

Docker, una guía práctica

Backend



Docker, una guía práctica

Cuántas veces hemos dicho o escuchado decir a alguien: “¡Pues en mi ordenador funciona!”. ¿Decenas? ¿Centenas? Pues bien, el objetivo de este fastbook es eliminar esta frase de nuestro día a día, ¿y cómo lo haremos...? Con Docker.

Aprenderemos qué es Docker y a manejarnos con el sistema desde lo más básico hasta la puesta en producción de proyectos complejos con varios contenedores.

Ah, y un aviso a los futuros dockerers: no esperes una guía teórica, para eso ya dispones de la documentación oficial de Docker. A lo largo de estas páginas abordaremos el estudio de esta tecnología desde un punto de vista práctico. ¡Vamos allá!

Autor: Antonio Blanco Oliva

☰ Qué es Docker

☰ Imágenes

☰ Contenedores

☰ Dockerfile

☰ Docker Compose

☰ ¿Y cómo desplegamos la aplicación?

☰ Conclusiones

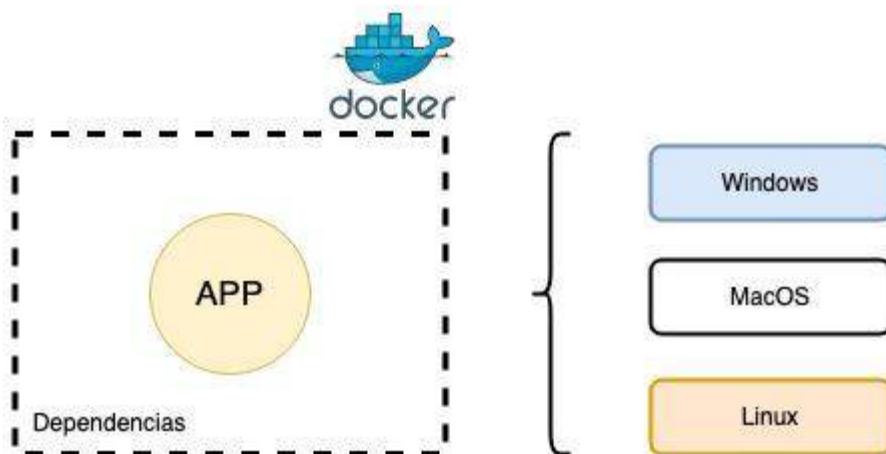
Qué es Docker



Qualentum Lab

Docker es una **tecnología de código abierto** que nos permite el manejo de imágenes y contenedores de formas sencilla y automatizada. Gracias a ella, podremos crear, desplegar y ejecutar nuestras aplicaciones en entornos controlados mediante el uso de contenedores.

Los contenedores son entornos de ejecución contenidos en sí mismos, que tienen todo lo necesario para que funcionen independientemente del entorno en el que se encuentren.



Esto nos ofrece **un potente control sobre errores** en nuestra aplicación, ya que el funcionamiento de la app no dependerá de si está en un sistema operativo u otro, o si nos faltan unas librerías o módulos, ya que correrá sobre un entorno controlado cuyas dependencias ya habremos solucionado y que podremos correr sobre cualquier otro sistema.

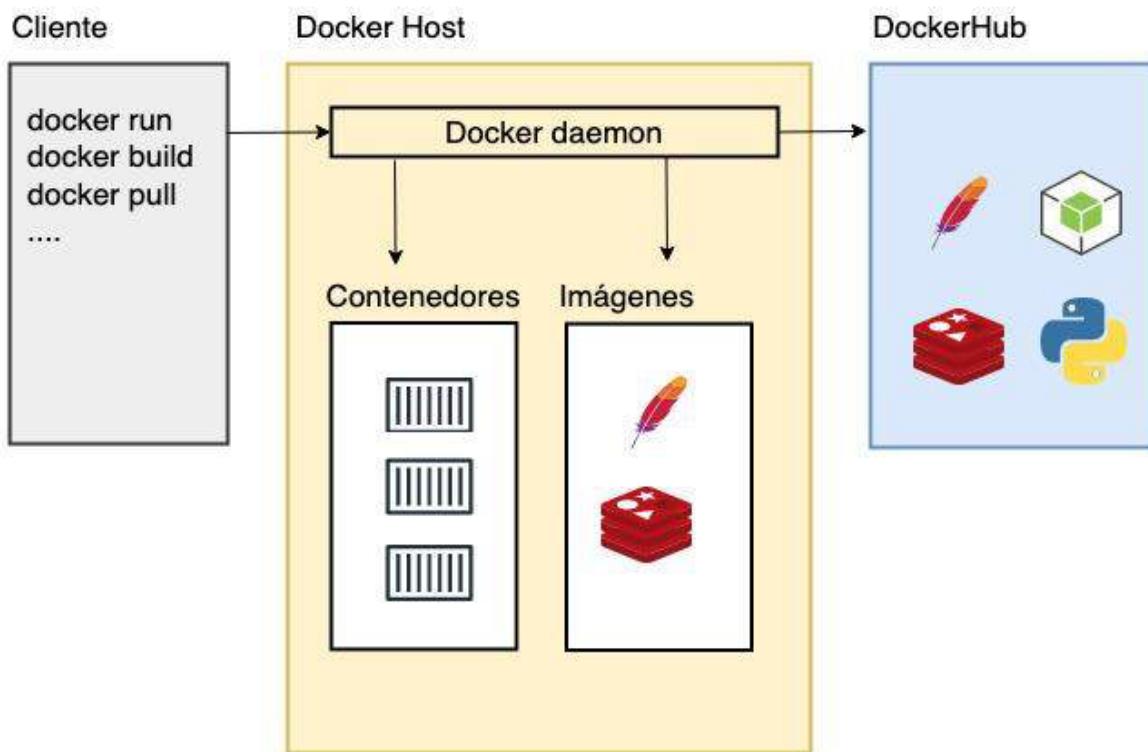
Arquitectura de Docker

Docker se puede dividir en **tres elementos a nivel de arquitectura**: el cliente, el host y el registro. Vamos a definir cada uno de ellos:

EL CLIENTE	EL HOST O SERVIDOR	UN REGISTRO O DOCKER HUB
Corresponde a la instalación que haremos en nuestro equipo y que permite llamar al sistema host de Docker. Nos ofrece la posibilidad de ejecutar instrucciones Docker.		

EL CLIENTE	EL HOST O SERVIDOR	UN REGISTRO O DOCKER HUB
Es la parte encargada del manejo de los contenedores y las imágenes, así como de comunicarse con la parte del registro de imágenes (Docker Hub).		

EL CLIENTE	EL HOST O SERVIDOR	UN REGISTRO O DOCKER HUB
Hablamos de un repositorio de imágenes donde encontraremos no solo imágenes oficiales de multitud de entornos, sino que podremos tener nuestras propias imágenes y distribuirlas desde allí, se puede decir que es el Github de imágenes Dockers.		



Pautas para preparar el entorno

Para empezar a trabajar con Docker, lo primero que debemos hacer es **instalarlo** en nuestro equipo. Podemos **encontrar el programa para descargar** y cómo llevar a cabo dicha instalación en la documentación de Docker.

 Puedes consultarla [aquí](#) la documentación de Docker.

Docker Desktop for Mac

A native application using the macOS sandbox security model which delivers all Docker tools to your Mac.

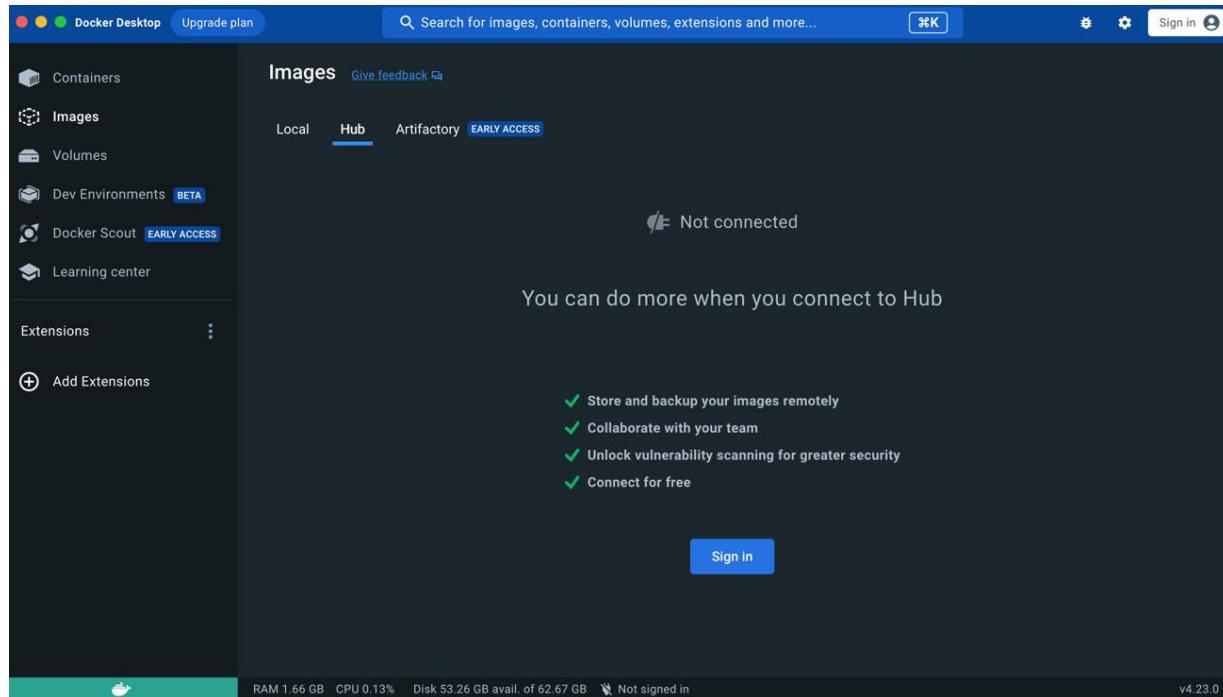
Docker Desktop for Windows

A native Windows application which delivers all Docker tools to your Windows computer.

Docker Desktop for Linux

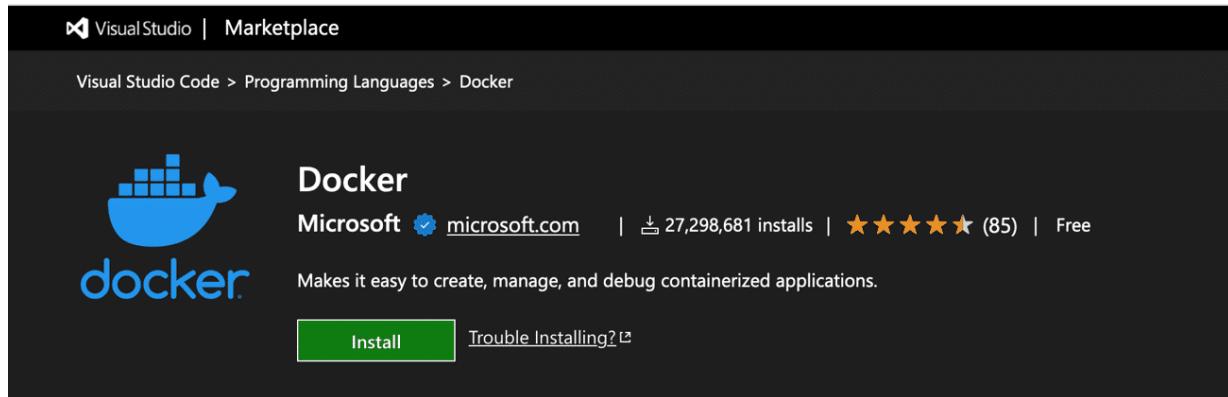
A native Linux application which delivers all Docker tools to your Linux computer.

Una vez instalado y tras ejecutarlo, nos pedirá **loguearnos**, por lo que tendremos que **crear una cuenta en Docker Hub** si no la tenemos aún.



Creada la cuenta o logueado, podremos **acceder a la aplicación y ver nuestro perfil** entre otros elementos.

Seguramente, tienes instalado [Microsoft Visual Studio](#) como IDE de programación, por lo que te recomendamos tener instalada la extensión Docker para él ([consulta este enlace](#)).



Imágenes



Una imagen Docker es un paquete que contiene una aplicación, sus dependencias, un sistema operativo y una serie de configuraciones.

Estas imágenes serán los elementos que se usarán para **montar los servicios o contenedores**. En el trabajo con Docker, todo empieza con las imágenes, pero hay que encontrar la que mejor se adapta a nuestras necesidades. Por ende, adaptarla a nuestros requerimientos será el primer paso para desplegar nuestras aplicaciones.

Estudiaremos algunas de las instrucciones más comunes que solemos usar en el día a día para **el manejo de imágenes**. Usaremos ejemplos reales para que veas casos de usos prácticos. ¡Adelante!

```
docker images
```

Nos permite **listar las imágenes** que tenemos disponibles en nuestro equipo.

```
docker pull python:3.9.18
```

Descarga desde Docker Hub la imagen 'python', cuya versión o tag es **3.9.18**.

Si no se indica un **TAG**, se descargará la última versión (*latest*), lo cual no es recomendable, ya que, dependiendo de cuándo ejecutemos el comando, la última versión será una u otra, por eso, insisto, **se recomienda siempre indicar un TAG**.

```
docker run python:3.9.18
```

Actúa como el comando *pull*, pero además de descargar la imagen, la monta creando un contenedor con ella.

Esta sería **una de las instrucciones que más usemos** y con la que nos pelearemos en el día a día. Veamos ahora una versión más avanzada de dicha instrucción para poder analizarla:

```
docker run -d -v ./bbdd:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=password  
-p 3306 mysql:8.1.0
```

-v

-e

-p

Con este parámetro montaremos un volumen, es decir, una carpeta de sincronización de datos entre el sistema host y el contenedor.

-v

-e

-p

Para definir variables de entorno dentro del contenedor (muy útil para las configuraciones de aplicaciones).

-v

-e

-p

Mapea un puerto entre la máquina host y el contenedor.

```
docker rmi 60b356429fb9
```

Elimina una imagen a partir de su ID. ¿Y de dónde sacamos su ID?

Con *docker images* obtendremos no solo el listado de imágenes que tenemos con sus nombres y etiquetas, sino que también obtendremos sus IDs.

Si dicha imagen se está usando en algún contenedor, no nos dejará eliminarla, por lo tanto, tendremos que usar el **parámetro '-f'** para forzar el borrado en este caso.

```
docker save -o mi_imagen.tar mysql:8.1.0
```

Guarda una copia de la imagen *mysql:8.1.0* en un fichero '.tar' llamado 'mi_imagen.tar'.

```
docker load -i mi_imagen.tar
```

Carga una imagen anteriormente guardada en 'mi_imagen.tar' para tenerla **disponible** en nuestro sistema.

```
docker build .
```

Construye una imagen a partir de un Dockerfile (veremos más adelante qué es esto del Dockerfile). Lo más habitual es tenerlo todo en la misma carpeta, por lo que se suele usar como parámetro el valor punto (.).

Contenedores



Los contenedores son instancias de entornos aislados y ligeros que nos permiten ejecutar aplicaciones y sus dependencias, pudiendo montarlos, ejecutarlos y eliminarlos fácilmente.

Los contenedores tienen que contender un sistema operativo, unas aplicaciones y más información que se necesite, es decir, una imagen.

Las **principales características** que nos ofrece trabajar con contenedores son:

Aislamiento

Al montar cada servicio en un contenedor, evitamos conflictos entre ellos y problemas de dependencias.

Portabilidad

Al no tener dependencias el sistema host en el que corren, podemos tener el mismo contenedor, por ejemplo, corriendo en un Mac, Windows o Linux.

Eficiencia

Son más livianos y eficientes que las máquinas virtuales.

Rapidez

Podemos iniciar y detener contenedores rápidamente, facilitando su administración es escalabilidad.

Versionado

Tener la opción de versionado nos permitirá ejecutar nuestra aplicación en distintas versiones y dependencias.

Veamos algunas instrucciones a modo de ejemplos que usaremos en nuestro día a día en el manejo de contenedores.

```
docker ps
```

Muestra el listado de contenedores que tengamos ejecutándose. Si le añadimos **el parámetro -a**, mostrará todos los contenedores independientemente de su estado.

```
docker rm 4615da274618
```

Elimina un contenedor. El contenedor no puede estar en ejecución. Si usamos **el parámetro -f**, forzamos su eliminación incluso si está corriendo.

```
docker start 4615da274618
```

Arranca un contenedor que esté parado.

```
docker stop 4615da274618
```

Para un contenedor que se esté ejecutando.

```
docker build -t mi_imagen:v1 .
```

Construye una imagen a partir de un archivo Dockerfile. El segundo parámetro '.' indica el directorio donde se encuentra el fichero Dockerfile.

Estas son solo algunas de las instrucciones que tiene disponible Docker, que quizá sean las que más usarás en tu día a día. Recuerda: tienes disponible en [la página de documentación](#), información sobre todas ellas y sus parámetros.

Red de contenedores

Ya tenemos nuestros contenedores, ¡genial!, pero aislados, es decir, no tienen forma de comunicarse entre ellos, por lo que para poco nos valen, ¿cierto?

Solucionemos el problema. Veamos a continuación cómo conectarlos y, para ello, usaremos la aplicación. ¡Toma nota!

```
docker network
```

Cuando hemos usado el comando `docker run`, hemos podido exponer puertos de nuestro contenedor, e incluso asignarle una IP.

Pero **esta comunicación es entre el equipo anfitrión y el contenedor**, lo cual se hace a través de una de las redes por defecto que tiene Docker, la red 'host'.

Veamos ahora **las redes que tenemos disponibles** con...

```
docker network ls
```

Nos encontraremos con **3 redes** por defecto:

BRIDGE

HOST

NONE

Viene por defecto y permite comunicar contenedores que están en el mismo host.

BRIDGE**HOST****NONE**

La red del equipo host, por lo que, si la usamos para conectar un contenedor, este se encontrará bajo la misma red que el host.

BRIDGE**HOST****NONE**

Deja completamente aislado al contenedor, tanto de otras redes como del host.

```
docker network create mi_red
```

Nos permitirá **crear una red propia** donde poder conectar los contenedores. Una vez tenemos nuestra red creada, podremos usarla en *docker run* para arrancar contenedores directamente conectados a dicha red.

```
docker network connect mi_red 4615da274618
```

Nos permitirá **añadir un contenedor a una red** si no lo hemos hecho antes con 'run'.

```
docker network connect --alias mi_alias mi_red 4615da274618
```

Especial atención a los parámetros **--alias**, ya que nos permitirán ponerle un alias al contenedor dentro de la red, así no dependeremos de la IP que se le asigne, ya que esta puede variar.

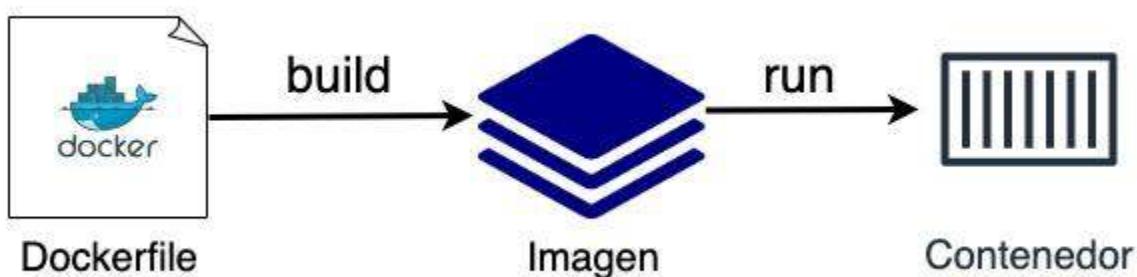
Dockerfile



Una vez que sabemos manejarnos con las imágenes, pasemos a crear nuestras propias imágenes personalizadas, que podrán **no solo tener un entorno preparado** con nuestras dependencias, sino también nuestro propio código ya desplegado.

El fichero Dockerfile se utiliza para definir las instrucciones y configuraciones necesarias para construir una imagen de Docker.

Esto nos permitirá **replicar la construcción de una misma imagen** por distintas personas y distintos entornos, simplemente ejecutando este fichero.



Veamos algunas de las **instrucciones** más utilizadas en un Dockerfile. ¡Apunta!

```
FROM ubuntu:20.04
```

Esta es la primera instrucción en un Dockerfile y **especifica la imagen base** desde la cual se construirá la nueva imagen. En este caso, será a partir de una imagen Ubuntu versión 20.04.

```
WORKDIR /app
```

Define el **directorio de trabajo** dentro del contenedor para las instrucciones siguientes.

```
COPY . /app
```

Se utiliza para **copiar archivos o directorios locales** en el sistema de archivos del contenedor.

```
RUN apt-get update && apt-get install -y package-name
```

Ejecuta comandos en el contenedor durante el proceso de construcción de la imagen.
Puede utilizarse para instalar paquetes, configurar el entorno, etc.

```
EXPOSE 80
```

Indica **qué puertos deben estar disponibles** para escuchar dentro del contenedor cuando se ejecute.

```
CMD ["python", "app.py"]
```

Definen **el comando o la entrada principal** que se ejecutará cuando se inicie un contenedor basado en la imagen.

```
ENV MI_VAR_DE_ENTORNO=valor
```

Establece **variables de entorno** en el contenedor.

```
USER mi_usuario
```

- Especifica el usuario que ejecutará los comandos en el contenedor.
- Todas las instrucciones que se ejecuten detrás serán bajo dicho usuario.

```
VOLUME /data
```

Crea un **punto de montaje para volúmenes externos**.

Recuerda que estas son algunas de las instrucciones más comunes que usaremos, pero puedes consultar la documentación oficial para ver el resto o parámetros adicionales de estas.

Docker Compose



Una de las herramientas que nos ofrece Docker es Docker Compose que nos permitirá la **gestión de múltiples contenedores**; por lo tanto, nos da la posibilidad de definir y administrar entornos de trabajo más sofisticados.

Nuestra aventura con Docker Compose empieza con la creación de un fichero YML donde definiremos nuestras directivas. El fichero será *docker-compose.yml*.

Veamos un ejemplo de fichero que nos permitiría **lanzar 2 contenedores** y sobre él analizaremos las **directivas** usadas:

```
version: "3.9"

services:
  # Primer servicio la web
  web:
    build: .
    ports:
      - 5000:5000
    volumes:
      - .:/code
    environment:
      FLASK_ENV: development

  # Segundo servicio redis
  redis:
    image: "redis:alpine"
```

1

version: define la versión de Docker Compose que se usará.

2

services: define los servicios (contenedores) que formarán parte de tu aplicación. Cada servicio tiene su propia configuración.

3

build: utilizado para construir una imagen personalizada a partir de un Dockerfile en la ubicación especificada.

4

ports: indica los puertos que se expondrán en el contenedor y su mapeo con la máquina host.

5

volumes: monta volúmenes en el contenedor y los enlaza con una ruta en el host.

6

environment: define variables de entorno para el contenedor.

7

networks: define redes personalizadas para conectar los servicios, por ejemplo:

```
services:  
  app:  
    networks:  
      - mi-red  
  db:  
    networks:  
      - mi-red  
networks:  
  mi-red:
```

Como ya se ha indicado, este es solo un ejemplo de las instrucciones más comunes con las que trabajaremos. Puedes ampliar la información sobre dichas instrucciones y otras más que existen en la página oficial de documentación de Docker.

¿Y cómo desplegamos la aplicación?



Aunque queda fuera del contenido de esta asignatura, es interesante comentar que los principales *players* del mercado cloud ofrecen soluciones para desplegar nuestros contenedores en sus entornos. Hagámosles una visita rápida.

AWS

Amazon nos ofrece [Elastic Container Service](#), ECS, como servicio para desplegar contenedores en su nube. Aunque si eres nuevo en el tema, quizás el servicio [Amazon Fargate](#) es el ideal, ya que ofrece una capa de abstracción adicional, para que nos olvidemos de la infraestructura y nos centremos en las aplicaciones.

Google

Google nos ofrece desde su suite Cloud Build, la opción de desplegar imágenes Docker.

 Puedes ver el proceso [aquí](#).

Azure

También los de Microsoft tienen un servicio específico para [Docker](#). Además, han añadido una opción para poder usar nuestros contenedores como fuente cuando vamos a desplegar [Power Apps](#).

Conclusiones



Qualentum Lab

Tal y como prometimos, en este fastbook has encontrado todas las pautas necesarias para trabajar con Docker, en su gran mayoría apoyadas en ejemplos prácticos.

No obstante, y aunque pequemos de insistentes, lo ideal para mejorar tus competencias en esta tecnología es que completes el estudio del fastbook revisando la documentación oficial de Docker en este [enlace](#).

También te compartimos los **temas en los que deberías incidir** a la hora de estudiar:

1

Los comandos tanto de imágenes como contenedores.

2

Dockerfile no tendrá ningún misterio si revisas su documentación oficial.

 Para saber más sobre comandos, clica en este [enlace](#) y para ampliar información sobre Dockerfile, entra [aquí](#).

¡Enhorabuena! Fastbook superado



Qualentum.com