Tema 1

Docker

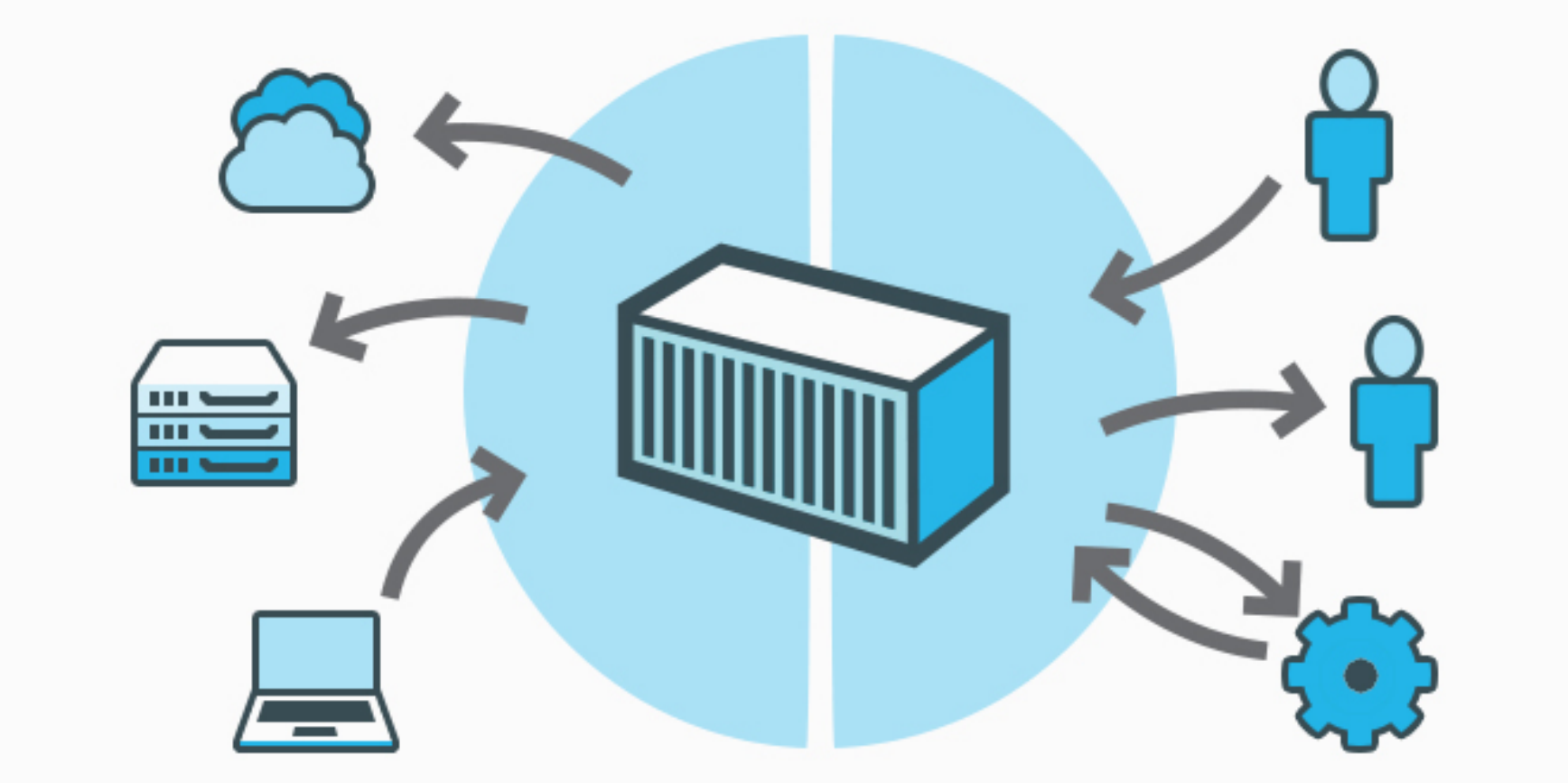
Universidad
Rey Juan Carlos

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/

Docker

- **¿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**

Docker

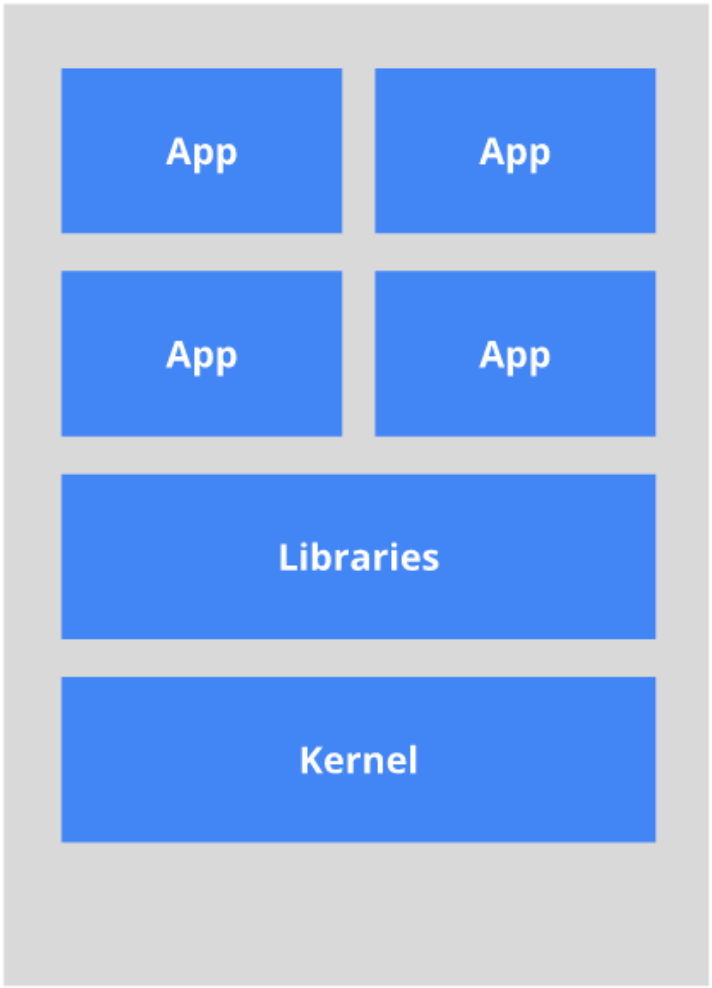
- **Formato de distribución y ejecución de servicios en linux**
  - Cada sistema linux tiene su **propio** sistema de **distribución y ejecución de servicios**
  - Los servicios **comparten recursos del servidor** sin ningún tipo de aislamiento entre ellas
  - Un **servicio** depende de las **versiones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

Docker

- **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

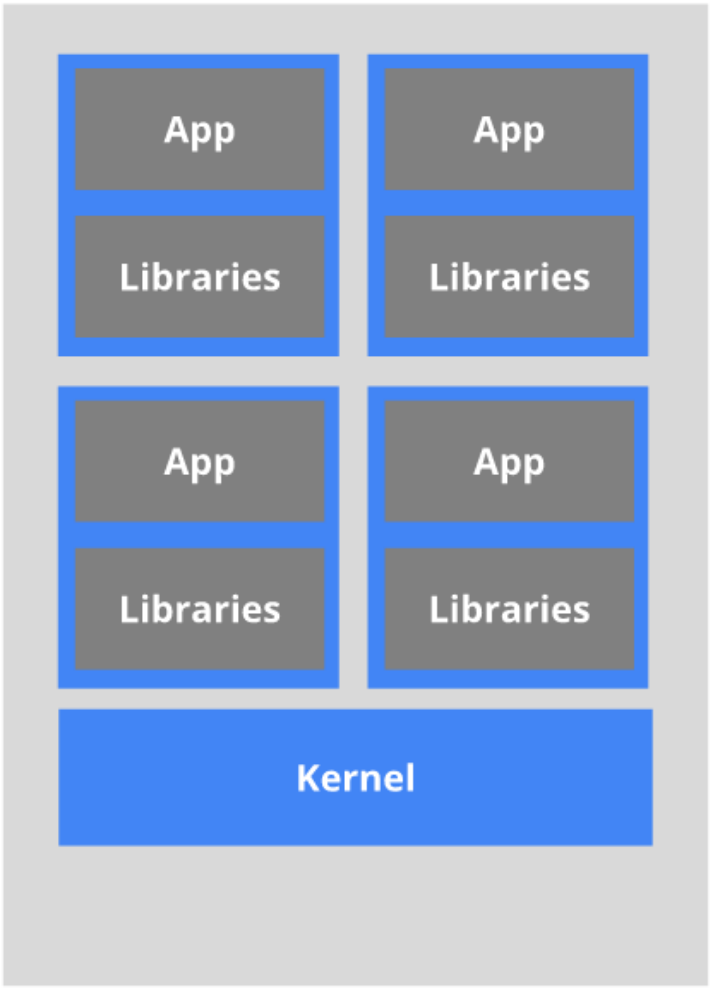
Docker

The old way: Applications on host



Heavyweight, non-portable
Relies on OS package manager

The new way: Deploy containers



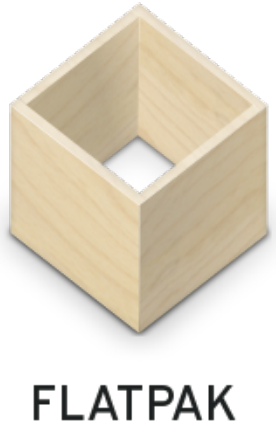
Small and fast, portable
Uses OS-level virtualization

Docker

- **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



https://snapcraft.io/

https://flatpak.org/

Docker

- **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 en su **propia máquina**
- Se tienen ventajas similares a las **máquinas virtuales** pero de forma mucho **más eficiente**

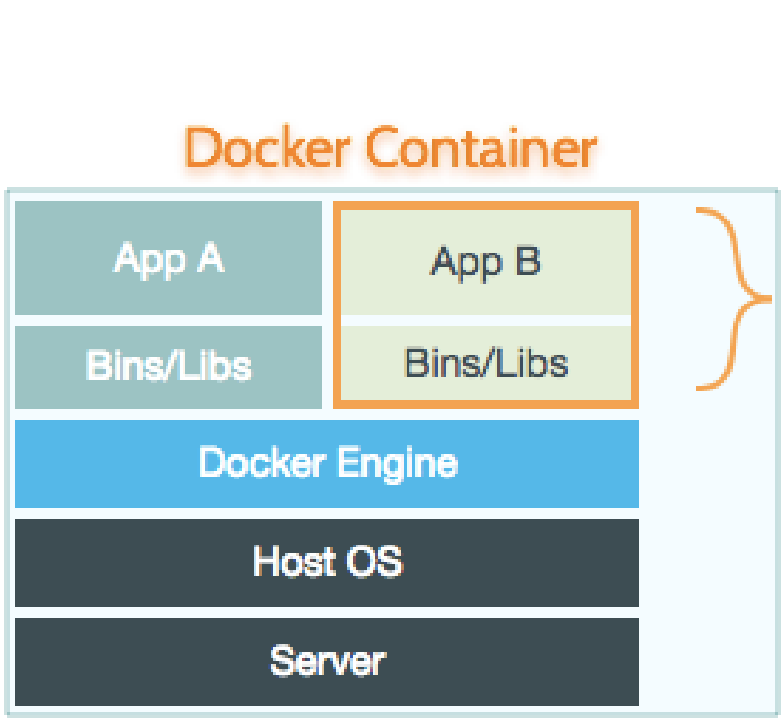
¿Cómo funciona Docker?

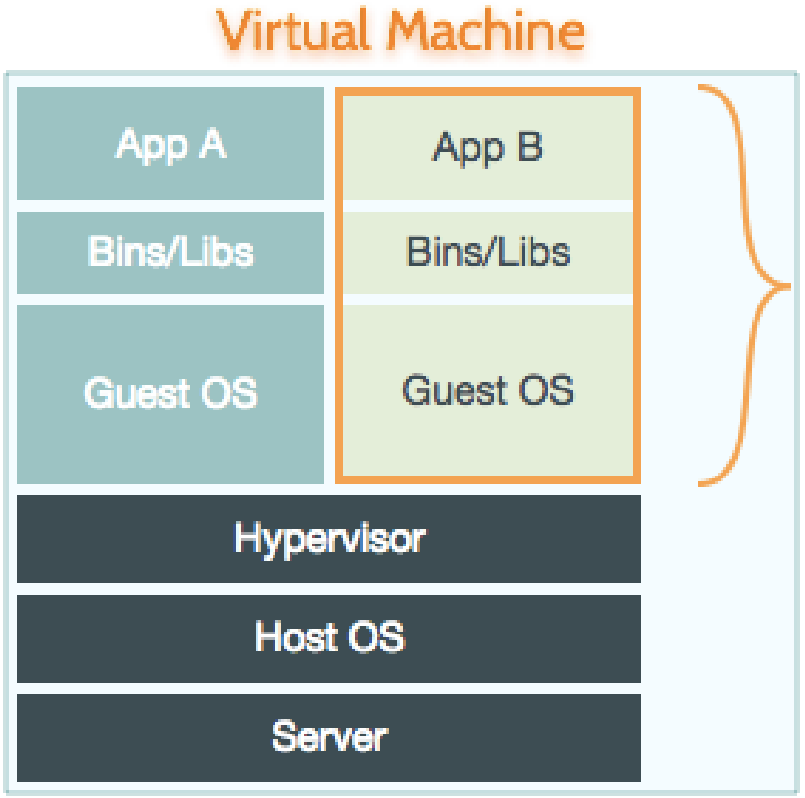
- Los **contenedores** son una tecnología que ofrece unas **ventajas similares** a las VMs pero **aprovechando** mejor los recursos:
  - Los contenedores tardan **milisegundos** en arrancar
  - Consumen únicamente la **memoria** que necesita la app ejecutada en el contenedor.
    - Una VMs **reserva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





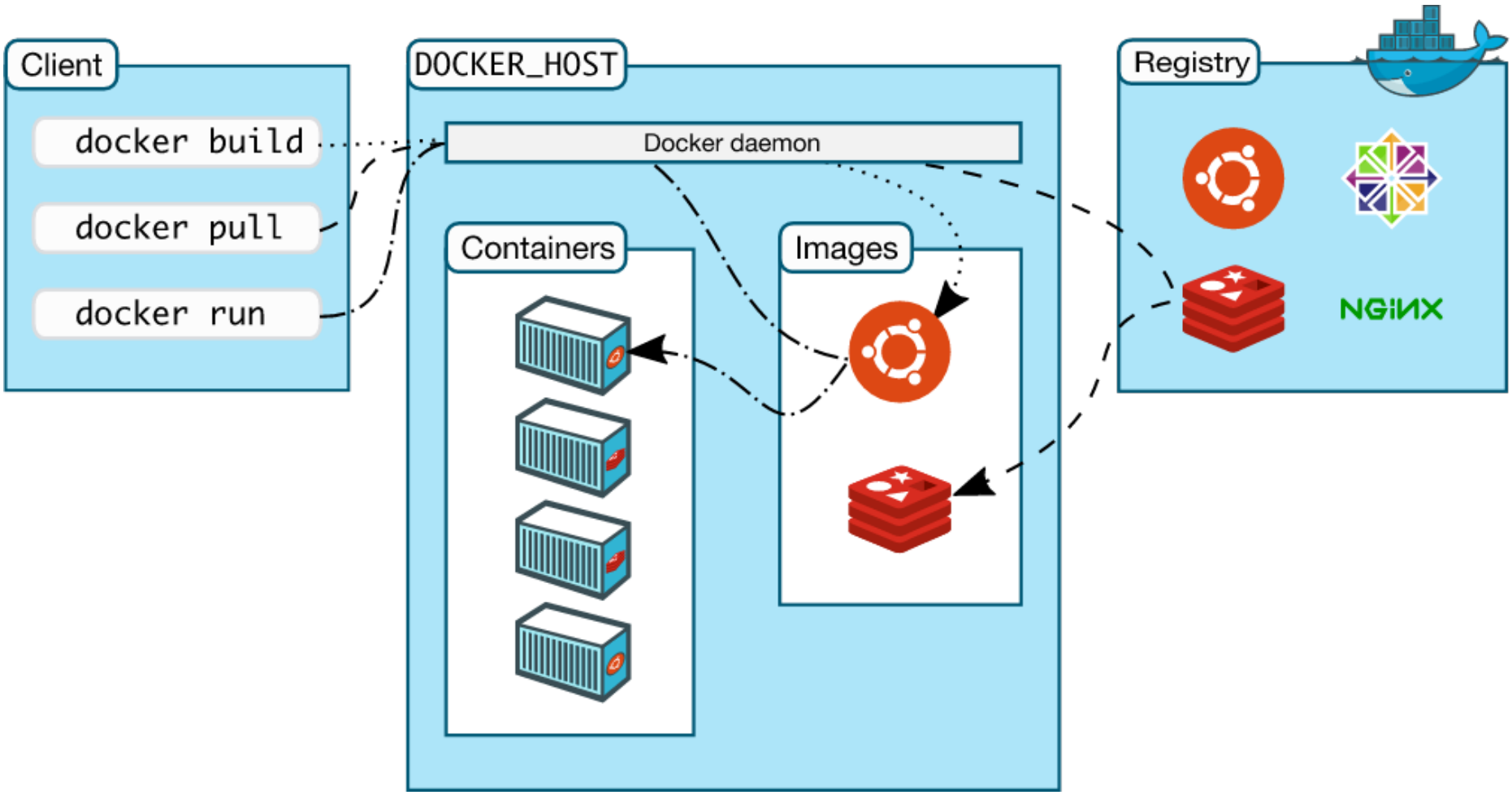
¿Cómo funciona Docker?

code URJC

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

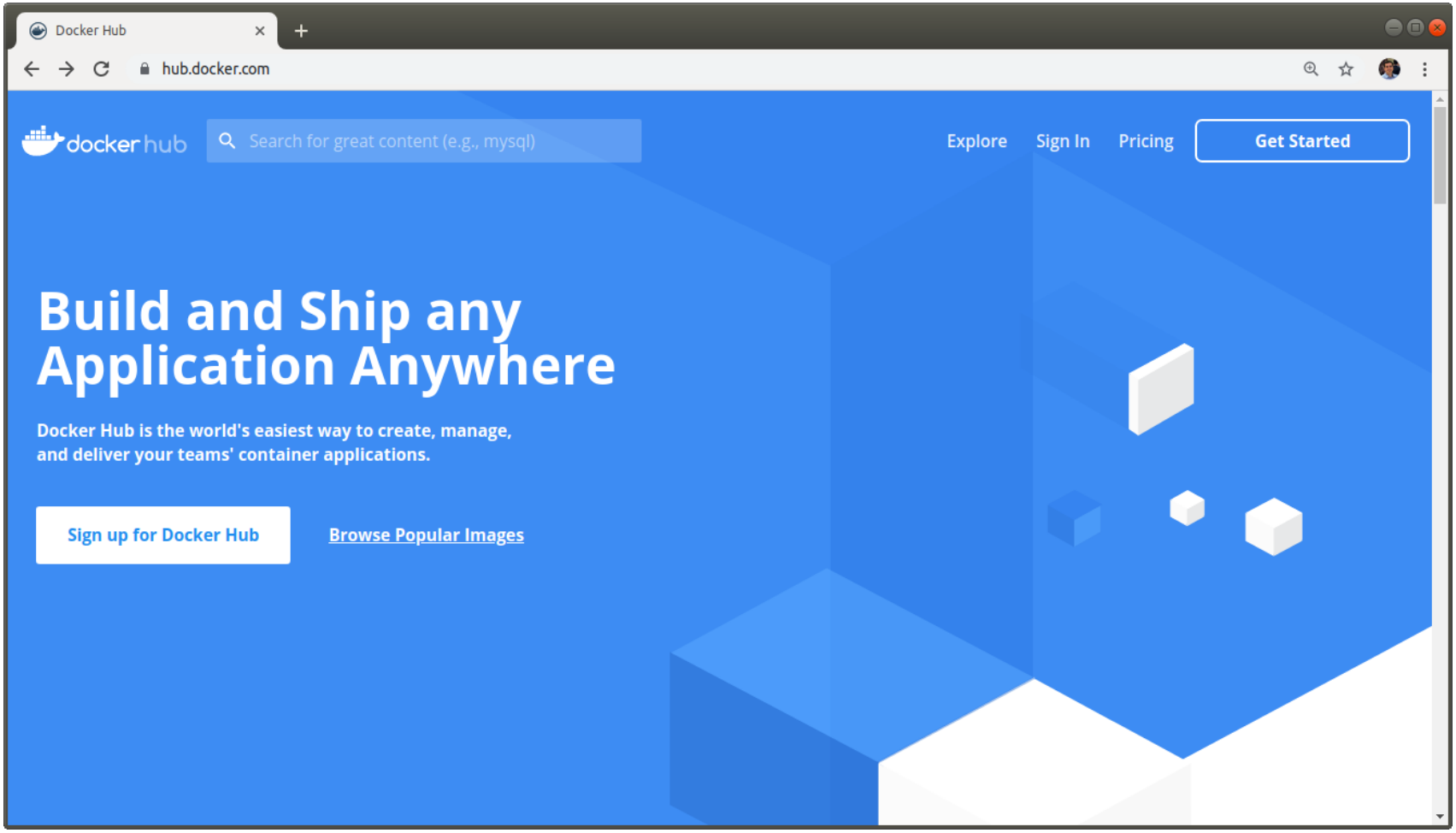
- **Imagen docker**
  - Un contenedor siempre se inicia desde una **imagen**
  - Si se quiere arrancar un contenedor partiendo de una imagen que no está disponible, se **descarga automáticamente** de Internet

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**

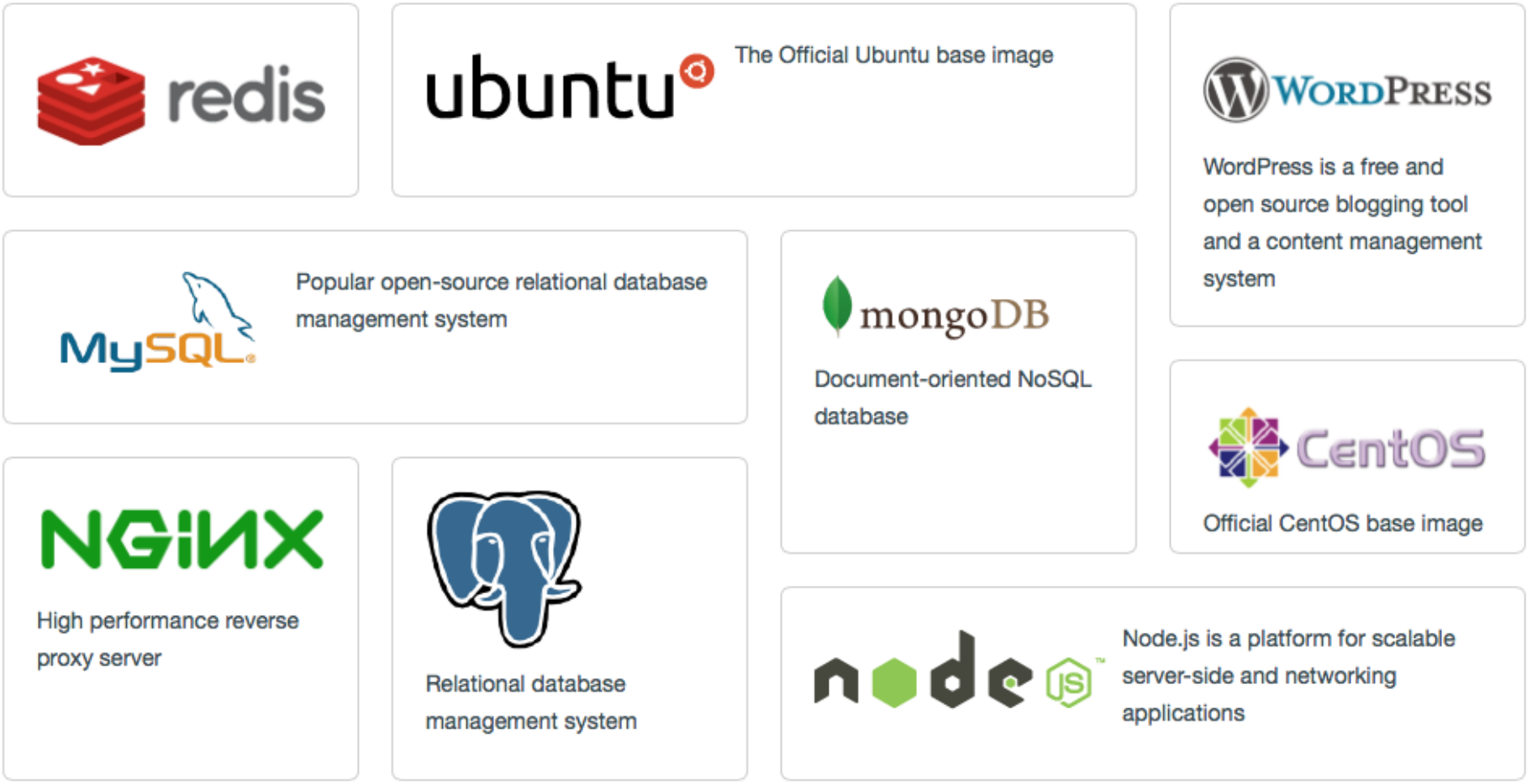
Conceptos Docker

- Docker Hub

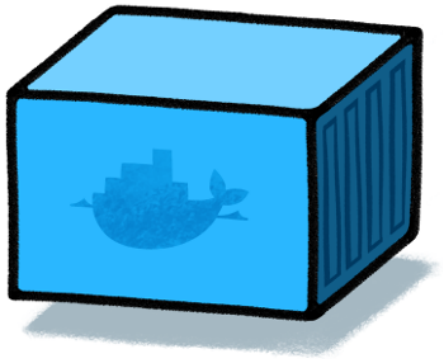


Conceptos Docker

- **Docker Hub:** Algunos repositorios oficiales



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:**
  - Ejecución nativa
  - Funcionalidad completa
- **Mac**
  - Docker for Mac
    - Usa virtualización ligera para ejecutar el kernel de linux
    - Funcionalidad algo más limitada que linux
- **Windows**
  - Docker for Windows (Win10 Professional or Enterprise 64-bit)
    - Usa virtualización ligera para ejecutar el kernel de linux
    - Funcionalidad algo más limitada que linux
  - Docker Toolbox
    - Usa VirtualBox para crear una VM pesada
    - Funcionalidad mucho más limitada

Instalación de Docker

- **Windows:**
  - Microsoft Windows 10 Professional or Enterprise 64-bit: https://store.docker.com/editions/community/docker-ce-desktop-windows
  - Other Windows versions: https://www.docker.com/products/docker-toolbox
- **Linux:**
  - Ubuntu: https://store.docker.com/editions/community/docker-ce-server-ubuntu
  - Fedora: https://store.docker.com/editions/community/docker-ce-server-fedora
  - Debian: https://store.docker.com/editions/community/docker-ce-server-debian
  - CentOS: https://store.docker.com/editions/community/docker-ce-server-centos
- **Mac:**
  - Apple Mac OS Yosemite 10.10.3 or above: https://store.docker.com/editions/community/docker-ce-desktop-mac
  - Older Mac: https://www.docker.com/products/docker-toolbox

Ejecución de contenedores

Ejecutar "hello-world" en un contendor

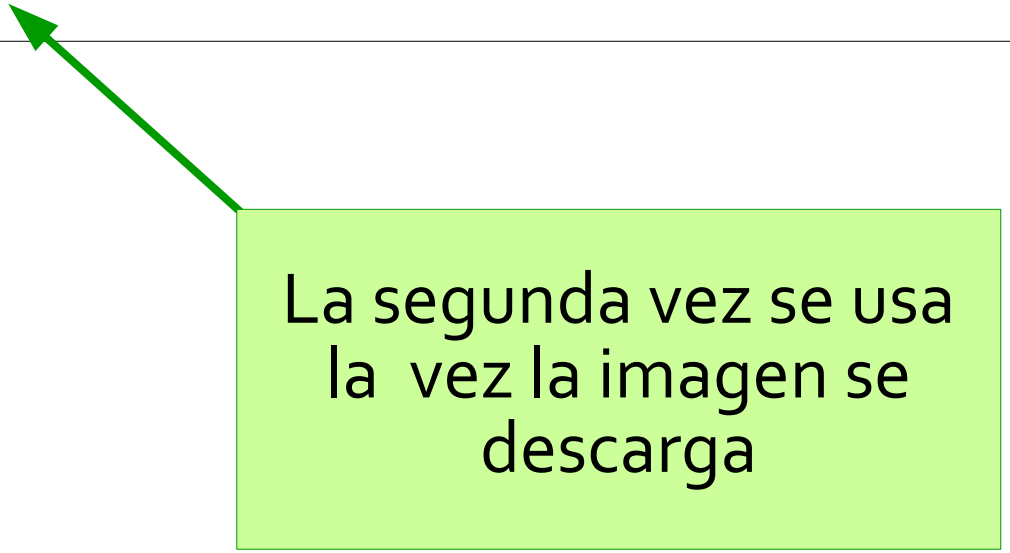
```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
03f4658f8b78: Pull complete
a3ed95caeb02: Pull complete
Digest:
sha256:8be990ef2aeb16dbcb9271ddfe2610fa6658d13f6dfb8bc72074cc1ca369
66a7
Status: Downloaded newer image for hello-world:latest

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

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.
...
```



Ejecución de contenedores

Inspeccionar los contendores 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
seqvence/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
...
```

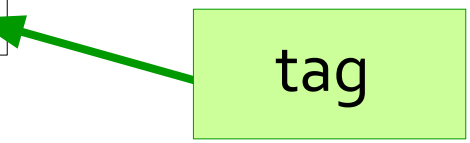
https://docs.docker.com/engine/reference/commandline/image/

Imágenes docker

- Tags

```
$ docker image ls
REPOSITORY                TAG       IMAGE ID       CREATED         SIZE
seqvence/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
```



- 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 la la **última versión estable** de la imagen

Servicios de red

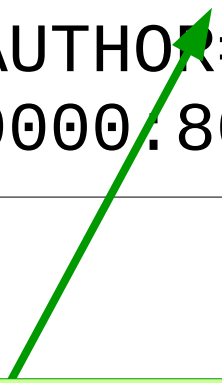
- Servidor web en un contenedor

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

Servicios de red

- Servidor web en un contenedor

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

-d

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

Servicios de red

- Servidor web en un contenedor

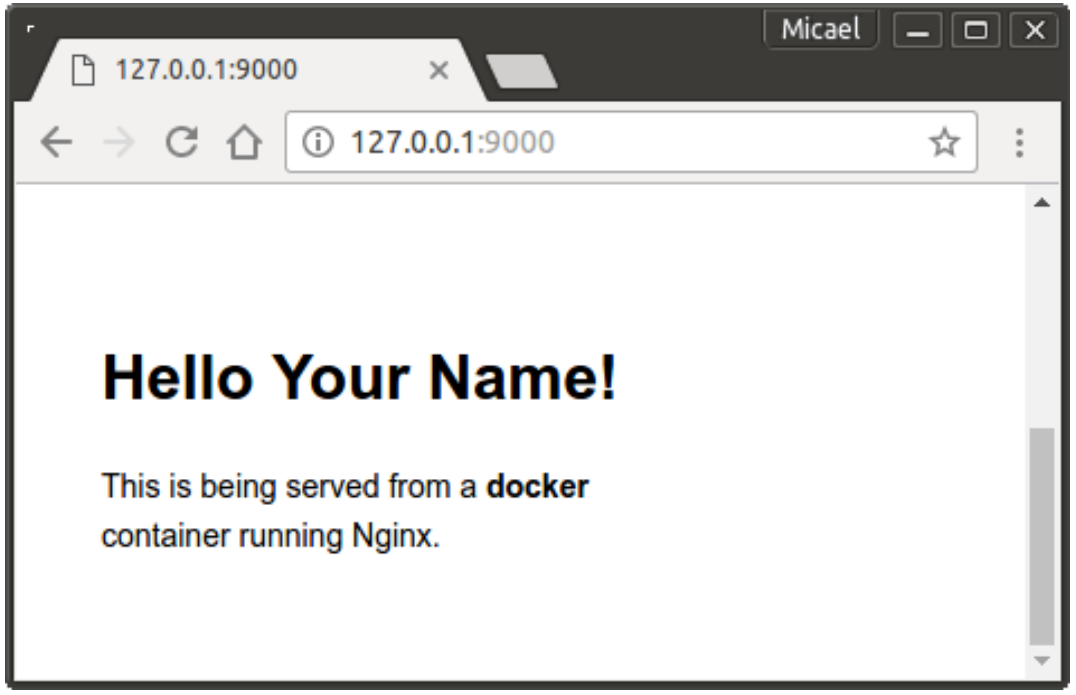
```
docker run --name static-site \
   -e AUTHOR="Your Name" -d \
   -p 9000:80 seqvence/static-site
```

-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



Servicios de red

- **Usar el servicio**
  - Si tienes **Docker Toolbox** para Mac o Windows no se puede usar la IP 127.0.0.1
  - Tienes que usar l**a IP de la máquina virtual** en la que se ejecuta Docker

```
$ docker-machine ip default
192.168.99.100
```

  - Típicamente tienes que usar http://192.168.99.100:9000/ en el browser

Gestión de contenedores

- Contenedores en ejecución

```
$ docker ps
CONTAINER ID        IMAGE                          COMMAND
CREATED             STATUS                  PORTS                    NAMES
a7a0e504ca3e        seqvence/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 a7a0e504ca3e
```

  - Borrar los ficheros del contenedor parado

```
$ docker rm a7a0e504ca3e
```

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 --name some-mysql -e
MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql
```

  - Configuración con variables de entorno:
    - **MYSQL_DATABASE, MYSQL_USER, MYSQL_PASSWORD**

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
  - Usa un único contenedor (no base de datos externa)

Ejercicio 1

- Ejecuta una web con Drupal en un contenedor docker

Ejercicio 1

- Solución

```
$ docker run -p 8081:80 drupal
```

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**
  - Compartir información entre contenedores
- 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**

https://docs.docker.com/engine/tutorials/dockervolumes/

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)

Linux y Mac

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

Windows

```
$ docker run -v "${pwd}":<container_dir> <image>
```

En Windows es posible que haya que dar permisos explícitos a la carpeta en la configuración de Docker

Volúmenes

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

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

  - Carpeta del contendor: **`/usr/share/nginx/html`**
  - Modo de solo lectura (**:ro**)
  - Abre el navegador en 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/

Volúmenes

- **Limitaciones en Docker Toolbox**
  - Docker Toolbox para Win o Mac por defecto sólo permite montar carpetas que se encuentren en la carpeta del usuario (C:/Users)
  - Se puede configurar en **VirtualBox**

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>
```

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/henrythemes/jekyll-minimal-theme
$ cd jekyll-minimal-theme
```

  - Ejecutar el contenedor para generar el sitio web en la carpeta descargada

```
$ docker run --rm -v "$PWD":/src grahamc/jekyll build
```

  - **`build`** es el parámetro del comando del contendor
  - **`--rm`** borra el contenedor al terminar su ejecución
  - El resultado se genera en la carpeta `_site`

https://hub.docker.com/r/grahamc/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**
  - Para volver a poner los permisos de los ficheros como el usuario del host se puede usar el comando

```
$ sudo chown -R username:group _site
```

https://hub.docker.com/r/grahamc/jekyll/

Ejercicio 2

- **Compilar una aplicación Java con un compilador dockerizado**
  - Maven es la herramienta usada para compilar aplicaciones Java
  - Busca una imagen de Maven en Docker Hub (usaremos el tag **3.3-jdk-8**)
  - Monta las carpetas adecuadas (para que el compilador pueda acceder al fuente y para que pueda generar el binario)

Ejercicio 2

- **Compilar una aplicación Java con un compilador dockerizado**
  - Aplicación Java de ejemplo

```
$ git clone https://github.com/codeurjc/curso-docker
$ cd curso-docker/java-webapp
```

  - Comando de compilación y empaquetado

```
$ mvn package
```

  - Resultado el fichero

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

Ejercicio 2

- **Solución**
  - Imagen **`maven:3.3-jdk-8`**
  - **`mvn package`** es el comando de compilación y empaquetado
  - **`-w`** configura el directorio de trabajo

```
$ docker run --rm -v "$PWD":/data -w /data maven:3.3-jdk-8 mvn package
```

Ejercicio 2

- **Solución**
  - Si tenemos Java instalado en el host, podemos ejecutar el fichero generado

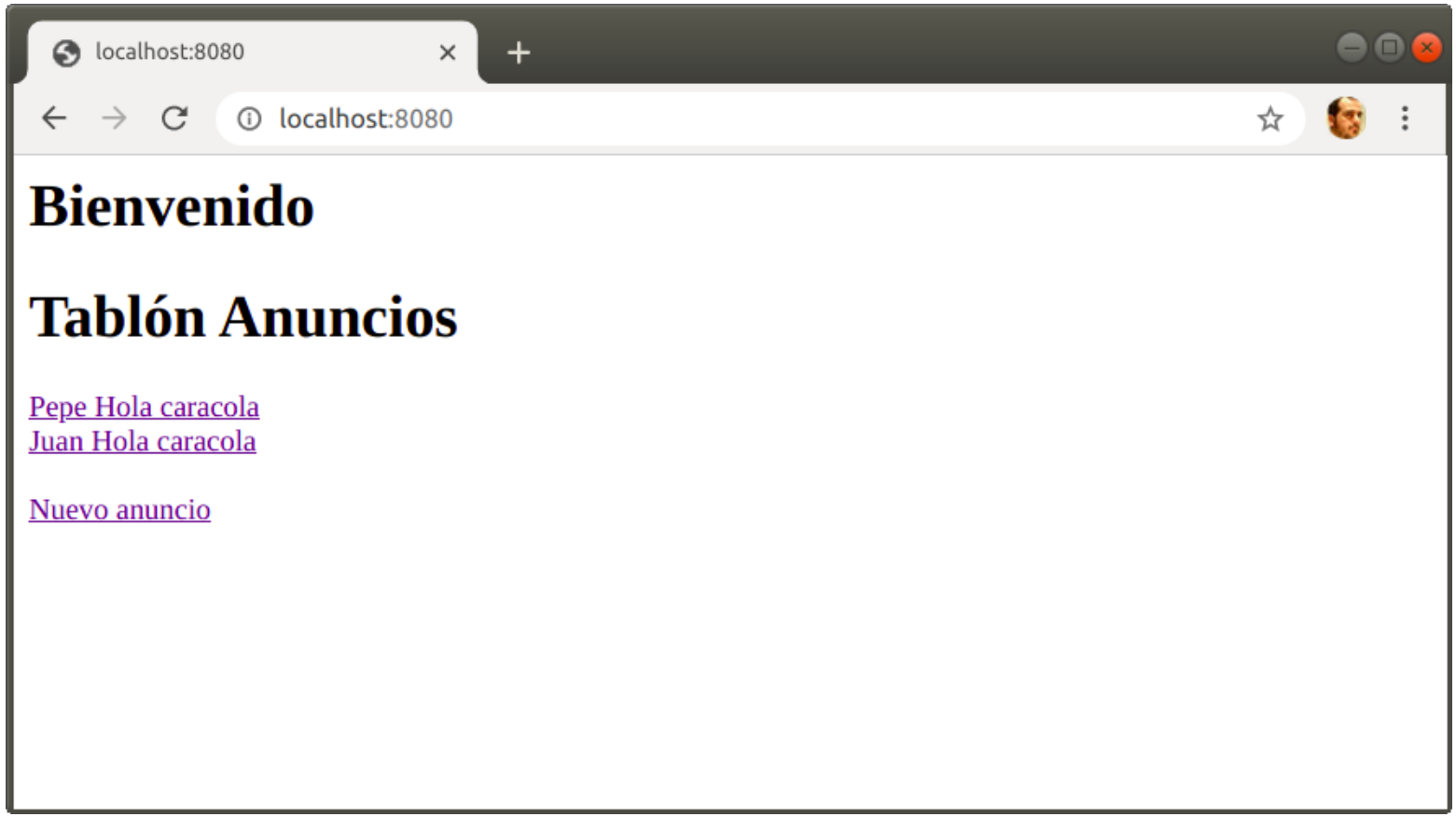
```
$ java -jar target/java-webapp-0.0.1.jar
```

  - La aplicación estará disponible en

```
http://localhost:8080
```

Ejercicio 2

- Solución



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 contendor con ubuntu

```
$ docker run -it ubuntu /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 contendor 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
  - Ejecuta la aplicación una vez compilada (recuerda bindear el puerto 8080 para que sea accesible desde el host)

Ejercicio 3

- Solución

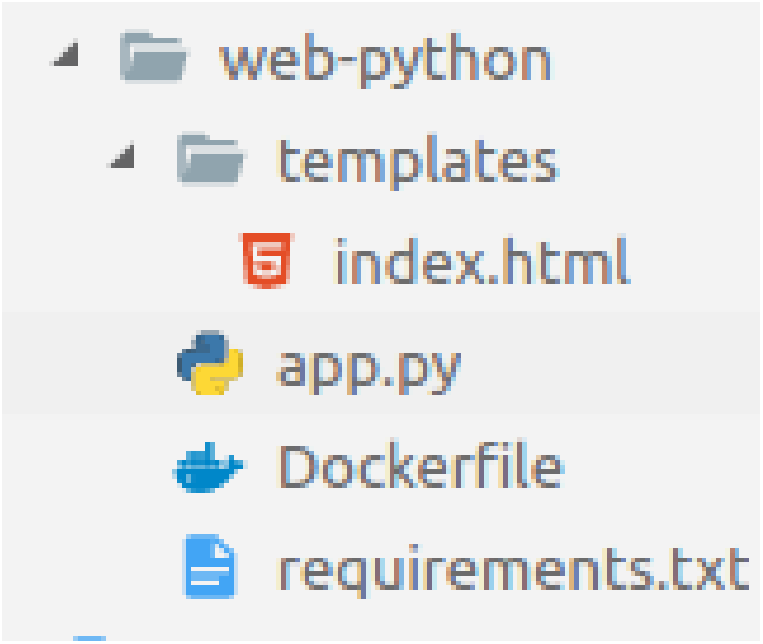
```
$ docker run -p 8080:8080 -v $PWD:/data -w /data \
 -it maven:3.3-jdk-8 /bin/sh

# mvn package

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

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 **Python**
- Descarga la web de ejemplo




```
$ git clone https://github.com/codeurjc/curso-docker
$ cd web-python
```

Dockerizar una aplicación

- **Contenido de la imagen docker:**
  - Código fuente de la aplicación
  - Python
  - Librerías necesarias (en este caso Flask)
- 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

Dockerfile

```
# Selecciona la imagen base
FROM alpine:3.5

# Instala python y pip
RUN apk add --update py-pip
RUN pip install --upgrade pip

# Copia los ficheros de la aplicación
COPY app.py /app/
COPY templates/index.html /app/templates/
COPY requirements.txt /app/

# Instala las librerías python que necesita la app
RUN pip install --no-cache-dir -r /app/requirements.txt

# Indica el puerto que expone el contenedor
EXPOSE 5000

# Comando que se ejecuta cuando se arranque el contenedor
CMD ["python", "/app/app.py"]
```

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/

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 el directorio desde el que se copian los ficheros a la imagen (comandos COPY)

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 9000:5000 miusuario/webgatos
 * Running on http://0.0.0.0:5000/
   (Press CTRL+C to quit)
```

- Abrir http://127.0.0.1:9000/ en el navegador web
- Usuarios de Docker Toolbox en Windows y Mac deberían usar la IP de la VM en vez de localhost

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

- **Caché de construcción**
  - Por cada comando COPY o RUN se crea una capa (layer)
  - En una nueva construcción, por cada comando se verifica si se puede reutilizar la capa de la ejecución anterior (caché):
    - **COPY**: Se reutiliza si los ficheros no cambian
    - **RUN**: Se reutiliza si el comando no cambia
  - El uso de la caché de una caché es muy rápido
  - Si hay que **repetir un comando**, los comandos **posteriores se tienen que repetir** aunque no hayan cambiado porque cambia su contexto y el resultado podría ser otro

Dockerizar una aplicación

- **Caché de construcción por capas**
  - Cambia la plantilla de la web en **`templates\index.html`**
  - Construye la imagen de nuevo

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

  - Verifica que se utiliza la caché de los comandos anteriores a

```
COPY templates/index.html /app/templates/
```

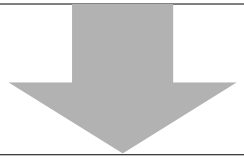
  - Verifica que los siguientes comandos se ejecutan (aunque no cambien)
  - La nueva imagen se crea más rápidamente
  - Pero se puede optimizar algo más

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

```
# Copia los ficheros de la aplicación
COPY app.py /app/
COPY templates/index.html /app/templates/
COPY requirements.txt /app/

# Instala las librerías python que necesita la app
RUN pip install --no-cache-dir -r /app/requirements.txt
```



```
# Copia el fichero de librerías
COPY requirements.txt /app/

# Instala las librerías python que necesita la app
RUN pip install --no-cache-dir -r
   /app/requirements.txt

# Copia el resto de ficheros de la aplicación
COPY app.py /app/
COPY templates/index.html /app/templates/
```

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

Ejercicio 4

- **Dockeriza una aplicación web Java**
  - Basada en una imagen con Maven para poder compilar la aplicación en el proceso de construcción de la imagen

```
$ git clone https://github.com/codeurjc/curso-docker
$ cd curso-docker/java-webapp
```

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

Ejercicio 4

- **Dockeriza una aplicación web Java**
  - Crea un Dockerfile
    - Imagen base
    - Copia del código de la aplicación a la imagen
    - Comandos de compilación y descarga de librerías de la aplicación
    - Definición de puerto
    - Comando de ejecución del contenedor

Ejercicio 4

- **Dockeriza una aplicación web Java**
  - Imagen base

```
maven:3.3-jdk-8
```

  - Comando de compilación (en la carpeta del código)

```
$ mvn package
```

  - Comando de ejecución (en la carpeta del código)

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

  - Puerto: **8080**

Ejercicio 4

- Solución

maven.dockerfile

```
# Selecciona la imagen base
FROM maven:3.3-jdk-8

# Copia el código del proyecto
COPY /src /project/src
COPY pom.xml /project/

# Define el directorio de trabajo donde ejecutar comandos
WORKDIR /project

# Compila proyecto y descarga librerías
RUN mvn package

# Indica el puerto que expone el contenedor
EXPOSE 8080

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

```
$ docker build -f maven.dockerfile -t miusuario/java-webapp-maven .
```

Ejercicio 4

- **Solución**
  - Con Maven no es posible separar completamente la descarga de librerías de la compilación del proyecto.
    - Existen algunas estrategias, pero sólo funcionan parcialmente
  - Por tanto, no nos podemos aprovechar fácilmente de las caché de las capas con este enfoque
  - Veremos otro enfoque de construcción de Java más adecuado

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

- **Problema de compilar en el Dockerfile**
  - Si compilamos en el proceso de construcción de la imagen docker (Dockerfile) el compilador (Maven) acabará en la imagen final, incrementando su tamaño
  - Una app Java dockerizada con esta estrategia ocupa 667MB

```
$ docker image ls miusuario/java-webapp-maven
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
miusuario/java-webapp-maven   latest              19659ba4a2cf        42 minutes ago      667MB
```

Dockerizar una aplicación compilada

- **Solución: Compilar "fuera" del Dockerfile y empaquetar sólo el fichero compilado (.jar)**
  - Compilar y generar el fichero .jar

```
$ docker run --rm -v "$PWD":/data -w /data maven mvn package
```

  - Crear un Dockerfile que copie el .jar dentro del contenedor
  - La imagen base sólo necesita el runtime de Java (sin Maven)

```
openjdk:8-jre
```

Dockerizar una aplicación compilada

- **Solución: Compilar "fuera" del Dockerfile y empaquetar sólo el fichero compilado (.jar)**
  - jar.dockerfile

```
FROM openjdk:8-jre
COPY target/*.jar /usr/app/
WORKDIR /usr/app
CMD [ "java", "-jar", "java-webapp-0.0.1.jar" ]
```

  - Construir el contenedor

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

  - Tamaño

```
$ docker image ls miusuario/java-webapp-jar
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
miusuario/java-webapp-jar   latest              81536175ee23        14 seconds ago      278MB
```

Dockerizar una aplicación compilada

- **Diferencias de tamaño**
  - Incluir Maven en la imagen final

667MB

  - Sólo incluir el JRE de Java y el .jar

278MB

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

```
FROM maven:3.3-jdk-8 as builder
COPY /src /project/src
COPY pom.xml /project/
WORKDIR /project
RUN mvn package

FROM openjdk:8-jre
COPY --from=builder /project/target/*.jar /usr/app/
WORKDIR /usr/app
CMD [ "java", "-jar", "java-webapp-0.0.1.jar" ]
```

```
$ docker build -f multistage.dockerfile -t miusuario/java-webapp-ms .
```

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

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
- La imagen está **optimizada para ejecutar aplicaciones Java** (*distroless*)

https://github.com/GoogleContainerTools/distroless

https://github.com/GoogleContainerTools/jib

Google jib

pom.xml

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

    ...

    <build>
        <plugins>
            <plugin>
                <groupId>com.google.cloud.tools</groupId>
                <artifactId>jib-maven-plugin</artifactId>
                <version>1.6.1</version>
            </plugin>
        </plugins>
    </build>
</project>
```

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

https://github.com/GoogleContainerTools/jib/blob/master/examples/spring-boot/pom.xml

Ejercicio 5

- Crea la imagen de la aplicación Java del Ejercicio 4 con jib

Desarrollo con contenedores

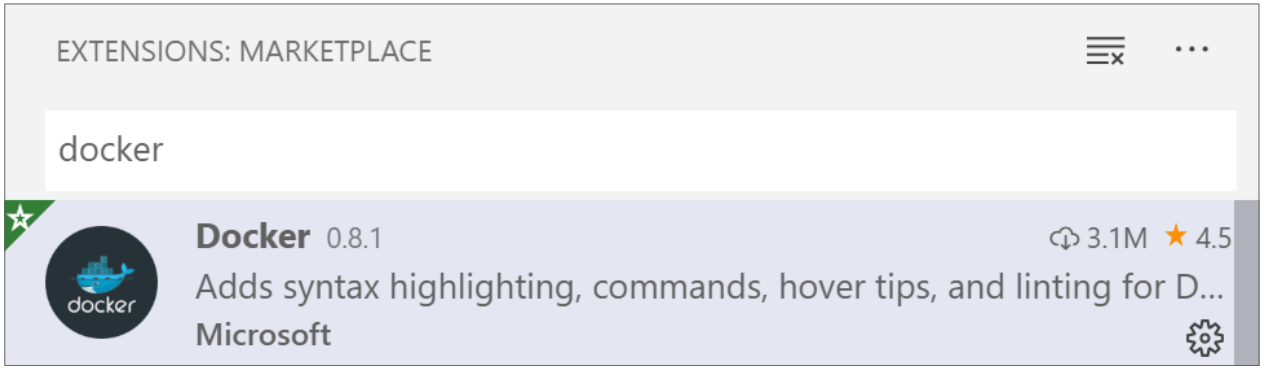
- **Ventajas**
  - Evita que varios desarrolladores tengan diferentes versions de las herramientas (Java, maven, MySQL…)
  - Muy sencillo probar el código con diferentes versiones de Java, Base de datos, etc.
  - Reduce el tiempo de setup inicial de los entornos, son muy reproducibles porque están en 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

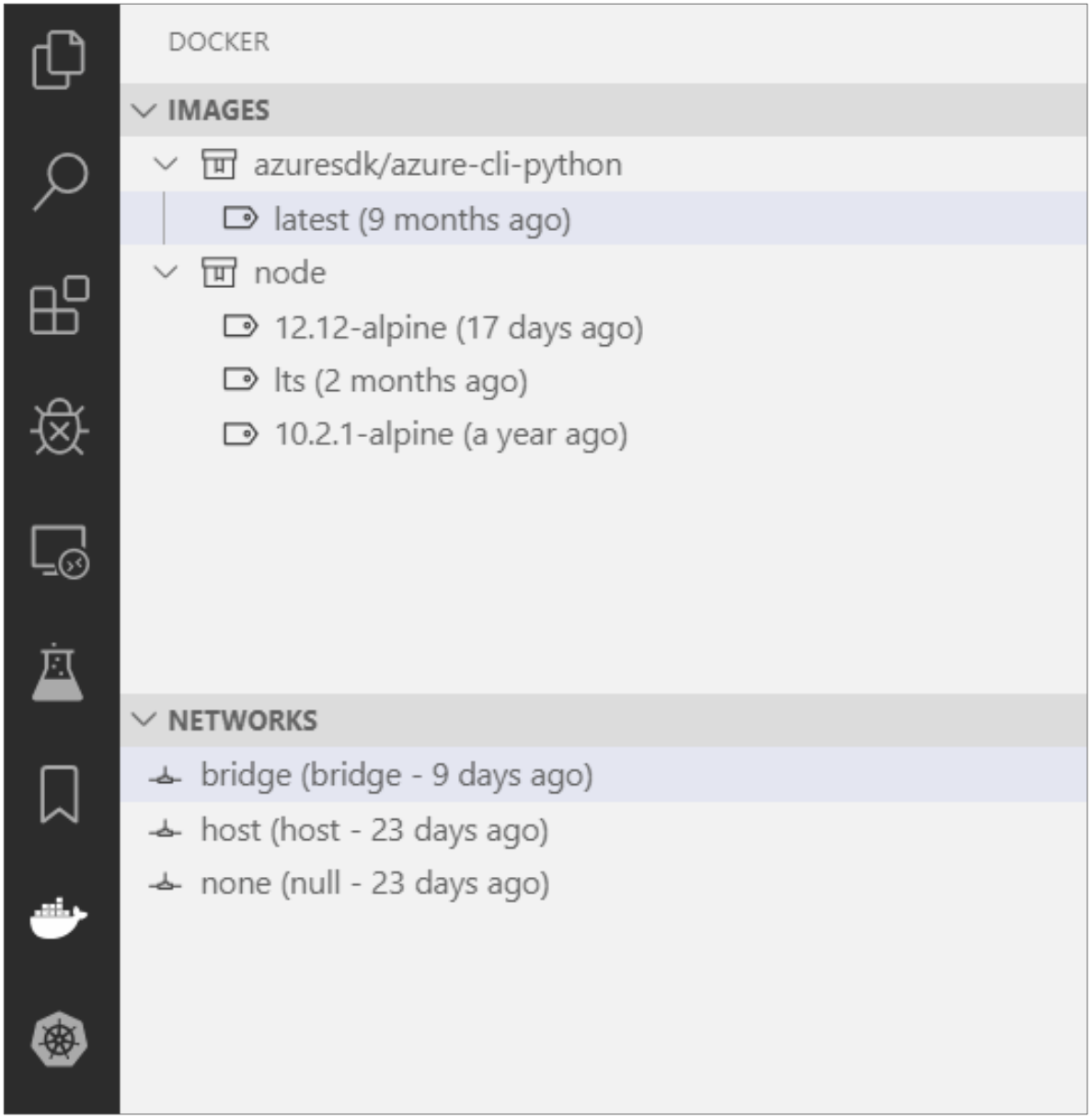
Desarrollo con contenedores

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



https://code.visualstudio.com/docs/containers/overview

Desarrollo con contenedores



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

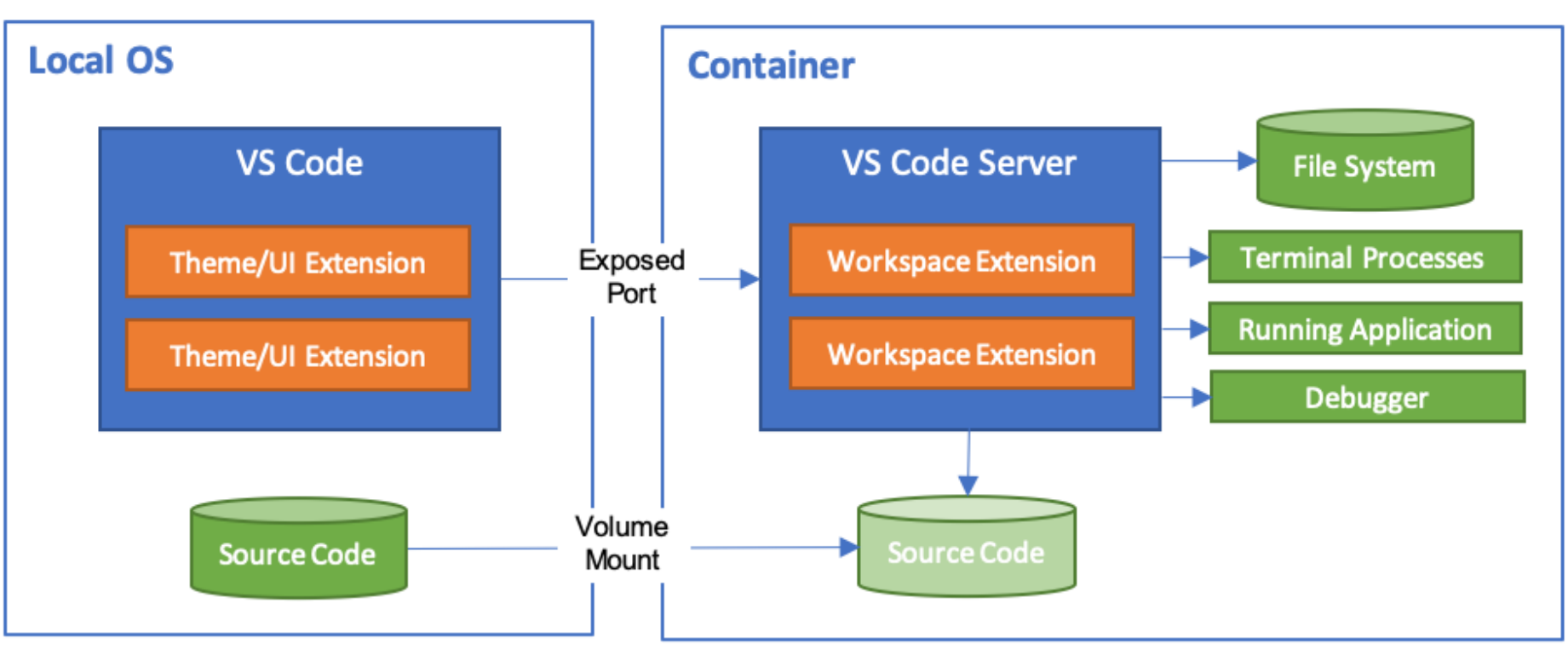
Desarrollo con contenedores

Visual Studio Code Remote - Containers

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

Desarrollo con contenedores

- VSCode remote - Containers

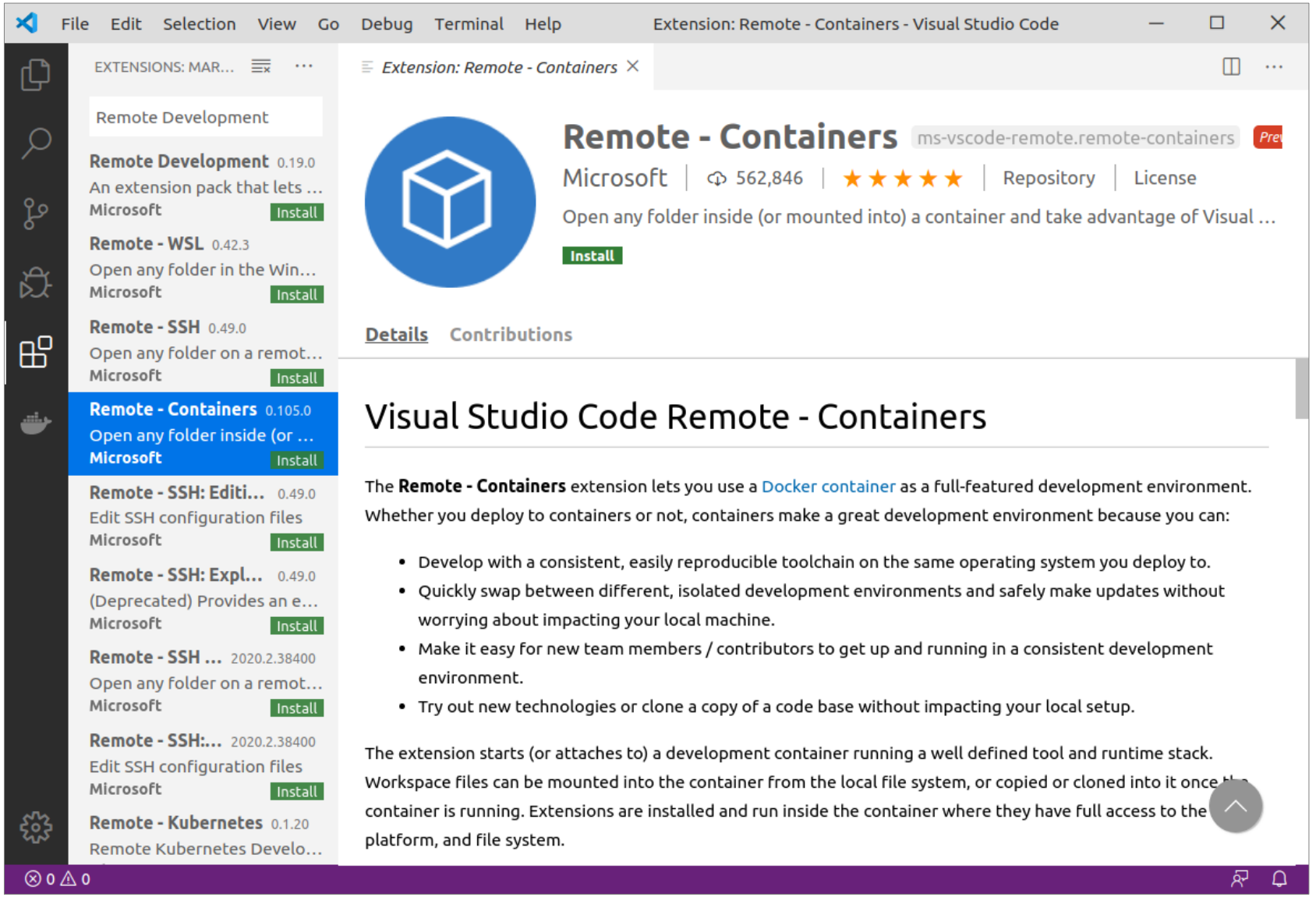


Desarrollo con contenedores

- **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

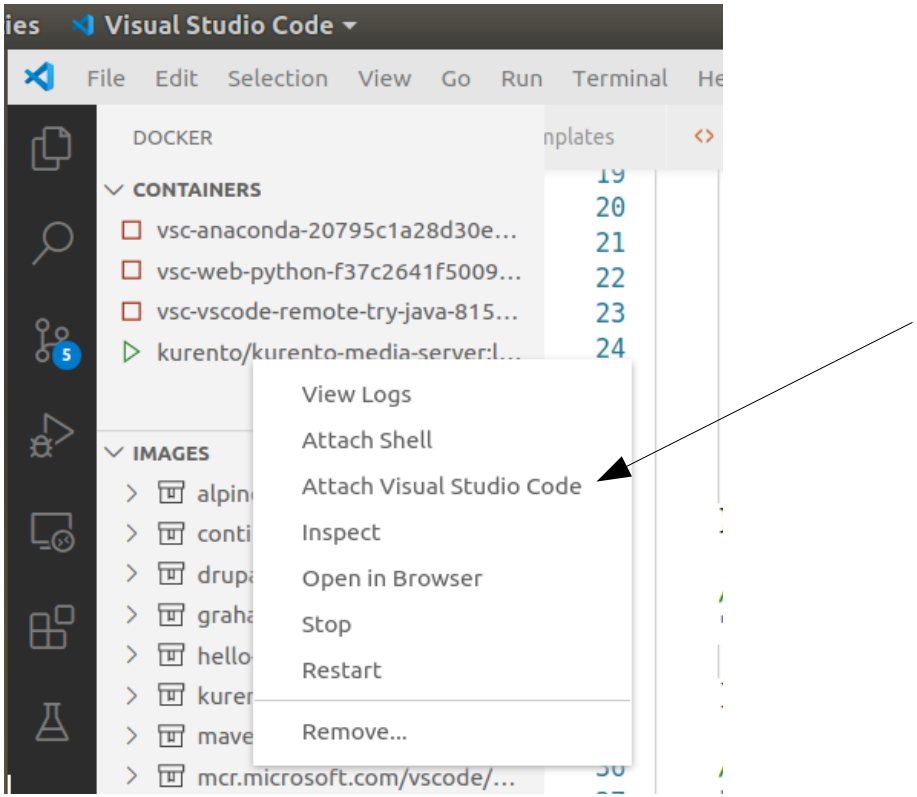


VSCode Remote - Containers

- **Modos de funcionamiento**
  - **Conectarse a un contenedor arrancado**
    - Navegar por sus carpetas. Leer y editar ficheros
    - Ejecutar comandos
  - **Desarrollar dentro del contenedor**
    - Usar el compilador y las tools del contenedor
    - Se pueden definir plugins VSCode específicos para ese contenedor

VSCode Remote - Containers

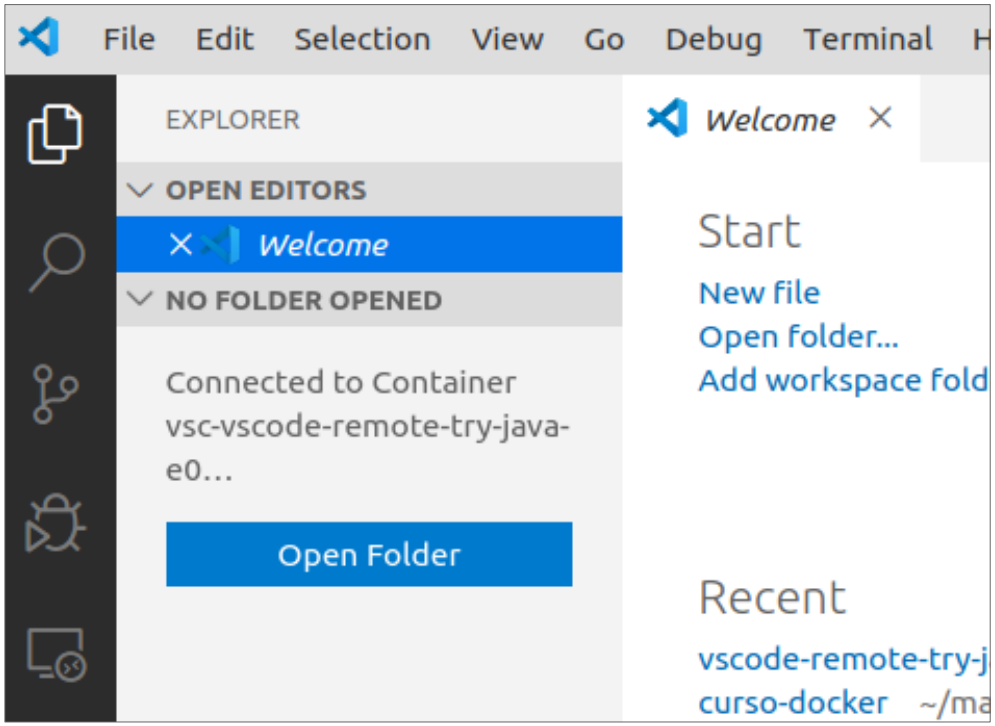
- **Conectarse a un contenedor arrancado**
  - Container > Attach Visual Studio Code



https://code.visualstudio.com/docs/remote/containers#_attaching-to-running-containers

VSCode Remote - Containers

- **Conectarse a un contenedor arrancado**
  - Se pueden ver y editar sus ficheros y ejecutar comandos



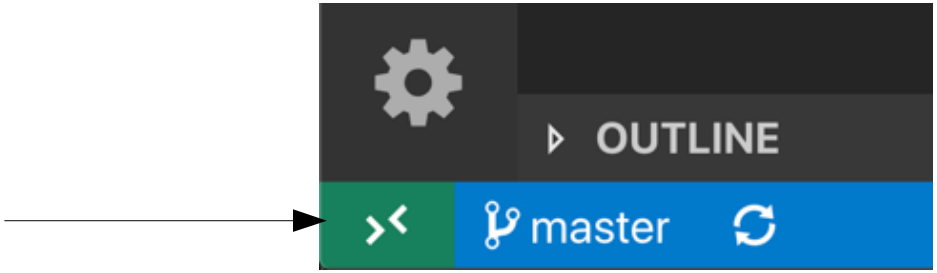
https://code.visualstudio.com/docs/remote/containers#_attaching-to-running-containers

VSCode Remote - Containers

- **Desarrollar dentro del contenedor**
  - 1) Repositorio de ejemplo para desarrollo Java

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

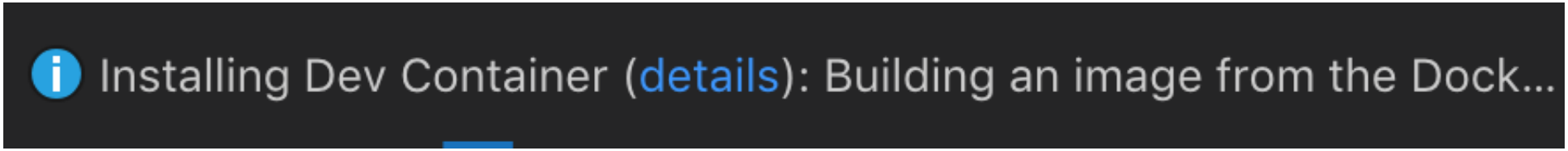
  - 2) Click en el icono de desarrollo remoto



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

VSCode Remote - Containers

- **Desarrollar dentro del contenedor**
  - Se crea el contenedor de desarrollo
    - Dockerfile: Compilador, tools…
    - .devcontainer: Plugins del IDE, puertos bindeados, etc



  - La primera vez tarda (descarga y creación)

VSCode Remote - Containers

- **Desarrollar dentro del contenedor**
  - Existen configuraciones predefinidas para desarrollar en cualquier tecnología (Node, Python, Java…).
  - La configuración se guarda en la carpeta del proyecto (y se sube al repositorio) para compartir