

Virtualización y Contenedores Docker

Virtualización y Contenedores Docker

1. Virtualización
2. Docker
3. Docker Compose

Virtualización

- Los desarrolladores quieren reducir las diferencias entre el entorno de desarrollo local (portátil), los servidores de integración continua y los sistemas de producción
- Quieren evitar los problemas de tipo “Funciona en mi máquina”

Virtualización

Contenedores

Virtualización

- **Virtualización**

- Máquina virtual con interfaz gráfica (VirtualBox)
- Máquina virtual tipo servidor, diseñada para servidores (Vagrant)

- **Contenedores**

- Docker



VirtualBox

VirtualBox

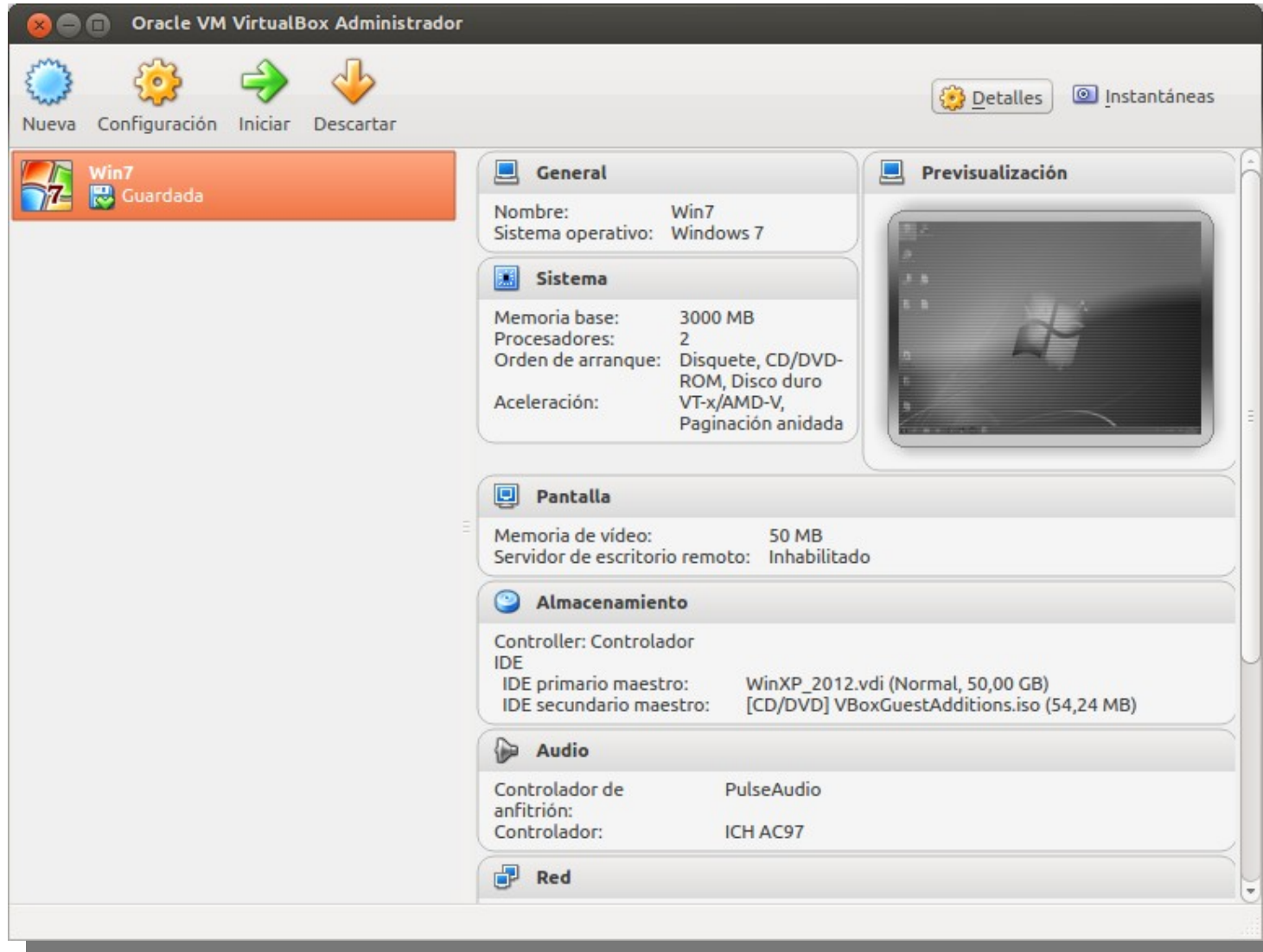
- Desarrollado por Oracle
- Software libre (con módulos gratuitos, pero no libres)
- Versiones para Windows, Linux y Mac
- Virtualización avanzada de escritorio
 - Compartir carpetas entre la máquina anfitriona y la VM invitada
 - Integración de teclado y ratón
 - Aceleración gráfica 3D
 - Cámara web

<https://www.virtualbox.org/>



VirtualBox

VirtualBox



VirtualBox

- Configuración interactiva

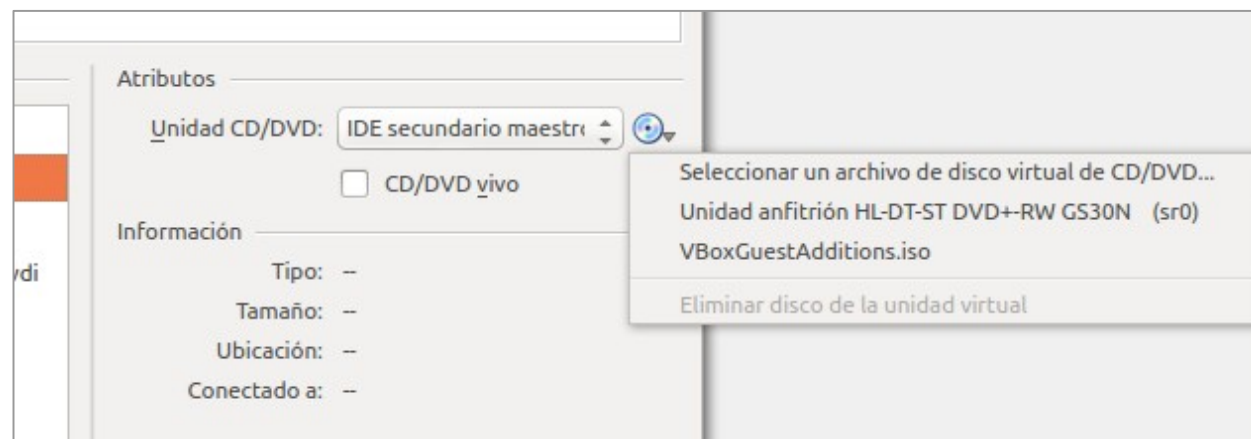
- Pasos:

- Crear una máquina virtual limpia
 - Conecta una imagen ISO (simulando un CD real)
 - Instala un sistema operativo completo
 - Consume tiempo y no es sencillo compartir la configuración de las máquinas virtuales

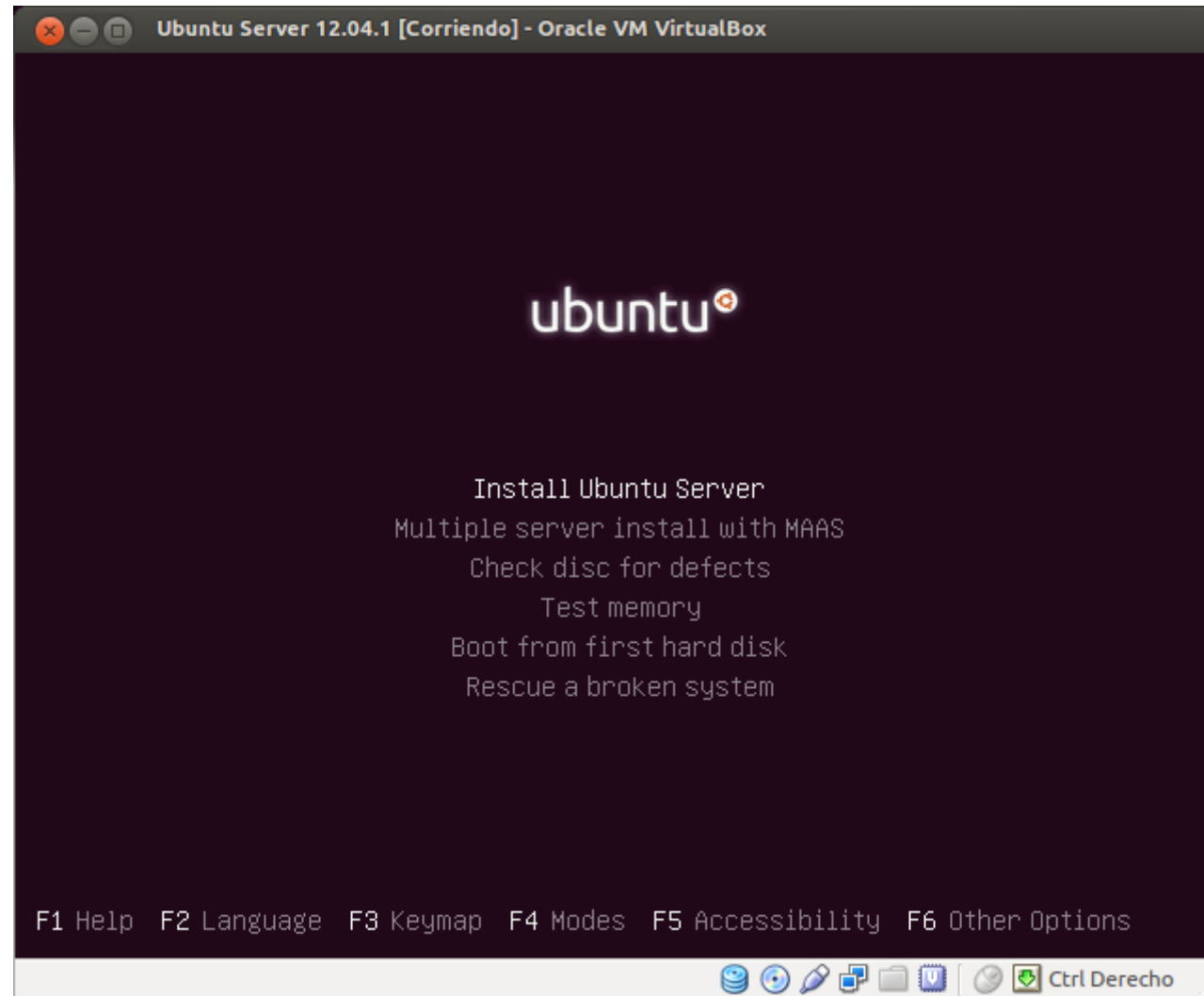


VirtualBox

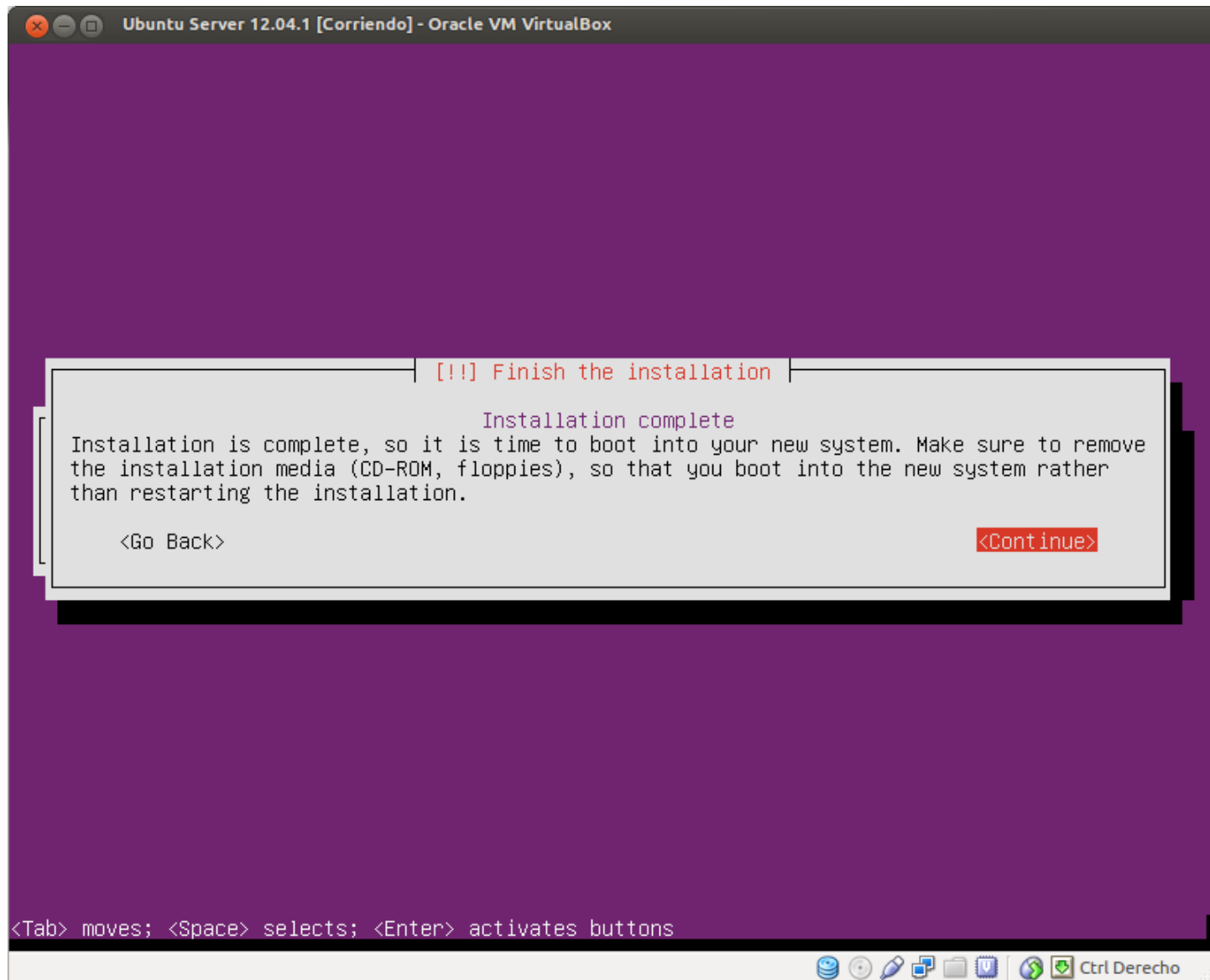
VirtualBox



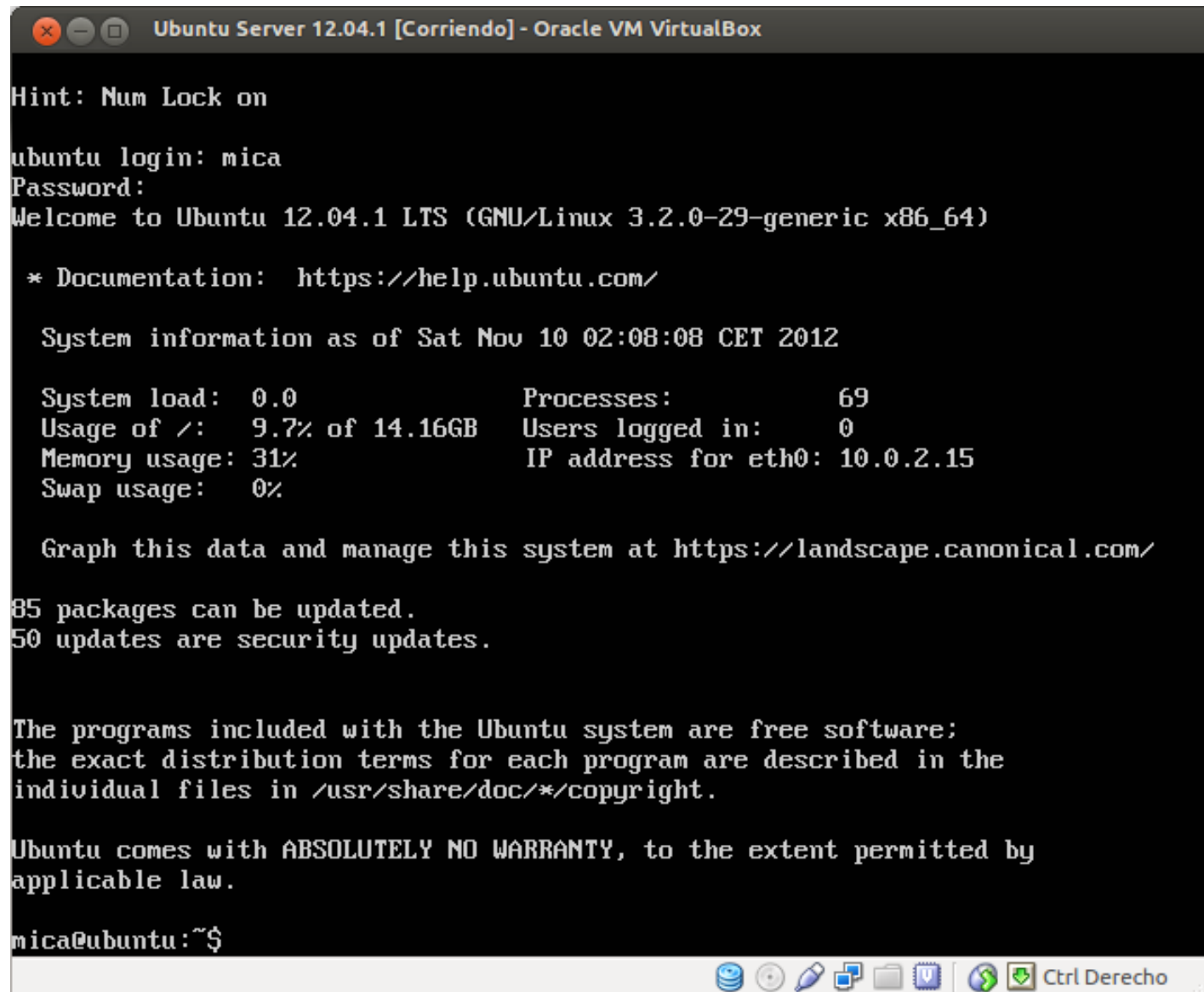
VirtualBox



VirtualBox



VirtualBox



```
Ubuntu Server 12.04.1 [Corriendo] - Oracle VM VirtualBox

Hint: Num Lock on

ubuntu login: mica
Password:
Welcome to Ubuntu 12.04.1 LTS (GNU/Linux 3.2.0-29-generic x86_64)

* Documentation:  https://help.ubuntu.com/

System information as of Sat Nov 10 02:08:08 CET 2012

System load:  0.0                Processes:            69
Usage of /:   9.7% of 14.16GB    Users logged in:     0
Memory usage: 31%               IP address for eth0: 10.0.2.15
Swap usage:   0%

Graph this data and manage this system at https://landscape.canonical.com/

85 packages can be updated.
50 updates are security updates.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

mica@ubuntu:~$
```

VirtualBox

- Para parar una VM:
 - Simulando el apagado de la máquina física
 - Enviado la señal de apagado
 - Haciendo que el sistema se pare (o reinicie) a sí mismo
- `sudo poweroff`
- `sudo halt`
- `sudo shutdown -h now`
- `sudo reboot`
 - Pausando para reanudar después



VAGRANT

Vagrant



- **Vagrant** es una herramienta pensada para desarrolladores que quieren configurar y compartir el entorno de trabajo (desarrollo) o despliegue de su aplicación
- Está basado en hypervisores como **VirtualBox**, **VMWare** o proveedores cloud como **AWS**

<https://www.vagrantup.com>

Vagrant



- Para **adaptar** una máquina (box) a las necesidades del proyecto se pueden usar:
 - Script de shell
 - Herramientas de alto nivel para provisionar software: Chef, Puppet, Ansible...

<https://www.vagrantup.com>

Vagrant

- Instalar Vagrant
 - Tener instalador VirtualBox (o cualquiera de los soportados)
 - Instalar Vagrant
- <https://www.vagrantup.com/docs/installation/>

Vagrant

- Crear una máquina virtual

```
$ mkdir project
$ cd project
$ vagrant init ubuntu/trusty64
$ vagrant up
```

- Conectarse a la VM por ssh

```
$ vagrant ssh
```

Vagrant

- Crear una máquina virtual

```
$ vagrant init ubuntu/trusty64
```

- Se genera un fichero **Vagrantfile** que describe la máquina virtual basada en la “**box**” de ubuntu trusty de 64bits publicada en el repositorio

<https://atlas.hashicorp.com/bento/boxes/ubuntu-16.04>

- Cualquiera puede crear una cuenta y subir sus propias **boxes** con las configuraciones necesarias

Vagrant

- Manejar la nueva máquina virtual

- Las máquinas se gestionan enteramente desde la **línea de comandos** (arrancar, parar, reanudar...)
- **Arrancar** la máquina virtual

```
$ vagrant up
```

- Puede tardar bastante tiempo:
 - Es posible que tenga que **descargar el binario** del box si no está disponible en la máquina
 - Las VMs pueden tardar **minutos en arrancar**

- **Manejar la nueva máquina virtual**

- Detener la ejecución de la máquina virtual pero mantener el estado (disco duro)

```
$ vagrant halt
```

- Destruir todos los ficheros de la máquina virtual

```
$ vagrant destroy
```

- Pausar la VM (mantiene la memoria):

```
$ vagrant pause
```

- Reanudar la VM pausada

```
$ vagrant resume
```

- **Manejar la nueva máquina virtual**

- La máquinas **no tienen interfaz gráfico**, sólo pueden usarse mediante una conexión **ssh** (lo habitual en el cloud).

```
$ vagrant ssh
```

- La conexión ssh se realiza con una **clave privada** (en vez de con contraseña). En la imagen de ubuntu oficial se genera una clave de forma **automática** para conectar
- Para cerrar la conexión ssh y volver a la shell del SO host:

```
vagrant@vagrant-ubuntu-trusty-64:~$ exit
```

- Configuración de red en la máquina virtual

- Para acceder a la máquina virtual por red se descomenta la siguiente línea de **Vagrantfile**

```
# config.vm.network "private_network", ip: "192.168.33.10"
```

- Verificar que la máquina arranca con ip 192.168.33.10 y tiene conexión a Internet

```
$ vagrant up
$ ping 192.168.33.10
$ vagrant ssh
ubuntu> ping www.google.es
ubuntu> exit
$ vagrant destroy -f
```

- Ejecutar aplicaciones en la máquina virtual
 - La carpeta en la que se encuentra el Vagrantfile es accesible directamente desde la máquina virtual en la ruta **/vagrant**
 - Un flujo de desarrollo puede ser copiar el binario de la aplicación en la carpeta del host para que esté accesible desde la máquina virtual

- Ejecutar una app web dentro de una VM
 - Copiar webapp.jar en la carpeta del fichero Vagrantfile
 - Iniciar la VM y arrancar la app

```
$ vagrant up
$ vagrant ssh
ubuntu> sudo add-apt-repository ppa:webupd8team/java
ubuntu> sudo apt-get update
ubuntu> sudo apt-get install oracle-java8-installer
ubuntu> cd /vagrant
ubuntu> java -jar java-webapp-0.0.1.jar
```

- Abrir <http://192.168.33.10:8080> en un browser
- Para la app y la VM

```
ubuntu> Ctrl+C
ubuntu> exit
$ vagrant destroy -f
```

Virtualización y Contenedores Docker

1. Virtualización
- 2. Docker**
3. Docker Compose

- 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

Docker



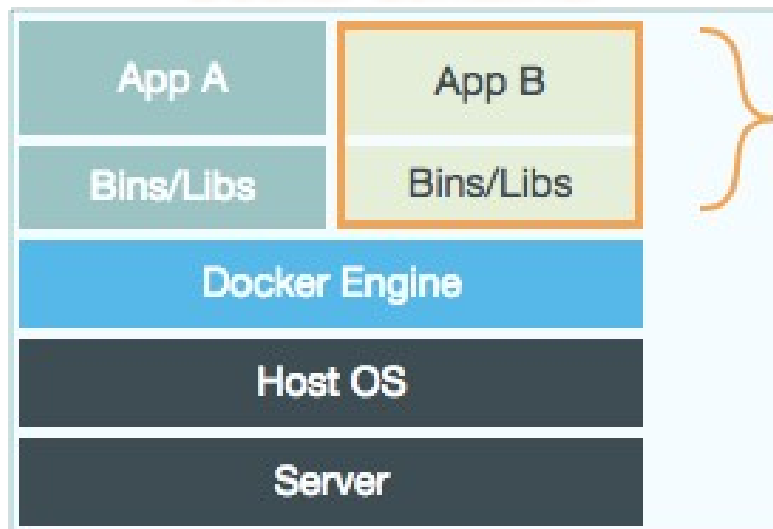
- Es la tecnología de contenedores **más popular** (creada en 2013)
- Inicialmente desarrollada para **linux**, aunque dispone de herramientas para **desarrolladores en windows y mac**
- Existe un **repositorio de imágenes públicas** (hub)

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

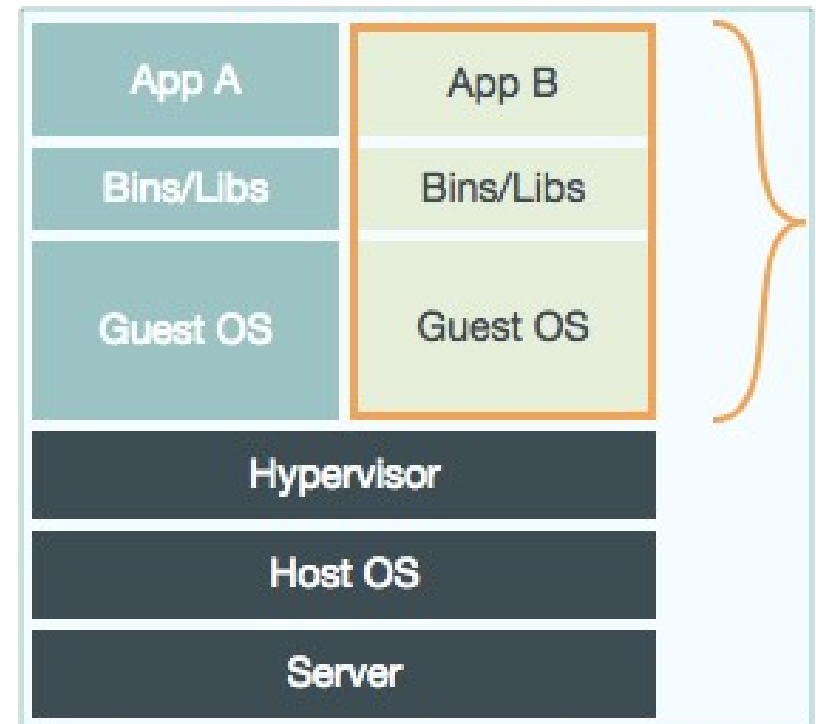
- ¿Por qué son tan eficientes los contenedores?
 - Para ejecutar un contenedor no se necesita **hypervisor** porque **no se ejecuta** un sistema operativo invitado y no hay que simular HW
 - Un contenedor es un **paquete** que contiene una **app** y todo el sw necesario para que se ejecute (python, Java, 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**

Docker

Docker Container



Virtual Machine



Principales diferencias

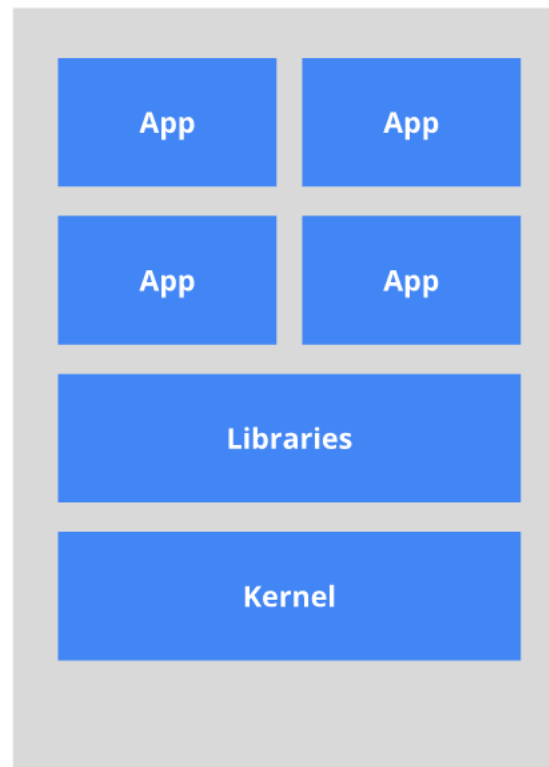
Máquinas Virtuales	Contenedores
Más pesadas	Más ligeras
Varios procesos	Optimizadas para un único proceso (aunque pueden tener varios)
Conexión por ssh (aunque esté en local)	Acceso directo al contenedor
Más seguridad porque están más aisladas del host	Potencialmente menor seguridad porque se ejecutan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

- **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

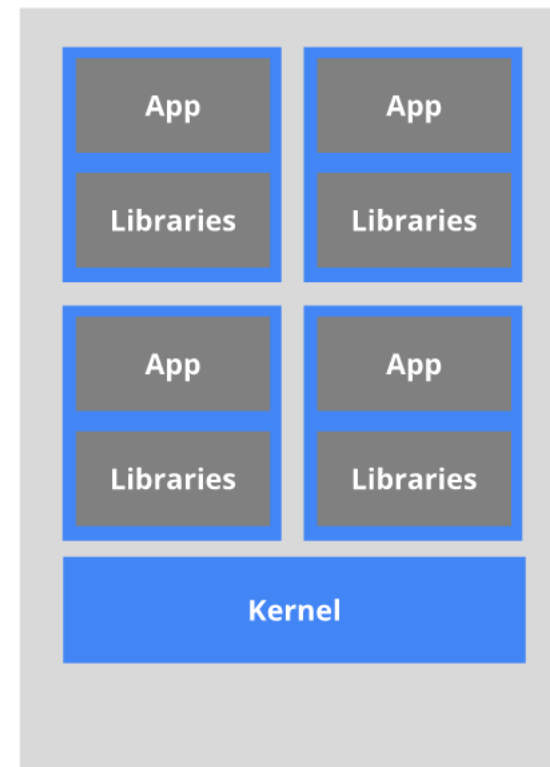
- Formato de distribución y ejecución de servicios con Docker

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*

- ¿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**

- **Tipos de aplicaciones:**

- Aplicaciones **de red:**

- Web, bbdd, colas de mensajes, cachés, etc.

- Aplicaciones **de línea de comandos:**

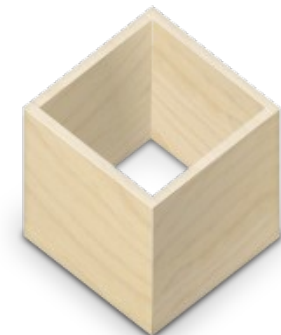
- Compiladores, generadores de sitios web, conversores de vídeo, generadores de informes...

Docker

- Tipos de aplicaciones:

- Aplicaciones **gráficas**:

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



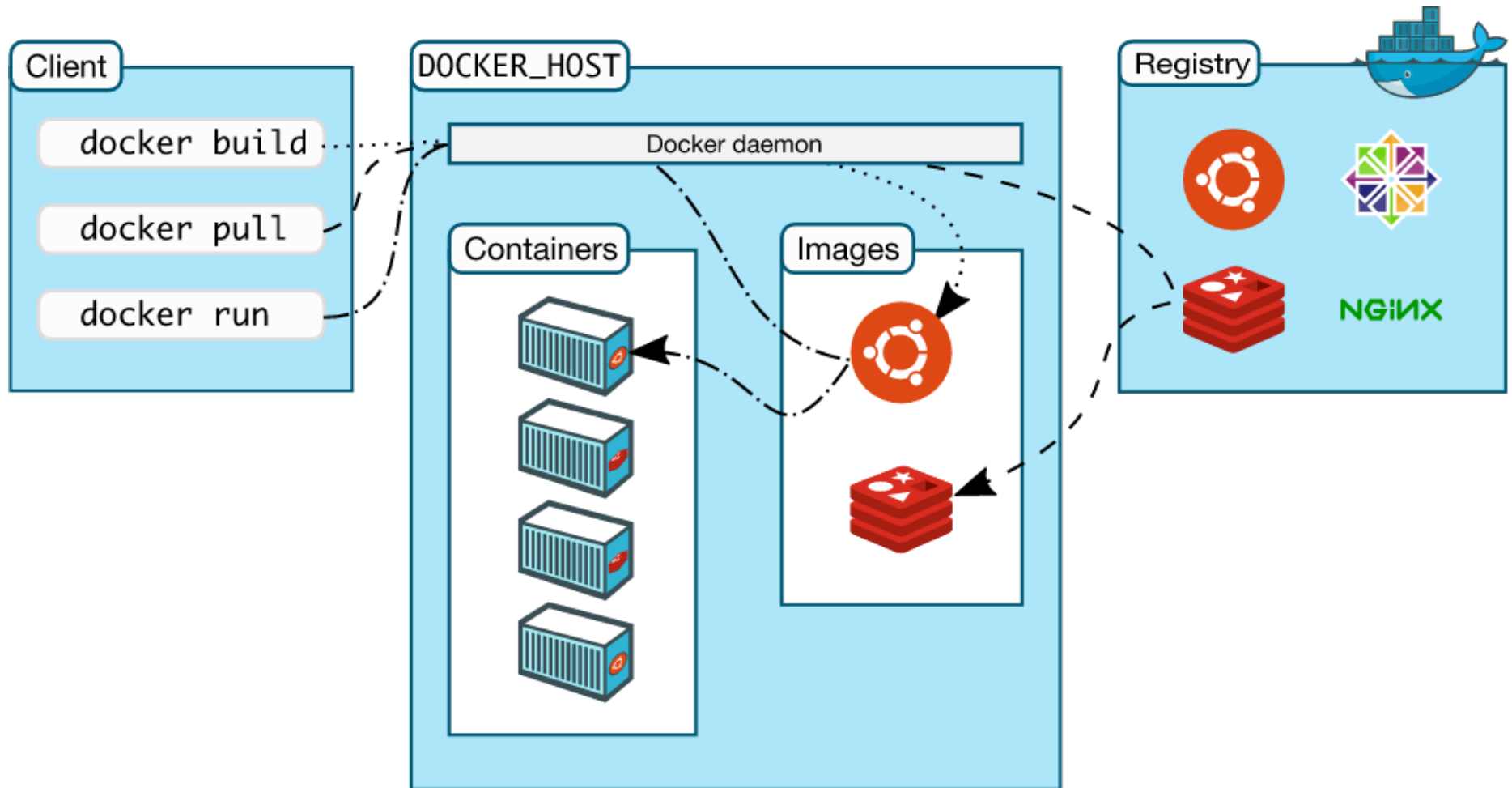
FLATPAK

<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 gestionadas por docker
 - Contenedores **windows**
 - Menos usados y menos maduros
 - Sólo se pueden ejecutar en windows server

Conceptos Docker



Conceptos Docker

- **Imagen docker**

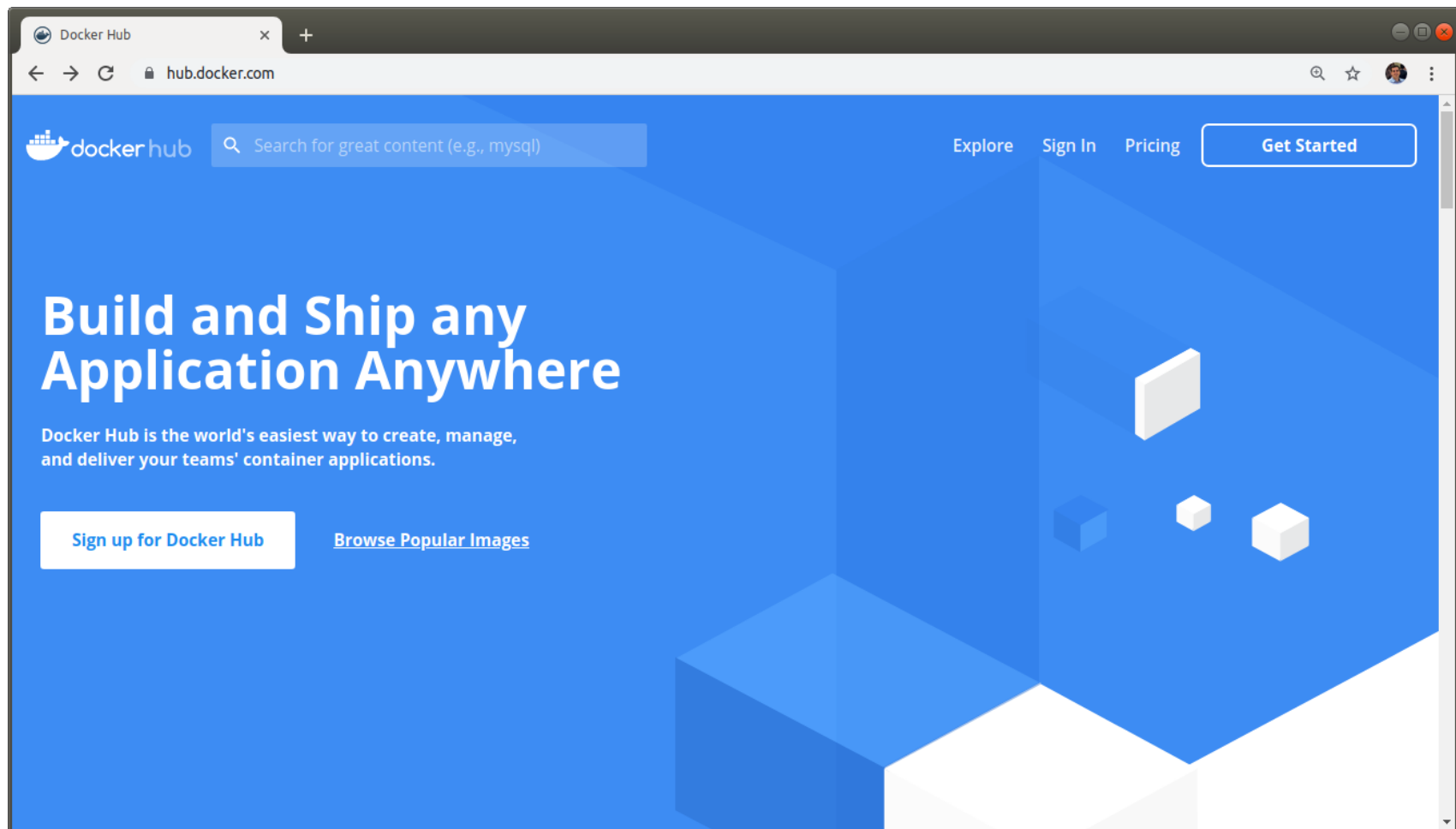
- Ficheros a los que tendrá acceso el contenedor cuando se ejecute.
- Herramientas/librerías de una distribución linux excepto el kernel (**ubuntu, alpine**), runtime de ejecución (**Java**) y la aplicación en sí (**webapp.jar**)
- Un contenedor siempre se inicia desde una **imagen**
- Si se quiere arrancar 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

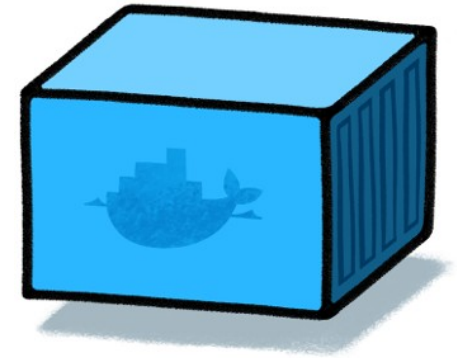


Relational database management system



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

•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:

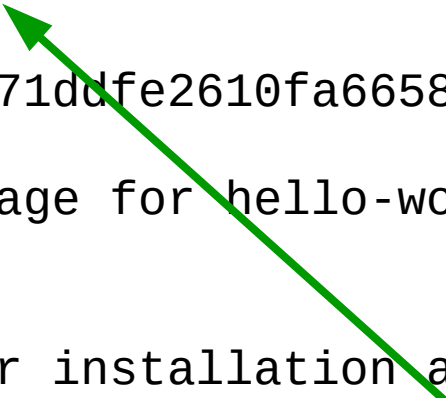
- Apple Mac OS Yosemite 10.10.3 or above:
<https://store.docker.com/editions/community/docker-ce-desktop-mac>
- Older Mac: <https://www.docker.com/products/docker-toolbox>

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.
...
```

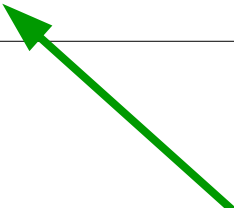


La primera vez la imagen se descarga

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 images
```

```
$ docker images
```

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 images
```

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
...				

- 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
```

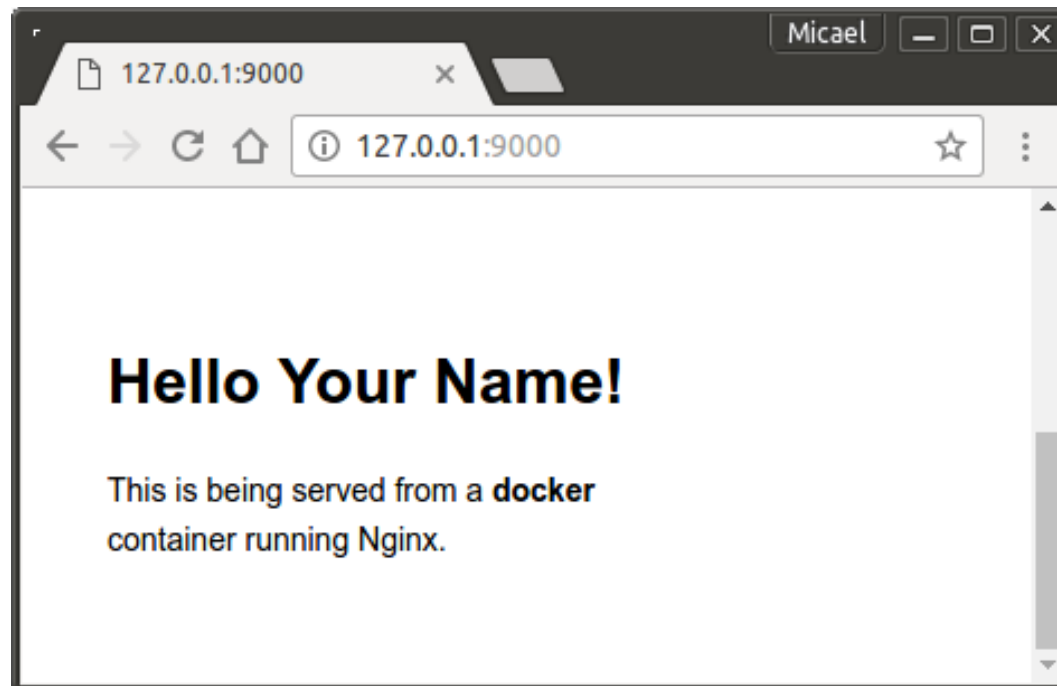
-p 9000:80

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



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 la **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       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 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

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

- Carpeta del contenedor: `/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

- **Genera el .jar de una aplicación con un contenedor docker**
 - 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)

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 /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
```

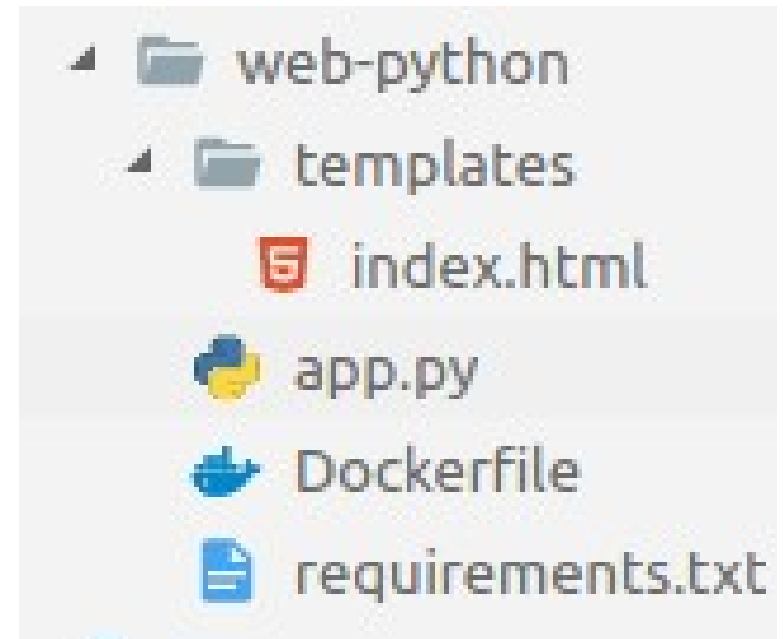
- Es posible ejecutar un **comando** dentro de un contenedor en ejecución
- Se suele ejecutar una shell para poder inspeccionar el **sistema de ficheros** del contenedor para **depurar** problemas

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

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

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

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

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

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

# Indica el puerto que expone el contenedor
EXPOSE 5000

# Comando que se ejecuta cuando se arranque el contenedor
CMD ["python", "/usr/src/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 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 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 capas**

- Cambia la plantilla de la web en **templates\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

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

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

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

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

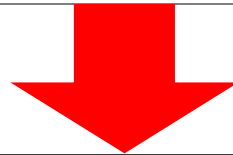
# Copia el resto de ficheros de la aplicación
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

# Indica el puerto que expone el contenedor
EXPOSE 5000

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

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

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



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

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

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

- Crea una imagen docker con una aplicación web Java
 - Basada en una imagen con Maven para poder compilar la aplicación

Ejercicio 5

- Optimiza las capas de la imagen del ejercicio anterior para no tener que descargar las dependencias cada vez que se construye la imagen
 - Comandos útiles:
 - `mvn dependency:go-offline`
 - `mvn -o install`

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, es necesario **compilar previamente el código antes de crear el contenedor**

Dockerizar una aplicación compilada

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

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

- Compilar y generar el fichero .jar

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

- 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 -t miusuario/java-webapp .
```

- Ejecutar el contenedor

```
$ docker run -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

Dockerfile.1

```
FROM maven as builder
COPY . /code/
WORKDIR /code
RUN mvn package

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

```
$ docker build -f Dockerfile.1 -t miusuario/java-webapp2 .
```

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

Ejercicio 6

- Crea un Multistage Docker file optimizando las capas para no descargar las librerías en cada construcción

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
```


Ejercicio 7

- Crea la imagen de la aplicación Java del Ejercicio 6 con jib

Virtualización y Contenedores Docker

1. Virtualización
2. Docker
- 3. Docker Compose**

Docker Compose

- Es una herramienta para definir aplicaciones formadas por **varios contenedores**
- Un fichero YAML define los **contenedores** (imagen, puertos, volúmenes...) y cómo se **relacionan** entre sí
- Los contenedores se comunican:
 - Protocolos de red
 - Volúmenes compartidos

Docker Compose

- La herramienta docker-compose tiene que **instalarse** por separado en linux (no viene incluida con docker)
- El fichero YAML se suele llamar

docker-compose.yml

- En la carpeta donde está el fichero, la aplicación se ejecuta con el comando

```
$ docker-compose up
```

Docker Compose

- Definición de cada contenedor
 - Imagen
 - Puede descargarse de DockerHub
 - Puede estar construida localmente
 - Puede construirse con un Dockerfile en el momento de iniciar la aplicación

Docker Compose

- **Definición de cada contenedor**
 - **Puertos:**
 - Mapeados en el host (para ser usados desde localhost)
 - No mapeados (sólo se pueden conectar otros contenedores)
 - **Volúmenes:**
 - Carpetas del host accesible desde el contenedor
 - Compartidos entre contenedores (en una carpeta interna de docker)

Docker Compose

- **Ejemplo: App web con BBDD**
 - Web con tecnología Python y framework Flask
 - BBDD MongoDB
 - 2 Contenedores

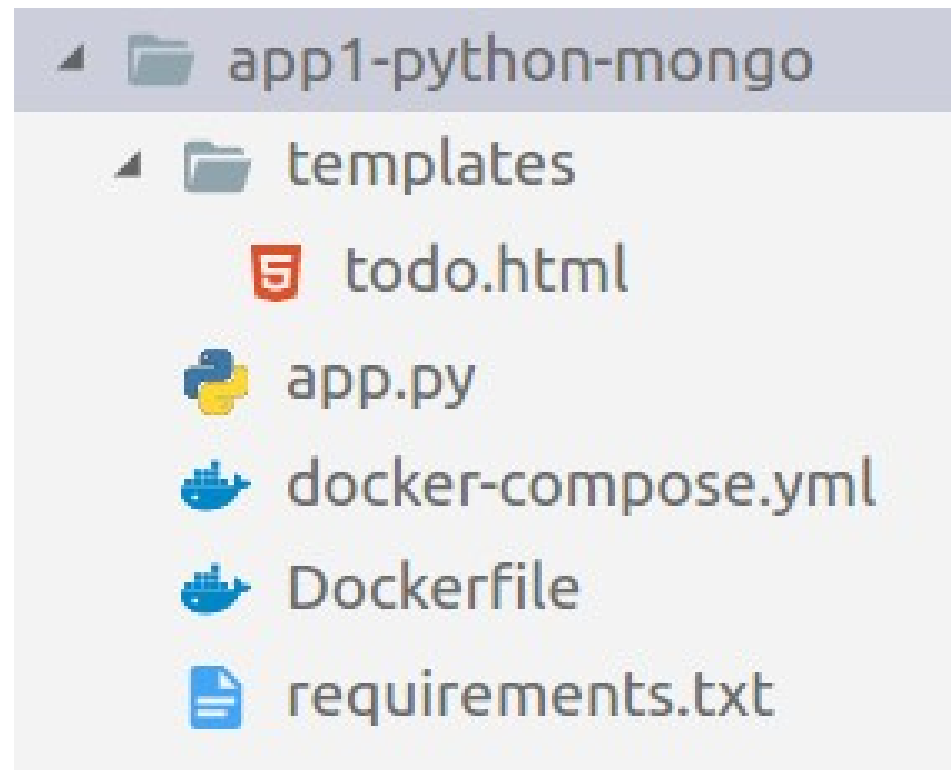
http://localhost:5000



<http://containertutorials.com/docker-compose/flask-mongo-compose.html>

Docker Compose

- Ejemplo: App web con BBDD



<https://github.com/codeurjc/curso-docker/tree/master/app1-python-mongo>

Docker Compose

app.py

- Python web app

```
import os
from flask import Flask, redirect, url_for, request, render_template
from pymongo import MongoClient

app = Flask(__name__)

client = MongoClient('db', 27017)
db = client.tododb

@app.route('/')
def todo():

    _items = db.tododb.find()
    items = [item for item in _items]

    return render_template('todo.html', items=items)

@app.route('/new', methods=['POST'])
def new():

    item_doc = {
        'name': request.form['name'],
        'description': request.form['description']
    }
    db.tododb.insert_one(item_doc)

    return redirect(url_for('todo'))

if __name__ == "__main__":
    app.run(host='0.0.0.0', debug=True)
```

Docker Compose

- HTML Template

templates/todo.html

```
<form action="/new" method="POST">
  <input type="text" name="name"></input>
  <input type="text" name="description"></input>
  <input type="submit"></input>
</form>

{% for item in items %}
  <h1> {{ item.name }} </h1>
  <p> {{ item.description }} </p>
{% endfor %}
```

- App libraries

requirements.txt

```
flask
pymongo
```

Docker Compose

- Dockerfile de la aplicación web

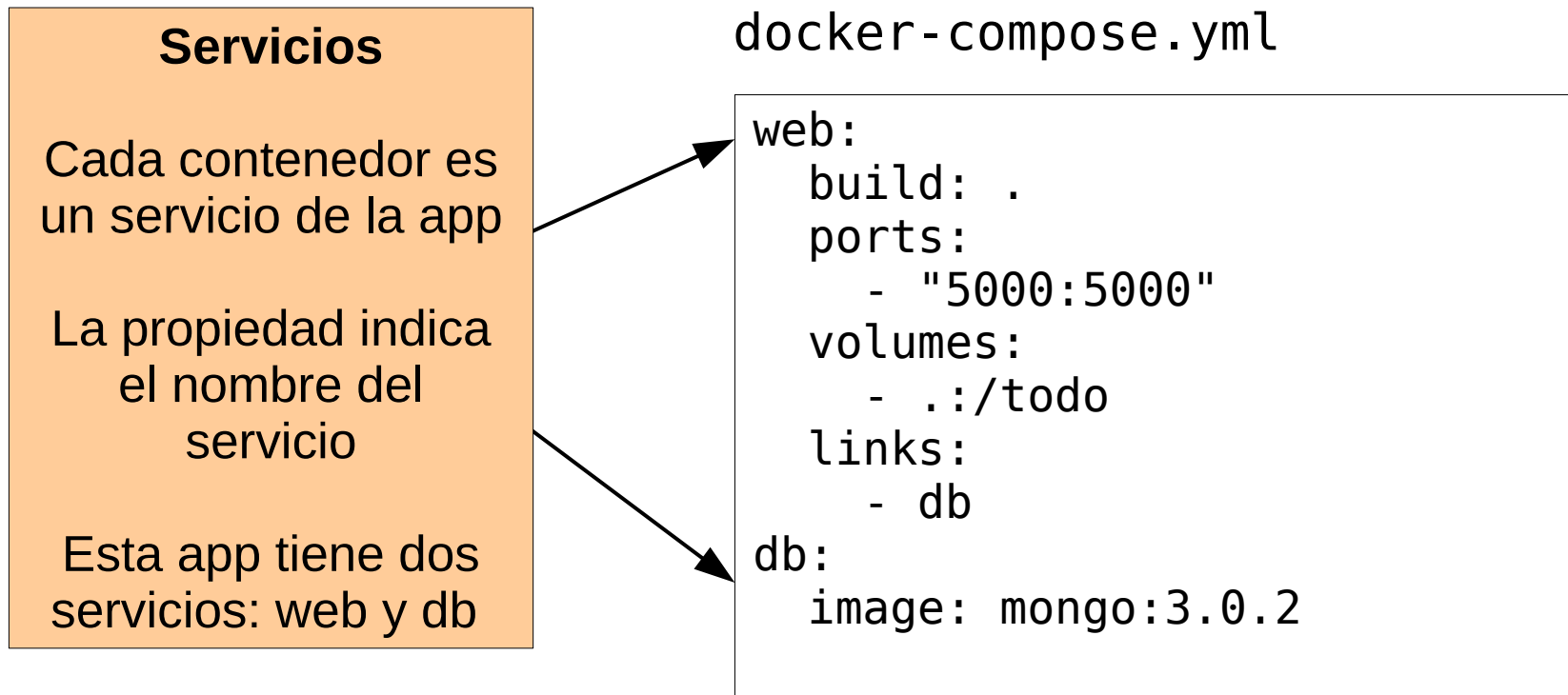
Dockerfile

```
FROM python:2.7
ADD . /todo
WORKDIR /todo
RUN pip install -r requirements.txt
CMD ["python", "-u", "app.py"]
```

- Podemos construir la imagen con este Dockerfile desde línea de comandos si queremos, pero vamos a usar docker-compose para que lo haga

Docker Compose

- Fichero docker-compose.yml



Docker Compose

- Fichero docker-compose.yml

docker-compose.yml

build

Se indica la ruta del Dockerfile para construir la imagen

```
web:
  build: .
  ports:
    - "5000:5000"
  volumes:
    - ./todo
  links:
    - db
db:
  image: mongo:3.0.2
```

Docker Compose

- Fichero docker-compose.yml

image

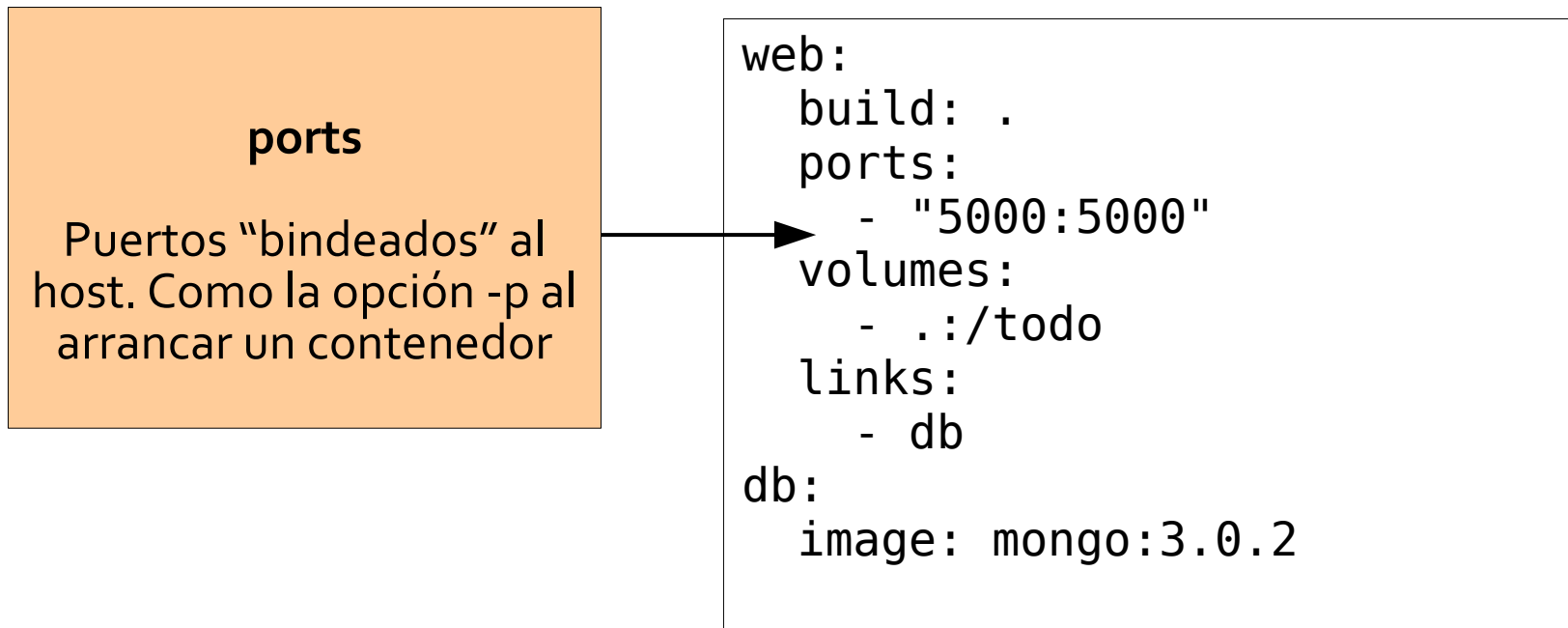
Se indica el nombre de la imagen en Dockerhub o en local

docker-compose.yml

```
web:
  build: .
  ports:
    - "5000:5000"
  volumes:
    - ./:/todo
  links:
    - db
db:
  image: mongo:3.0.2
```

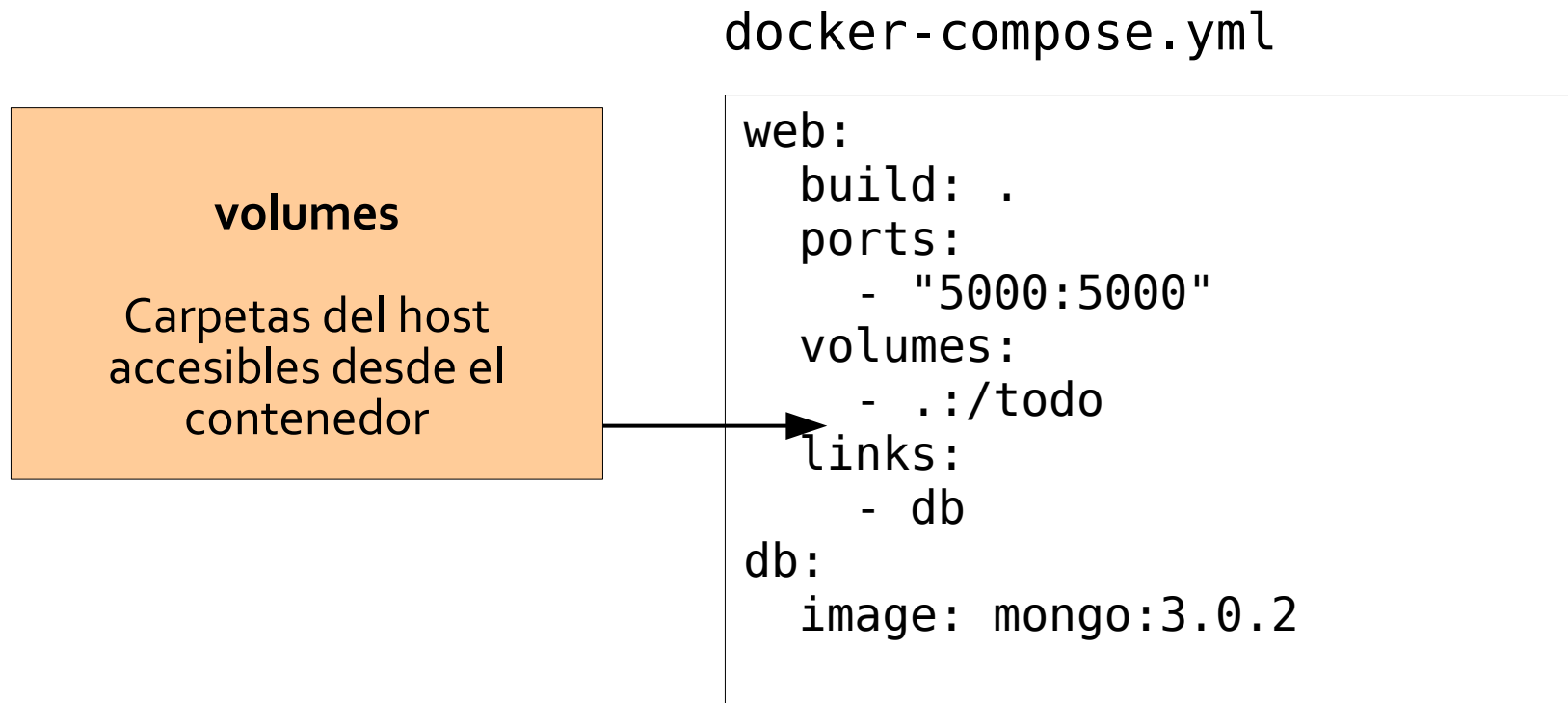
Docker Compose

- Fichero docker-compose.yml



Docker Compose

- Fichero docker-compose.yml



Docker Compose

- Fichero docker-compose.yml

links

Permite acceder al servicio 'db' desde el servicio 'web' usando como nombre del host 'db'

docker-compose.yml

```
web:
  build: .
  ports:
    - "5000:5000"
  volumes:
    - .:/todo
  links:
    - db
db:
  image: mongo:3.0.2
```

Docker Compose

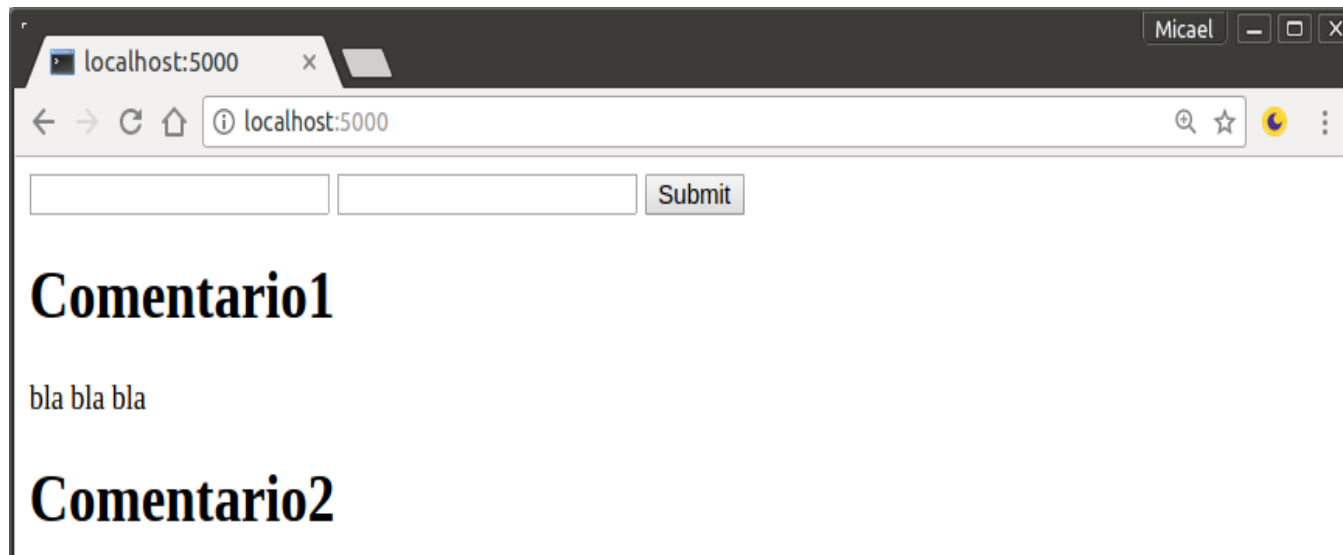
- Arrancar la aplicación

```
$ docker-compose up
```

- Construye la imagen si no está construida ya.
 - Si la imagen está disponible, no se reconstruye aunque cambie el código o el Dockerfile, es necesario ejecutar **docker-compose up --build**
- Se descarga la imagen de MongoDB si no está en local
- Inicia los dos contenedores

Docker Compose

- Arrancar la aplicación
 - La app está disponible en <http://localhost:5000/>



Nota: Con Docker Toolbox la web está disponible en la IP de la VM (obtenida con el comando **docker-machine ip default**)

Docker Compose

- ¿Cómo funciona?
 - Se muestran los logs de todos los contenedores
 - Para parar la app, **Ctrl+C** en la consola (también **docker-compose stop**)
 - Si se para y arranca de nuevo el servicio sin cambiar la configuración de un contenedor, **se vuelve a iniciar el mismo** (no se crea uno nuevo). Los datos de la BBDD no se pierden porque están dentro del contenedor

Docker Compose

- **Ideal para desarrollo**

- Podemos definir una app con múltiples contenedores en un fichero de texto (y subirlo a un repositorio)
- Cualquier desarrollador puede arrancar la app sin tener nada instalado en local (sólo docker y docker-compose)
- Es muy cómodo iniciar y parar todos los servicios a la vez y sólo cuando realmente se necesitan (no tienen que estar iniciados al arrancar la app)
- Todos los logs centralizados

Docker Compose

- **Distribución de apps dockerizadas**
 - Si todos los contenedores del compose están en DockerHub, para distribuir una app multicontenedor dockerizada basta con descargar el docker-compose.yml y arrancarlo.
 - Con curl disponible y el docker-compose.yml en github:

```
curl https://raw.githubusercontent.com/phundament/app-tutum/docker-compose.yml \
| docker-compose -f - up -d
```

Ejercicio 7

- **Dockeriza la aplicación java-webapp-bbdd**
 - Enunciado `curso-docker/java-webapp-bbdd-enunciado`
 - Se usará docker-compose
 - La aplicación necesita una BBDD MySQL
 - Password de root: `pass`
 - En una aplicación SpringBoot se puede configurar la ruta de la BBDD y el esquema con la variable de entorno:
 - `SPRING_DATASOURCE_URL=jdbc:mysql://<host>/<database>`
 - Desactivar los tests al construir la app Maven: `-DskipTests=true`