

If more than one application will be sharing this database, a unique table name prefix - such as *demo\_umami\_* - will prevent collisions.

Save and continue

# Redes en Docker

## Conectar Drupal con la base de datos

Utilizaremos el nombre del contenedor **"mysql"** para conectarnos con la base de datos

Esto es posible debido a que los servicios **"mysql"** y **"drupal"** están en la misma red y pueden utilizar la resolución de nombres proporcionada por Docker

### Database configuration

#### Database type \*

- ☒ MySQL, MariaDB, Percona Server, or equivalent  
☐ PostgreSQL  
☐ SQLite

#### Database name \*

drupal

#### Database username \*

user

#### Database password

....

#### ADVANCED OPTIONS

##### Host \*

mysql

##### Port number

3306

##### Table name prefix

If more than one application will be sharing this database, a unique table name prefix - such as *demo\_umami\_* - will prevent collisions.

Save and continue

# Docker

- Introducción
- Volúmenes
- Configuración de contenedores
- Aplicaciones de consola
- Redes
- **Creación de imágenes**
- Herramientas para la creación de imágenes
- Desarrollo con contenedores

# Dockerizar una aplicación

- Para dockerizar una aplicación hay que crear una imagen docker de la aplicación
- Crearemos una imagen con una **aplicación web** implementada en **Node**
- Descarga la web de ejemplo

```

v ejemplo-1-node
  > .devcontainer
  > views
  ◆ .gitignore
  🐳 cache.Dockerfile
  🐳 Dockerfile
  🐳 multistage.Dockerfile
  {} package-lock.json
  {} package.json
  JS server.js
  
```

```

$ git clone \
    https://github.com/MasterCloudApps/3.2.Contenedores-y-orquestadores
$ cd 3.2.Contenedores-y-orquestadores/docker/ejemplo-1-node
  
```



# Dockerizar una aplicación

- Contenido de la imagen docker:
  - Código fuente de la aplicación
  - Node
  - Librerías necesarias (express y mustache-express)
- Una vez creada la imagen, se puede **ejecutar la aplicación dockerizada**
- También se puede **publicar en DockerHub** (o cualquier otro registro) para compartirla

# Dockerizar una aplicación

- **Dockerfile**

- Fichero usado para describir el contenido de una imagen docker
- Contenido:
  - Imagen en la que se basará la nueva imagen
  - Comandos que añaden el software necesario a la imagen base
  - Ficheros de la aplicación para incluir en la imagen
  - Puertos abiertos para poder bindearlos al host
  - Comando por defecto a ejecutar al arrancar el contenedor

```
# Selecciona la imagen base
FROM node:lts-alpine

# Especificamos esta variable para la correcta ejecución de
las librerías en modo de producción
ENV NODE_ENV production

# Definimos el directorio de trabajo en /usr/src/app/
WORKDIR /usr/src/app/

# Copiamos los ficheros de la aplicación
COPY src /usr/src/app/src
COPY package.json /usr/src/app/

# Instalamos las dependencias que necesita la app
RUN npm install --only=production

# Indica el puerto que expone el contenedor
EXPOSE 5000

# Comando que se ejecuta cuando se arranque el contenedor
CMD ["node", "src/server.js"]
```

# Dockerizar una aplicación

## • Dockerfile

- **FROM:** Imagen base
- **RUN:** Ejecuta comandos para instalar y configurar el software de la imagen
- **COPY:** Copy ficheros desde la carpeta del Dockerfile
- **EXPOSE:** Define los puertos públicos
- **CMD:** Comando por defecto que se ejecuta al arrancar el contenedor

[https://docs.docker.com/engine/userguide/eng-image/dockerfile\\_best-practices/](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/)

# Dockerizar una aplicación

- Construir la imagen

- Se puede crear una imagen para que sea usada **únicamente en la máquina** que se ha creado
- Lo más habitual es crear una imagen para **subirla a un registro** de imágenes (como DockerHub)
  - Creamos una **cuenta en DockerHub**
  - **Conectamos** nuestra máquina a DockerHub

```
$ docker login
```

# Dockerizar una aplicación

- Construir la imagen

- En la carpeta del **Dockerfile** se ejecuta

```
$ docker build -t miusuario/webgatos .
```

- **miusuario** corresponde al usuario creado en DockerHub
- **webgatos** es el nombre del repositorio al que subir la imagen
- **.** es la ruta del Dockerfile

# Dockerizar una aplicación

- **Construir la imagen**

- Acciones ejecutadas:

- Se ejecuta un nuevo contenedor partiendo de la imagen base
- Se ejecutan los comandos (RUN y COPY) del Dockerfile en ese contenedor
- El resultado se empaqueta en el nuevo contenedor

# Dockerizar una aplicación

- Ejecutar la nueva imagen

```
$ docker run -p 5000:5000 miusuario/webgatos
* Running on http://localhost:5000/
(Press CTRL+C to quit)
```

- Abrir <http://localhost:5000/> en el navegador web



# Dockerizar una aplicación

- **Publicar la imagen**

```
$ docker push miusuario/webgatos
```

- La imagen se sube a DockerHub y se hace pública
- Cualquiera puede ejecutar un contenedor partiendo de esa imagen
- Se pueden instalar registros privados en una organización

# Dockerizar una aplicación

ejemplo-1-node

- **Caché de construcción por capas**

- Cambia la plantilla de la web en `src/views/index.html`
- Construye la imagen de nuevo

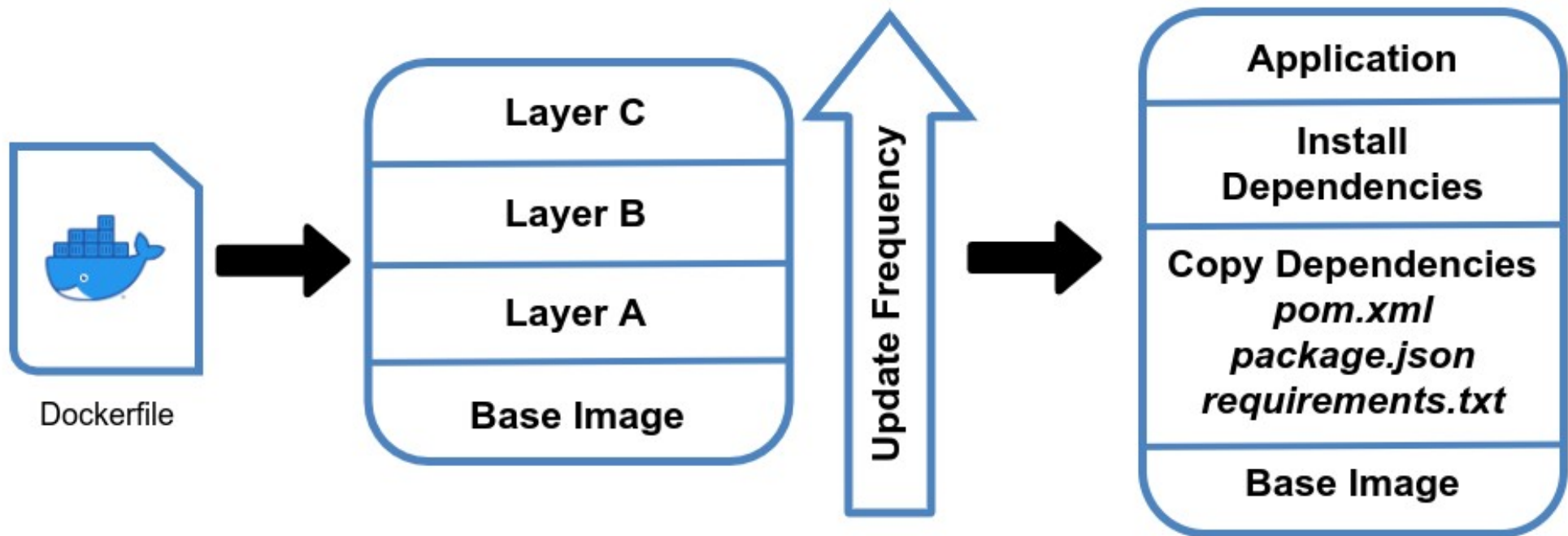
```
$ docker build -t miusuario/webgatos .
```

- Los pasos del Dockerfile que no han cambiado **NO se vuelven a ejecutar** (se reutilizan de la ejecución previa)
- Cada paso está en una **capa independiente**
- La nueva imagen se crea muy rápidamente

# Dockerizar una aplicación

- Buenas prácticas del Dockerfile
  - Aprovechamiento de caché de las capas
    - Las dependencias suelen cambiar poco, por eso se instalan antes del código (y quedan en una capa previa)
    - El código es lo que más cambia, por eso sus comandos van al final

# Dockerizar una aplicación



```
# Selecciona la imagen base
FROM node:lts-alpine

# Especificamos esta variable para la correcta ejecución de
las librerías en modo de producción
ENV NODE_ENV production

# Definimos el directorio de trabajo en /usr/src/app/
WORKDIR /usr/src/app/

# Copiamos los ficheros de la aplicación
COPY src /usr/src/app/src
COPY package.json /usr/src/app/

# Instalamos las dependencias que necesita la app
RUN npm install --only=production

# Indica el puerto que expone el contenedor
EXPOSE 5000

# Comando que se ejecuta cuando se arranque el contenedor
CMD ["node", "src/server.js"]
```

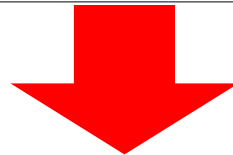
```
# Copiamos los ficheros de la aplicación
```

```
COPY src /usr/src/app/src
```

```
COPY package.json /usr/src/app/
```

```
# Instalamos las dependencias que necesita la app
```

```
RUN npm install --only=production
```



```
# Copiamos fichero de dependencias
```

```
COPY package.json /usr/src/app/
```

```
# Instalamos las dependencias que necesita la app
```

```
RUN npm install --only=production
```

```
# Copiamos el resto de ficheros de la aplicación
```

```
COPY src /usr/src/app/src
```

# Dockerizar una aplicación

- Buenas prácticas del Dockerfile

- Cada comando Dockerfile es una capa:

- Si un comando RUN graba ficheros y el siguiente comando los borra, los ficheros originales quedan en la imagen (en la capa)
- Se encadenan muchos comandos en un mismo RUN para limpiar los ficheros temporales

```
RUN apt-get update && apt-get install -y \
    curl \
    ruby1.9.1 \
    && rm -rf /var/lib/apt/lists/*
```

[https://docs.docker.com/develop/develop-images/dockerfile\\_best-practices/#run](https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#run)

# Ejercicio 4

- Crea una imagen docker con una aplicación web Java
- Utiliza la aplicación “**aplicacion-java-enunciado**”
- Basada en una imagen con Maven para poder compilar la aplicación en el proceso de construcción de la imagen



Existen **estrategias más convenientes** de empaquetar una aplicación Java en un contenedor Docker que veremos más adelante



# Dockerizar una aplicación compilada

- Dockerizar una aplicación con lenguaje de script es bastante sencillo, porque el **código fuente** se puede **ejecutar directamente**
- Cuando la aplicación está implementada con un lenguaje compilado, se realizan dos pasos:
  - 1) **Compilar** la aplicación (preferiblemente en un contenedor)
  - 2) **Empaquetar** la aplicación en un contenedor

# Dockerizar una aplicación compilada

- **Compilar una aplicación Java en un contenedor**
  - Descargar proyecto de ejemplo

```
$ git clone \
  https://github.com/MasterCloudApps/3.2.Contenedores-y-orquestadores
$ cd 3.2.Contenedores-y-orquestadores/docker/ejemplo-2
```

- Compilar y generar el fichero .jar

```
$ docker run --rm -it -v "$PWD":/app -w /app \
  maven:3.9.0-eclipse-temurin-17 mvn package
```

- “mvn package” es el comando de compilación y empaquetado
- -w configura el directorio de trabajo

# Dockerizar una aplicación compilada

- **Compilar una aplicación Java en un contenedor**
  - La aplicación compilada y empaquetada es un fichero .jar que se encuentra en la carpeta **target**
  - Para ejecutar ese fichero es necesario el **Java Runtime Environment (JRE)**, pero no es necesario un compilador ni otras herramientas de construcción como Maven
  - Se ejecuta con el comando

```
$ java -jar ./target/java-webapp-0.0.1.jar
```

# Dockerizar una aplicación compilada

- **Dockerizar la aplicación Java**
  - Hay que crear un nuevo contenedor con Java para poder ejecutar el .jar (No se necesita maven)
  - Hay que copiar el fichero .jar recién creado
  - Al arrancar el contenedor, se ejecuta

```
java -jar java-webapp-0.0.1.jar
```

# Dockerizar una aplicación compilada

- Dockerizar la aplicación Java

jar.Dockerfile

```
# Selecciona la imagen base
FROM eclipse-temurin:17-jdk

# Define el directorio de trabajo para el comando
WORKDIR /usr/src/app/

# Copia de la aplicación compilada
COPY target/*.jar /usr/src/app/

# Indica el puerto que expone el contenedor
EXPOSE 8080

# Comando que se ejecuta al hacer docker run
CMD [ "java", "-jar", "java-webapp-0.0.1.jar" ]
```

# Dockerizar una aplicación compilada

- Dockerizar la aplicación Java
  - Construir el contenedor

```
$ docker build -f jar.Dockerfile -t miusuario/java-webapp .
```

- Ejecutar el contenedor

```
$ docker run -it -p 5000:8080 miusuario/java-webapp
```

# Dockerizar una aplicación compilada

- **Multistage Dockerfile**

- Se han realizado dos pasos para dockerizar la aplicación
  - **Paso 1:** Compilar el código fuente y generar el binario usando un contenedor
  - **Paso 2:** Crear un contenedor con el binario generado
- Los **Multistage Dockerfiles** son ficheros Dockerfile que permiten definir varios pasos.
- Cada paso se ejecuta en su propio contenedor

<https://docs.docker.com/develop/develop-images/multistage-build/>

# Dockerizar una aplicación compilada

- Multistage Dockerfile

multistage.Dockerfile

```
# Imagen base para el contenedor de compilación
FROM maven:3.9.0-eclipse-temurin-17 as builder
WORKDIR /project
COPY /src /project/src
COPY pom.xml /project/
RUN mvn -B package -DskipTests

# Imagen base para el contenedor de la aplicación
FROM eclipse-temurin:17-jdk
WORKDIR /usr/src/app/
COPY --from=builder /project/target/*.jar /usr/src/app/
EXPOSE 8080
CMD [ "java", "-jar", "java-webapp-0.0.1.jar" ]
```

```
$ docker build -f multistage.Dockerfile -t miusuario/java-webapp2 .
```



# Dockerizar una aplicación compilada

ejemplo-1-node

- Multistage Dockerfile en una aplicación node

multistage.Dockerfile

```
# Imagen base para el contenedor de compilación
FROM node:lts-alpine as builder
WORKDIR /usr/src/app/
COPY package.json /usr/src/app/
RUN npm install --only=production

# Imagen base para el contenedor de la aplicación
FROM node:lts-alpine
ENV NODE_ENV production
WORKDIR /usr/src/app
COPY --from=builder /usr/src/app/node_modules
  /usr/src/app/node_modules
COPY src /usr/src/app/src
COPY package.json /usr/src/app/
EXPOSE 5000
CMD ["node", "src/server.js"]
```

# Dockerizar una aplicación compilada

- **Ventajas del Multistage Dockerfile**

- La ventaja de usar un Multistage Dockerfile es que con un **único comando** se puede compilar y dockerizar la aplicación
- El comando se puede usar en Linux, Windows o Linux, lo que facilita la **portabilidad** de las instrucciones de construcción
- Es muy adecuado para dockerizar aplicaciones en entornos de **integración continua**

# Dockerizar una aplicación compilada

- **Desventajas del Multistage Dockerfile**
  - El contenedor de construcción se borra automáticamente al finalizar su trabajo y **puede ser compleja** la depuración en caso de problemas porque no se puede acceder a ficheros temporales

# Ejercicio 5

- Crea un Multistage Docker file optimizando las capas para no descargar las librerías en cada construcción
- Utiliza la aplicación “**aplicacion-java-enunciado**”

# Ejercicio 5

- Solución

cache-multistage.Dockerfile

```
# Imagen base para el contenedor de compilación
FROM maven:3.9.0-eclipse-temurin-17 as builder
WORKDIR /project
COPY aplicacion-java-enunciado/pom.xml /project/
RUN mvn -B clean verify
COPY aplicacion-java-enunciado/src /project/src
RUN mvn -B -o package -DskipTests

# Imagen base para el contenedor de la aplicación
FROM eclipse-temurin:17-jdk
WORKDIR /usr/src/app/
COPY --from=builder /project/target/*.jar
/usr/src/app/
EXPOSE 8080
CMD [ "java", "-jar", "java-webapp-0.0.1.jar" ]
```

# Diferencias ENTRYPOINT y CMD

## • CMD

- Define un comando predeterminado que se ejecuta al iniciar un contenedor
- Este comando se puede sobrescribir al arrancar la imagen con el comando **"docker run image [OPTIONS]"**

```
FROM alpine:3.12
CMD ["echo", "Hello from CMD"]
```

```
$ docker container run my-image
Hello from CMD
```

```
$ docker container run my-image echo "Hello from the CLI"
Hello from the CLI
```

# Diferencias ENTRYPOINT y CMD

- **ENTRYPOINT**

- También define un comando por defecto que se ejecuta al iniciar un contenedor
- El **ENTRYPOINT** es ideal para las imágenes que siempre ejecutan el mismo **comando**
- Por defecto **ENTRYPOINT** ejecuta el comando **"/bin/sh"**
- A diferencia de **CMD** este comando no se puede reemplazar con el comando **"docker run image [OPTIONS]"**
- Sino que todos los comandos que pasemos al **CMD** se enviarán como parámetros al **ENTRYPOINT**

# Diferencias ENTRYPOINT y CMD

- ENTRYPOINT

```
FROM alpine:3.12
ENTRYPOINT ["echo"]
CMD ["Hello from CMD"]
```

```
$ docker container run my-image
Hello from CMD
```

```
$ docker container run my-image "Hello from the CLI"
Hello from the CLI
```



# Ejemplo ENTRYPOINT y CMD

ejemplo-3

- Aplicación de línea de comandos Java

Application.java

```
public class Application {
    public static void main(String[] args) {
        System.out.println("=====");
        System.out.println("Command line application");
        System.out.println("=====");

        System.out.println("=> Read arguments:");
        Arrays.stream(args)
            .forEach(arg -> System.out.println("\t" + arg));
    }
}
```

Dockerfile

```
FROM eclipse-temurin:17-jdk
WORKDIR /usr/src/app/
COPY --from=builder /project/target/app-jar-with-dependencies.jar
/usr/src/app/
EXPOSE 8080
ENTRYPOINT [ "java", "-jar", "app-jar-with-dependencies.jar" ]
CMD [ "--server" ]
```

# Ejemplo ENTRYPOINT y CMD

ejemplo-3

```
$ git clone \
  https://github.com/MasterCloudApps/3.2.Contenedores-y-orquestadores
$ cd 3.2.Contenedores-y-orquestadores/docker/ejemplo-3
$ docker build -t my-image .
```

```
$ docker run my-image
=====
Command line application
=====
=> Read arguments:
    --server
```

```
$ docker run my-image --override-argument1 --override-argument2
=====
Command line application
=====
=> Read arguments:
    --override-argument1
    --override-argument2
```

# Docker

- Introducción
- Volúmenes
- Configuración de contenedores
- Aplicaciones de consola
- Redes
- Creación de imágenes
- **Herramientas para la creación de imágenes**
- Desarrollo con contenedores

# Google jib

- Para ciertas aplicaciones de **tipos concretos** se han creado herramientas más optimizadas para crear las imágenes Docker
- **jib** es un plugin de Maven y Gradle desarrollado por Google que empaqueta aplicaciones Java directamente como contenedores Docker (sin pasar por un .jar)
- Las capas optimizadas para cachear librerías
- Al no generar el .jar envía sólo los .class de la aplicación

# Google jib

- **jib** es un **plugin de Maven y Gradle** que empaqueta aplicaciones Java directamente como contenedores Docker (**sin generar el .jar**)
- Las capas **optimizadas** para cachear librerías
- Al no generar el .jar **envía sólo los .class** de la aplicación (muy poco tamaño > poco tiempo de transferencia)
- La aplicación **arranca más rápido** (*exploded jar*)



<https://github.com/GoogleContainerTools/jib>

# Google jib

- **jib no necesita el docker engine** para generar las imágenes Docker, todo lo hace con Java
- **Aumenta la seguridad** en el entorno de CI porque no necesita permisos de administración (necesarios para Docker) para crear una imagen
- Desde la versión 3+ de jib se utiliza la imagen base **eclipse-temurin** anteriormente **distroless**

[https://hub.docker.com/\\_/eclipse-temurin](https://hub.docker.com/_/eclipse-temurin)

<https://github.com/GoogleContainerTools/distroless>

[https://github.com/GoogleContainerTools/jib/blob/master/docs/default\\_base\\_image.md](https://github.com/GoogleContainerTools/jib/blob/master/docs/default_base_image.md)

- Crear una imagen utilizando jib
  - Para crear una imagen de un proyecto java con maven ejecutamos el siguiente comando

```
$ ./mvnw compile \
  com.google.cloud.tools:jib-maven-plugin:3.3.1:build \
  -Dimage=miusuario/repositorio
```

- Es posible almacenar la imagen en el **docker engine local** sin pasar por un registro remoto

```
$ ./mvnw compile \
  com.google.cloud.tools:jib-maven-plugin:3.3.1:dockerBuild \
  -Dimage=miusuario/repositorio
```

# Google jib

pom.xml

```
<project>
  <groupId>example</groupId>
  <artifactId>sprint-boot-example</artifactId>
  <version>0.1.0</version>

  ...

  <build>
    <plugins>
      <plugin>
        <groupId>com.google.cloud.tools</groupId>
        <artifactId>jib-maven-plugin</artifactId>
        <version>3.3.1</version>
      </plugin>
    </plugins>
  </build>
</project>
```

```
$ ./mvnw compile jib:build -Dimage=miusuario/repositorio
```

```
$ ./mvnw compile jib:dockerBuild -Dimage=miusuario/repositorio
```



# Google jib

- jib se puede utilizar tanto con registros **privados** o **públicos** como Docker Hub
- Existen varias formas de almacenar las credenciales
  - La más común es disponer del fichero **"\$HOME/.docker/config.json"**

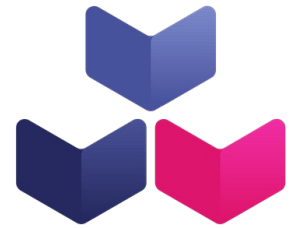
```
{
  "auths": {
    "https://index.docker.io/v1/": {
      "auth": "aaaaaaaaaaaaaaaa"
    }
  }
}
```

# Ejercicio 6

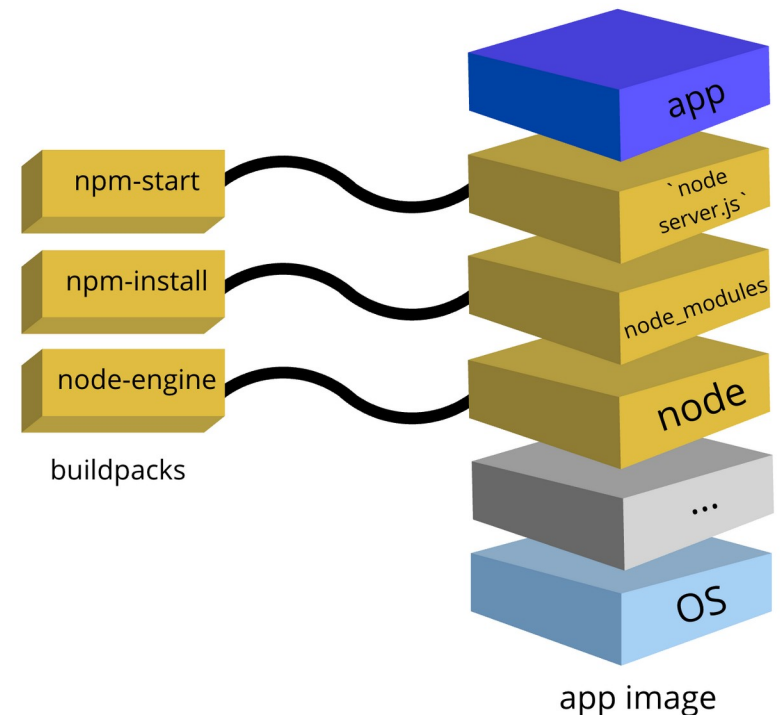
- Crea una imagen Docker con jib
  - Utiliza la aplicación “**aplicacion-java-enunciado**”

# Buildpacks

- Traduce el **código** fuente a **imágenes**
- No hay que utilizar Dockerfiles
- Cacheo de capas de forma optimizada
- Multilenguaje
- Imágenes minimas
- Builder y buildpacks
- **Pack CLI utiliza Docker**



Buildpacks.io



<https://buildpacks.io/>  
<https://buildpacks.io/docs/tools/pack/>

# Paketo



- Implementación de Buildpacks
- Nos proporciona diferentes Builders para diferentes lenguajes (Java, Node, python, Go...)
- Existen diferentes implementaciones de otros proveedores (Google, Heroku)

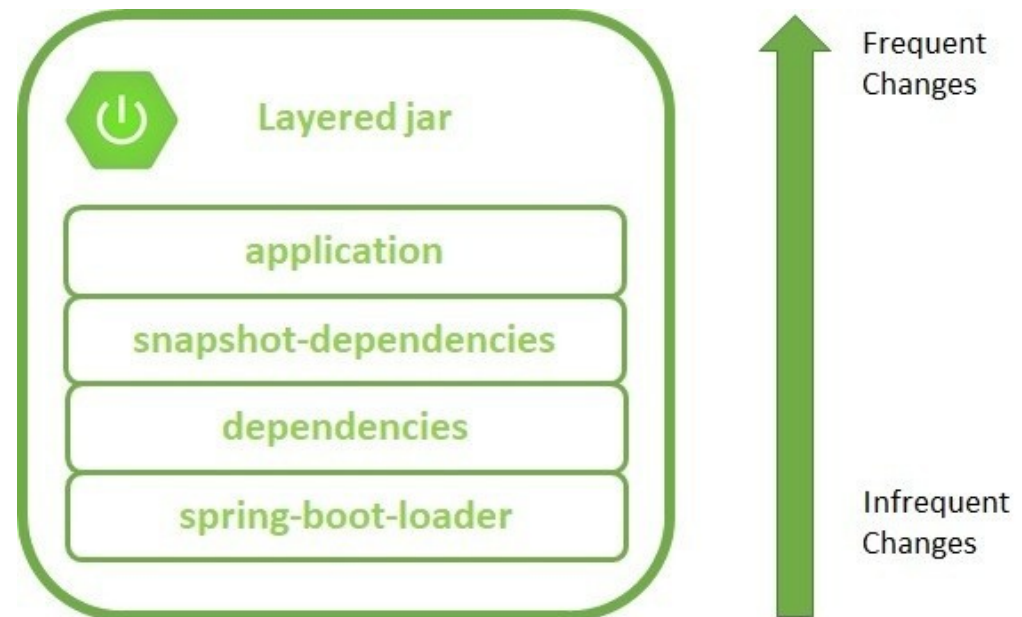
<https://paketo.io/>

<https://cloud.google.com/docs/buildpacks>

<https://github.com/heroku/builder>

# Spring Boot y Buildpacks

- A partir de Spring Boot 2.3.0
  - Soporte para Buildpacks
  - JAR con capas



# Spring Boot y Buildpacks

## • Compilación utilizando Buildpacks

- Crear imagen de la aplicación Spring Boot utilizando Buildpacks

```
$ mvn spring-boot:build-image
```

- Por defecto el nombre de la imagen será **"artifactId:version"**
- Crear imagen especificando un nombre

```
$ mvn spring-boot:build-image \
  -Dspring-boot.build-image.imageName=miusuario/mi-app:v1
```

- Subir la imagen a Docker Hub

```
$ docker push miusuario/mi-app:v1
```

# Spring Boot y Buildpacks

- Especificar el nombre de la imagen en el fichero pom.xml

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>

  <configuration>
    <image>
      <name>miusuario/mi-app:${project.version}</name>
    </image>
  </configuration>
</plugin>
```

# Ejercicio 7

- Crea una imagen Docker con Buildpacks
  - Utiliza la aplicación “aplicacion-java-enunciado”



# Imagen nativa GraalVM

- Las imágenes nativas de GraalVM son **ejecutables específicos del sistema operativo**
- **No requieren una JVM** para ejecutarse (la llevan integrada)
- Inician su ejecución mucho **más rápido** que las aplicaciones con JVM y consumen **menos memoria**
- Es una tecnología **muy novedosa** (primera versión en 2019) y está **evolucionando** muy rápido
- La generación del paquete optimizado tarda **varios minutos**



<https://www.graalvm.org/latest/reference-manual/native-image/>

# Spring Boot native image y Builpack

- Disponible en **Spring Boot** desde Nov 2022
- Ciertas opciones de **Spring Boot** no son compatibles con **native images**
- Haciendo uso de **Builpacks** podremos generar una imagen Docker con **native image**
- Las imágenes Docker generadas no contienen ninguna **JVM**, dentro de la imagen solo se encuentra un fichero **native image**
- Ejecución mas **rápida**, con un **consumo menor** de memoria y **tamaño** de las imágenes Docker **reducido**

<https://docs.spring.io/spring-boot/docs/current/reference/html/native-image.html>

# Spring Boot native image y Builpack

- **Compilación utilizando Buildpacks**
  - Primero tendremos que añadir la dependencia “**GraalVM Native Support**”
  - Esto configura el plugin de Spring Boot para soporte de imágenes nativas

```
<plugins>
  <plugin>
    <groupId>org.graalvm.buildtools</groupId>
    <artifactId>native-maven-plugin</artifactId>
  </plugin>
  <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
  </plugin>
</plugins>
```

# Spring Boot native image y Builpack

## • Compilación utilizando Buildpacks

- Crear imagen de la aplicación Spring Boot utilizando Buildpacks

```
$ mvn -Pnative spring-boot:build-image
```

- Por defecto el nombre de la imagen será "artifactId:version"
- Crear imagen especificando un nombre

```
$ mvn -Pnative spring-boot:build-image \
-Dspring-boot.build-image.imageName=miusuario/mi-app-native:v1
```

- Subir la imagen a Docker Hub

```
$ docker push miusuario/mi-app-native:v1
```

# Spring Boot native image y Builpack

- Especificar el nombre de la imagen en el fichero pom.xml

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>

  <configuration>
    <image>
      <name>miusuario/mi-app-native:${project.version}</name>
    </image>
  </configuration>
</plugin>
```

# Ejercicio 8

- Crea una imagen Docker GraalVM native con Builpacks
- Utiliza la aplicación “aplicacion-java-enunciado”

# Docker

- Introducción
- Volúmenes
- Configuración de contenedores
- Aplicaciones de consola
- Redes
- Creación de imágenes
- Herramientas para la creación de imágenes
- **Desarrollo con contenedores**

# Desarrollo con contenedores

## • Ventajas

- Evita que varios desarrolladores tengan diferentes versiones de las herramientas (Java, maven, MySQL...)
- Muy sencillo probar el código con diferentes versiones de Java, Base de datos, etc.
- Reduce el tiempo de setup inicial de los entornos, son muy reproducibles porque están en contenedores
- Las mismas herramientas se pueden usar en la máquina de desarrollo y en CI



# Desarrollo con contenedores

## • Ventajas

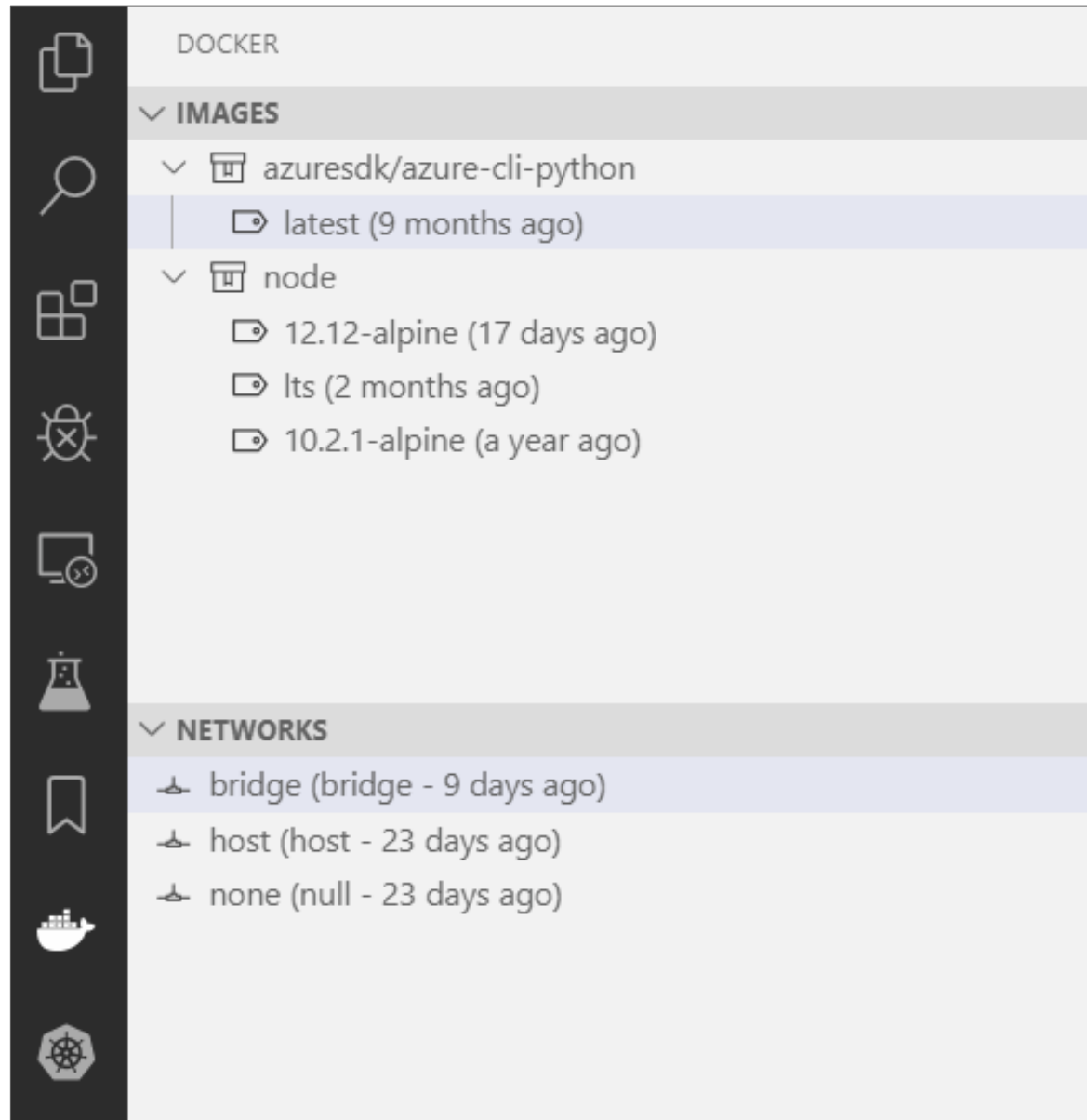
- Con lenguajes de script (Node.js, python...) se puede montar un volumen entre el host y el contenedor con el código fuente
- Se editan los ficheros en el host usando un Entorno de Desarrollo (IDE) y se reinicia la aplicación dentro del contenedor

# Desarrollo con contenedores

- Plugins para IDEs
  - Que permiten gestionar contenedores e imágenes de forma interactiva

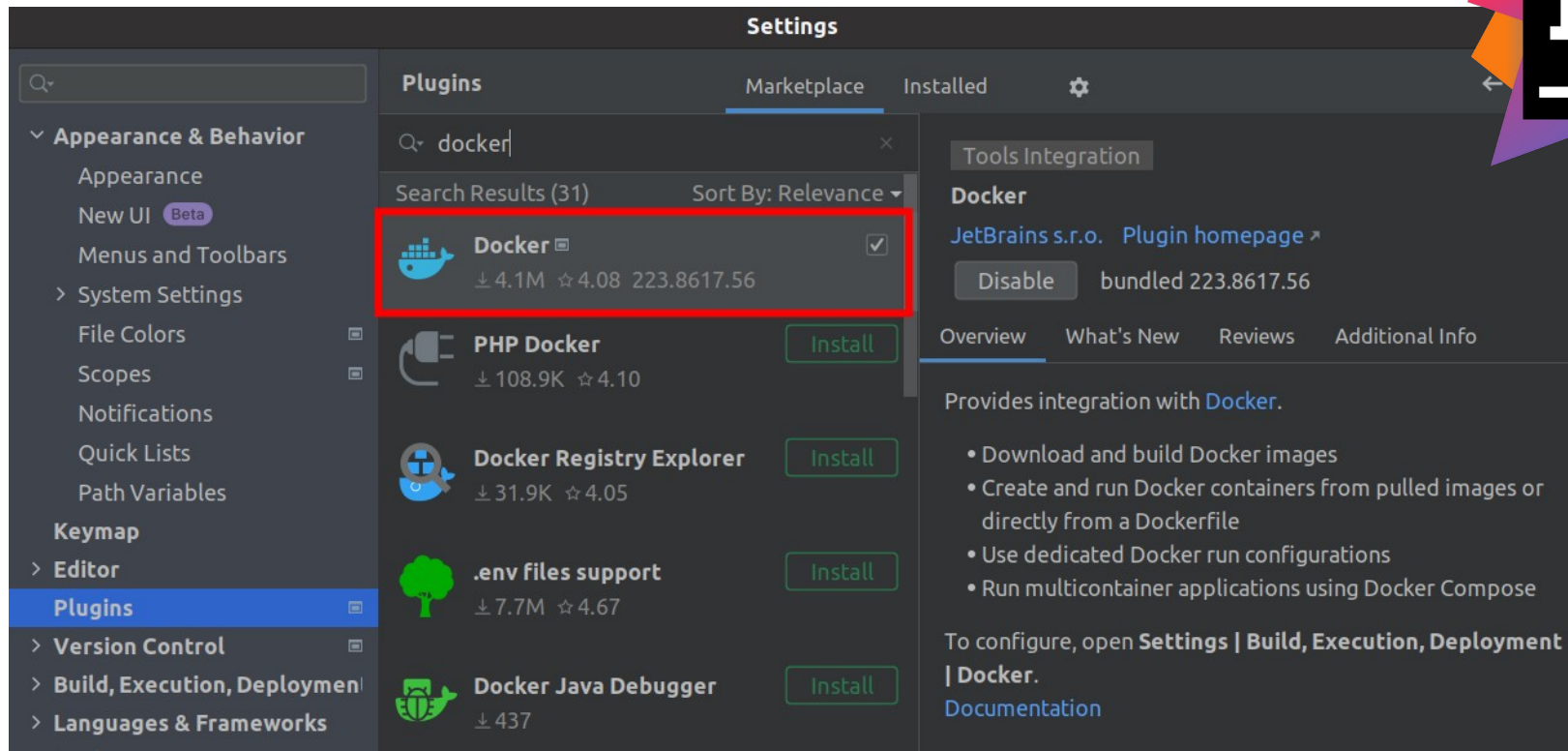


# Desarrollo con contenedores



# Desarrollo con contenedores

- Plugins para IDEs
  - Que permiten gestionar contendores e imágenes de forma interactiva



# Desarrollo con contenedores

Services

Build Log Log Dashboard Terminal (1)

adoring\_grothendieck 0ab0695a <none>: <none> Restart Stop Terminal

2023-03-09T20:53:49.392Z INFO 1 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApp

2023-03-09T20:53:49.394Z INFO 1 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initial

2023-03-09T20:53:49.873Z INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (ht

2023-03-09T20:53:49.893Z INFO 1 --- [main] e.u.code.daw.tablonanuncios.Application : Started Application in 2.068 second

ls

- Recreate Container
- Edit Configuration
- Restart Container
- Pause Container
- Stop Container Ctrl+F2
- Image >
- Show Files
- Show Log
- Copy Container ID
- Copy Image ID
- Inspect
- Show Processes
- Attach
- Exec
- Create Terminal
- Open in New Tab
- Open Each in New Tab
- Open Each Type in New Tab
- Delete... Delete
- Jump to Source F4

```
1 # Selecciona la imagen base
2 FROM maven:3.9.0-eclipse-temurin-17
3
4 # Define el directorio de trabajo donde ejecutar comandos
5 WORKDIR /project
6
7 # Copia el código del proyecto
8 COPY /src /project/src
9 COPY pom.xml /project/
10
11 # Compila proyecto y descarga librerías
12 RUN mvn -B package -DskipTests
13
14 # Indica el puerto que expone el contenedor
15 EXPOSE 8080
16
17 # Comando que se ejecuta al hacer docker run
18 CMD ["java", "-jar", "target/java-webapp-0.0.1.jar"]
```

# Desarrollo con contenedores

## • Desventajas

- En lenguajes compilados (Java, Go...) dónde se realiza la compilación?
- **En el contenedor usando línea de comandos? >**  
Mala experiencia del desarrollador
- **En el host usando un IDE como Eclipse? >**  
Necesitamos Java y Maven en el host. No hay independencia/reproducibilidad

# Desarrollo con contenedores



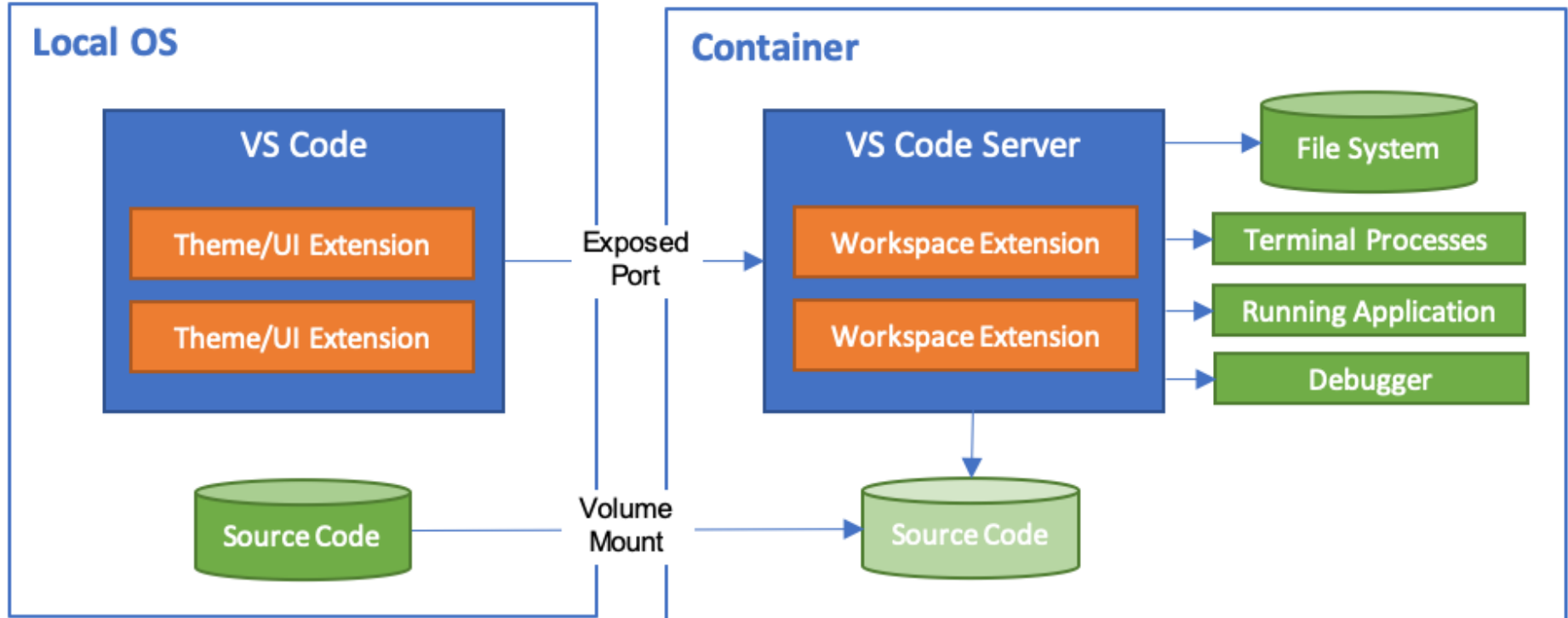
## Visual Studio Code

Visual Studio Code Remote - Containers

<https://code.visualstudio.com/docs/remote/containers>

# Desarrollo con contenedores

- VSCode remote - Containers





# Desarrollo con contenedores

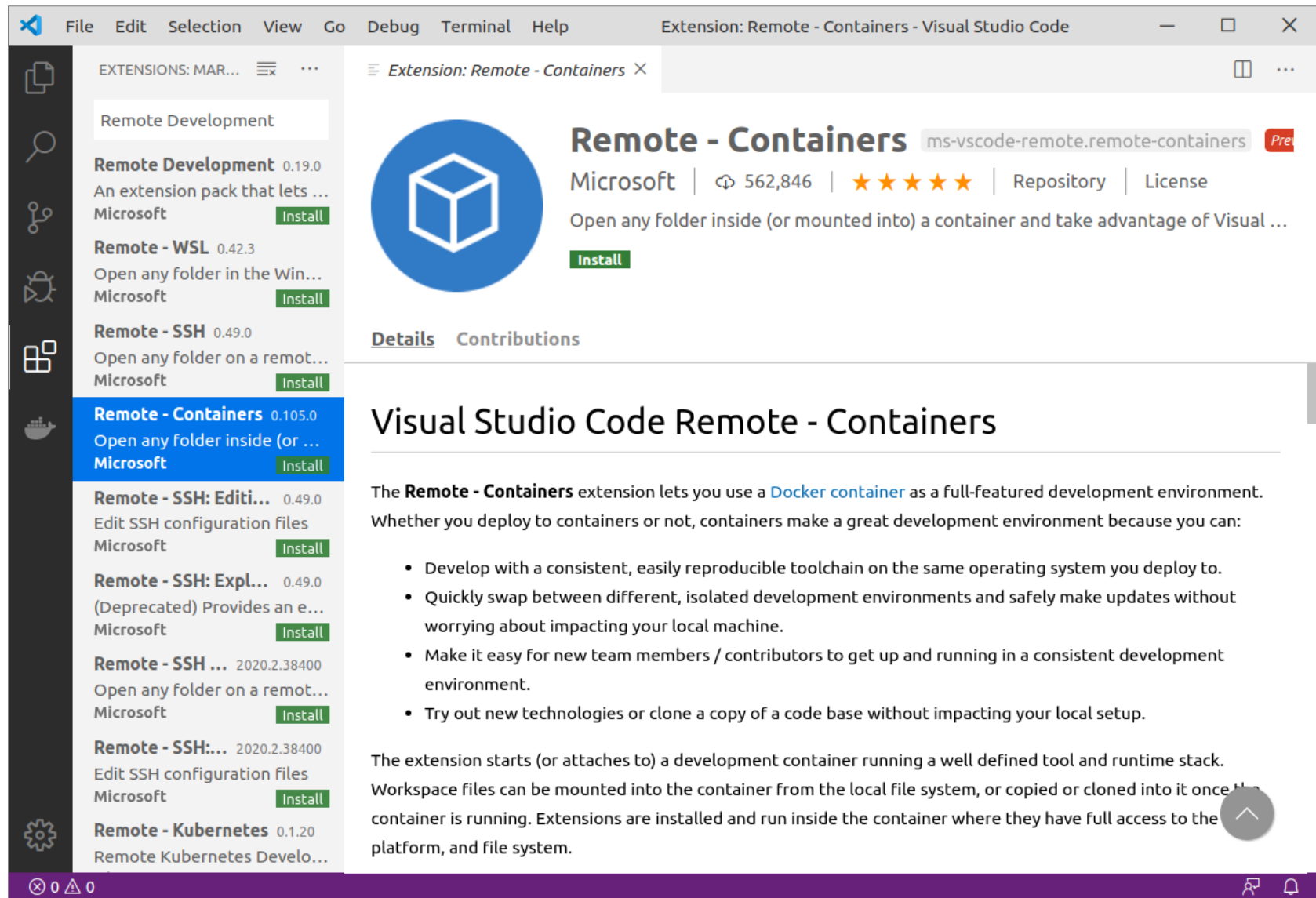
- **VSCode remote - Containers**
  - Permite al usuario programar en el VSCode instalado en el host
  - Los plugins de compilación (Java, Go...) y depuradores se ejecutan en el contenedor
  - Portabilidad y experiencia interactiva



## Algunas limitaciones

- No soporta Docker Toolbox
- Contenedores sin glibc (como alpine) pueden tener problemas
- En windows y Mac hay que revisar la configuración de carpetas compartidas

# Desarrollo con contenedores



The screenshot shows the Visual Studio Code interface with the 'Remote - Containers' extension page open. The left sidebar displays a list of extensions, with 'Remote - Containers' selected and highlighted in blue. The main panel shows the details for the 'Remote - Containers' extension by Microsoft, including its version (0.105.0), a five-star rating, and an 'Install' button. Below the extension details, the title 'Visual Studio Code Remote - Containers' is followed by a description: 'The Remote - Containers extension lets you use a Docker container as a full-featured development environment. Whether you deploy to containers or not, containers make a great development environment because you can:'. A bulleted list follows, detailing the benefits of using containers for development. At the bottom, a paragraph explains how the extension starts a development container and allows workspace files to be mounted into it.

EXTENSIONS: MAR... **Remote Development** 0.19.0  
An extension pack that lets ...  
Microsoft **Install**

**Remote - WSL** 0.42.3  
Open any folder in the Win...  
Microsoft **Install**

**Remote - SSH** 0.49.0  
Open any folder on a remot...  
Microsoft **Install**

**Remote - Containers** 0.105.0  
Open any folder inside (or ...  
Microsoft **Install**

**Remote - SSH: Editi...** 0.49.0  
Edit SSH configuration files  
Microsoft **Install**

**Remote - SSH: Expl...** 0.49.0  
(Deprecated) Provides an e...  
Microsoft **Install**

**Remote - SSH ...** 2020.2.38400  
Open any folder on a remot...  
Microsoft **Install**

**Remote - SSH:...** 2020.2.38400  
Edit SSH configuration files  
Microsoft **Install**

**Remote - Kubernetes** 0.1.20  
Remote Kubernetes Develo...

**Remote - Containers** ms-vscode-remote.remote-containers **Pre**  
Microsoft | 562,846 | ★★★★★ | Repository | License  
Open any folder inside (or mounted into) a container and take advantage of Visual ...  
**Install**

**Remote - Containers** ms-vscode-remote.remote-containers **Pre**  
Microsoft | 562,846 | ★★★★★ | Repository | License  
Open any folder inside (or mounted into) a container and take advantage of Visual ...  
**Install**

**Visual Studio Code Remote - Containers**

The **Remote - Containers** extension lets you use a [Docker container](#) as a full-featured development environment. Whether you deploy to containers or not, containers make a great development environment because you can:

- Develop with a consistent, easily reproducible toolchain on the same operating system you deploy to.
- Quickly swap between different, isolated development environments and safely make updates without worrying about impacting your local machine.
- Make it easy for new team members / contributors to get up and running in a consistent development environment.
- Try out new technologies or clone a copy of a code base without impacting your local setup.

The extension starts (or attaches to) a development container running a well defined tool and runtime stack. Workspace files can be mounted into the container from the local file system, or copied or cloned into it once the container is running. Extensions are installed and run inside the container where they have full access to the platform, and file system.

# VSCode Remote - Containers

- **Modos de funcionamiento**
  - Desarrollo completo dentro de un contenedor
  - Conexión a un contenedor en ejecución para inspeccionar su contenido y ejecutar comandos

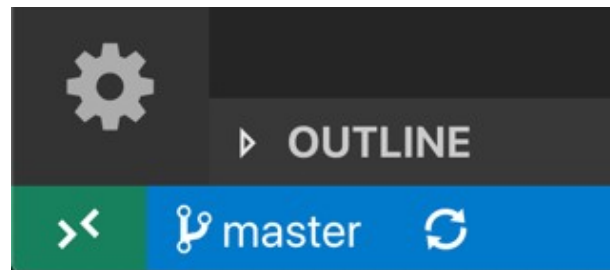
# VSCode Remote - Containers

- **Desarrollo completo dentro de un contenedor**

- 1) Clona el repositorio

```
$ git clone https://github.com/Microsoft/vscode-remote-try-java
```


- 2) Click en el icono de desarrollo remoto



- 3) Select Remote-Containers: Selecciona la carpeta del repositorio

# VSCode Remote - Containers

- **Desarrollo completo dentro de un contenedor**
  - Se crea el contenedor de desarrollo

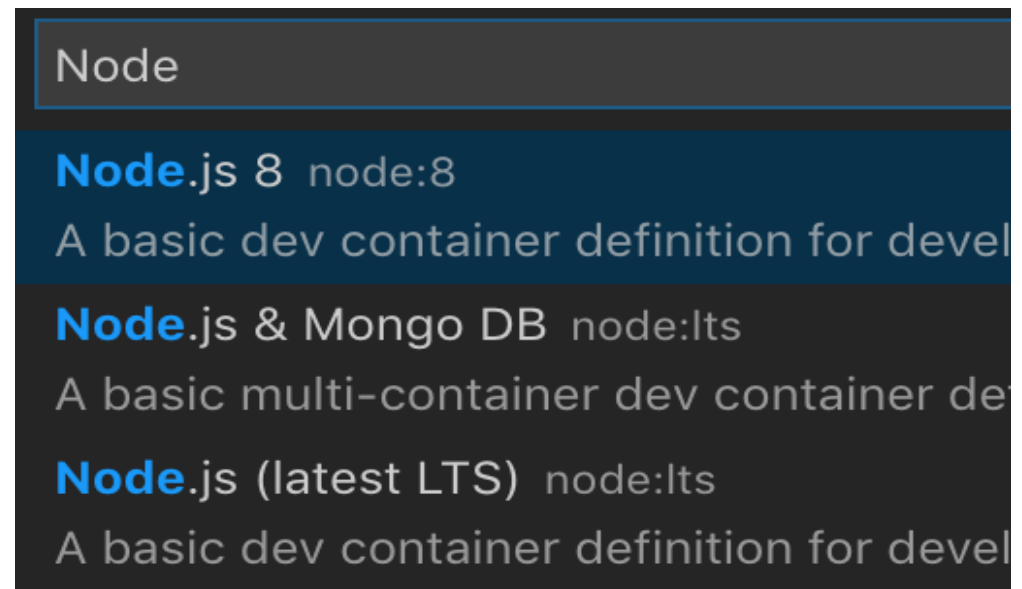
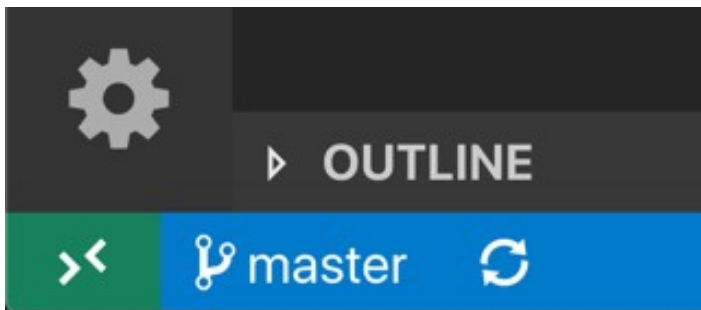
 Installing Dev Container ([details](#)): Building an image from the Dock...

- El README contiene un tutorial de uso
- El proyecto sabe la imagen que abrir porque está configurada en el fichero

`.devcontainer/devcontainer.json`

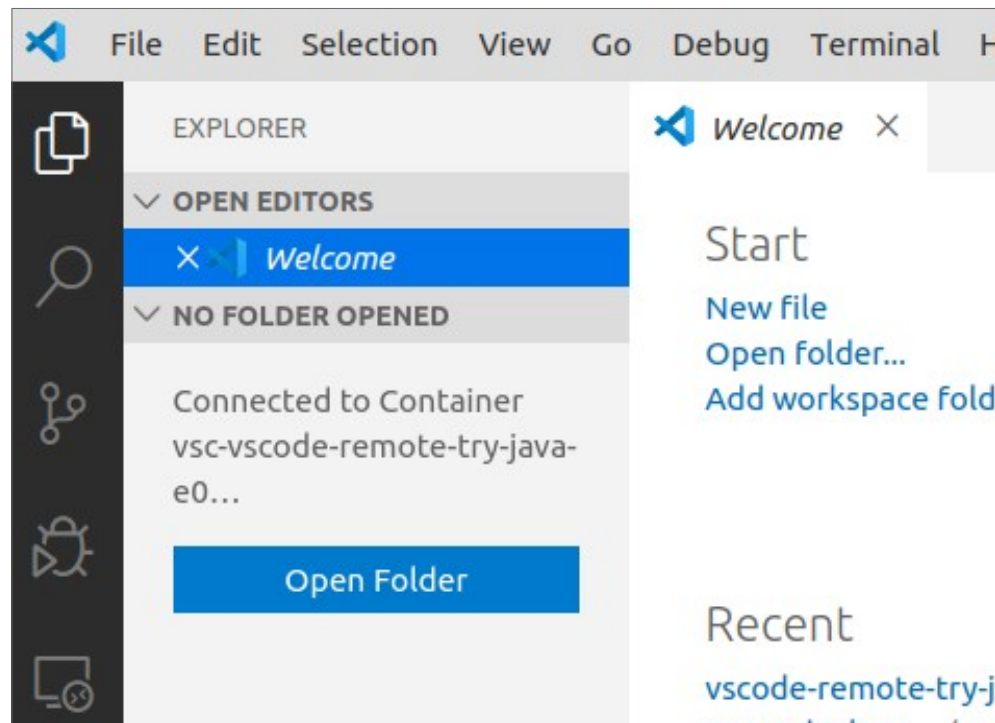
# VSCode Remote - Containers

- **Desarrollo completo dentro de un contenedor**
  - Existen imágenes base predefinidas en VSCode para “Abrir una carpeta dentro de un contenedor” si no tiene configuración previa



# VSCode Remote - Containers

- Conectarse a un contenedor arrancado
  - Remote-Containers: Attach to Running Container...



[https://code.visualstudio.com/docs/remote/containers#\\_attaching-to-running-containers](https://code.visualstudio.com/docs/remote/containers#_attaching-to-running-containers)