TUTORIAL CONTENEDORES DOCKER

Contenido

[Introducción 2](#_Toc82507843)

[Estructura de los contenedores Docker 2](#_Toc82507844)

[Creación de un contenedor Docker 3](#_Toc82507845)

[Despliegue de contenedores Docker 6](#_Toc82507846)

[Ejemplo práctico 7](#_Toc82507847)

[Creación de la aplicación práctica 7](#_Toc82507848)

[Despliegue de la aplicación práctica 9](#_Toc82507849)

[Puesta en marcha 10](#_Toc82507850)

# Introducción

Primero, tendremos que saber qué es [Docker](https://docs.docker.com/get-started/overview/). Docker nos permite crear **contenedores ligeros** y **portables** para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado [[link](https://www.javiergarzas.com/2015/07/que-es-docker-sencillo.html)]. Es decir, Docker nos da soporte para crear entornos ligeros, los cuales van a tener solo lo necesario para que la aplicación pueda ejecutarse. Sin Docker, necesitaríamos instalar en nuestro ordenador cada requisito que necesiten cada aplicación que se ejecute, o crear máquinas virtuales e instalar lo necesario ahí.

# Estructura de los contenedores Docker

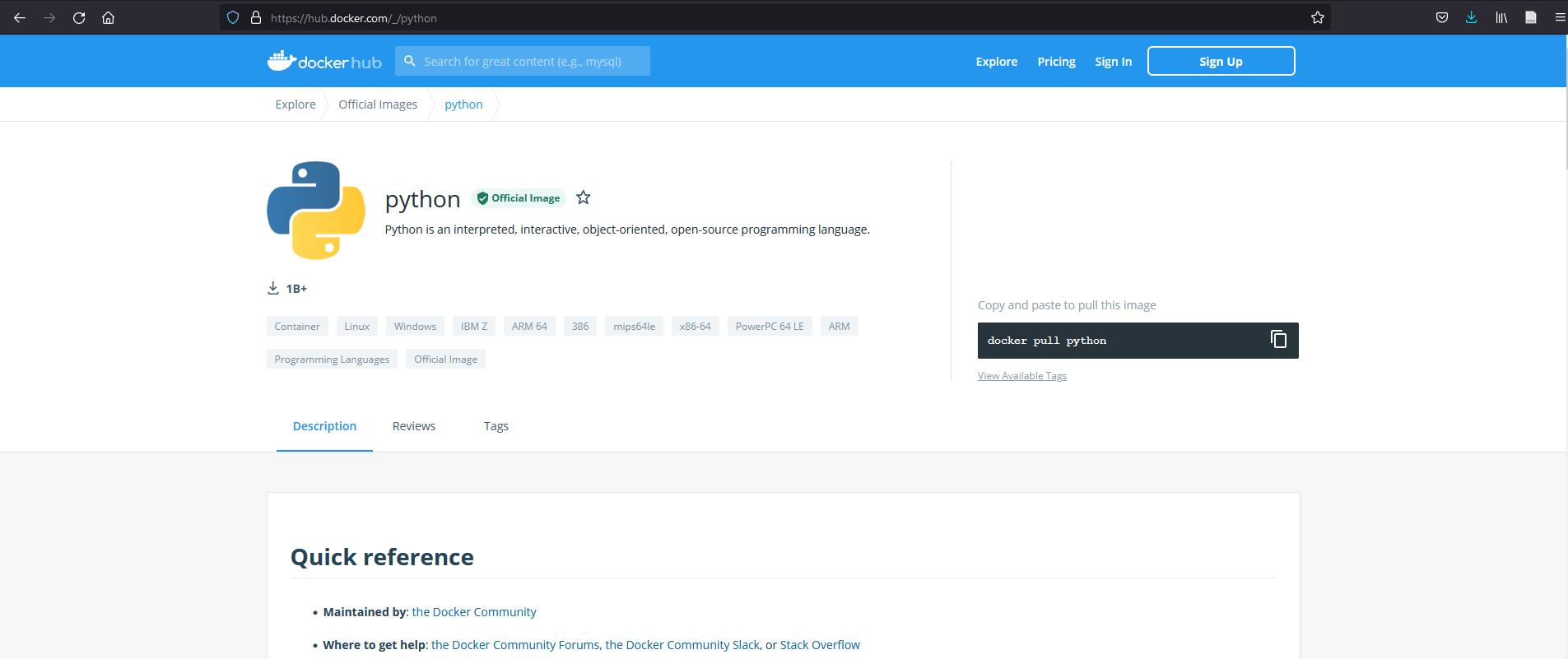
En un contenedor Docker aparecen varios conceptos. Antes de verlos, hay que repasar lo que hemos visto. Como hemos dicho, Docker sirve para crear entornos ligeros para la ejecución de aplicaciones, servidores, etc. Al fin y al cabo, los contenedores se tratan de **máquinas virtuales ligeras** con lo necesario para cumplir su función.

Los contenedores Docker siempre van a tener una **base**. Como pasa con los ordenadores o las máquinas virtuales, necesitan como base un sistema operativo. En Docker pasa lo mismo, vamos a poder utilizar una base ya desarrollada (p.e. una base del sistema operativo de Ubuntu Linux) o desarrollar una nuestra y reutilizarla. En Docker a estas bases se les llama **imágenes** y hay muchísimos tipos de ellas. Como hemos dicho, se pueden desarrollar y reutilizar, por lo que hay una gran cantidad de imágenes. Una imagen puede tener todo lo necesario para ejecutar una aplicación (el código fuente, el comando para arrancarlo, etc.), o ser solo una base para crear más imágenes (Ubuntu Linux). Los contenedores Docker se crearán desde una imagen Docker. Como se puede apreciar en la imagen, se pueden crear varios contenedores desde una misma imagen.



A la hora de crear un contenedor Docker, lo primero que se debe hacer es analizar la aplicación que queremos desarrollar, para saber qué imagen base hay que escoger. Por ejemplo, si queremos desarrollar una aplicación en lenguaje Python podremos buscar una imagen base que tenga lo necesario, o podremos utilizar una base como Ubuntu, e instalarle Python.

Docker tiene un [repositorio de imágenes](https://hub.docker.com/) bastante grande, donde podremos encontrar casi cualquier recurso. La imagen base oficial de Python es esta:



# Creación de un contenedor Docker

Hay varias formas de crear contenedores Docker. La más sencilla y más utilizada es mediante el archivo conocido como [*Dockerfile*](https://openwebinars.net/blog/que-es-dockerfile/). En este archivo añadiremos los comandos que se ejecutarán en la creación de la imagen Docker.

Primero, crearemos el archivo [Dockerfile](https://docs.docker.com/engine/reference/builder/). No necesita extensión, ya que es un archivo de texto plano. Después, definiremos la imagen base comentada anteriormente. Para ello se utiliza el comando **FROM**. Siguiente el ejemplo anterior, si queremos utilizar la imagen base de Python o la de Ubuntu, sería de la siguiente manera.

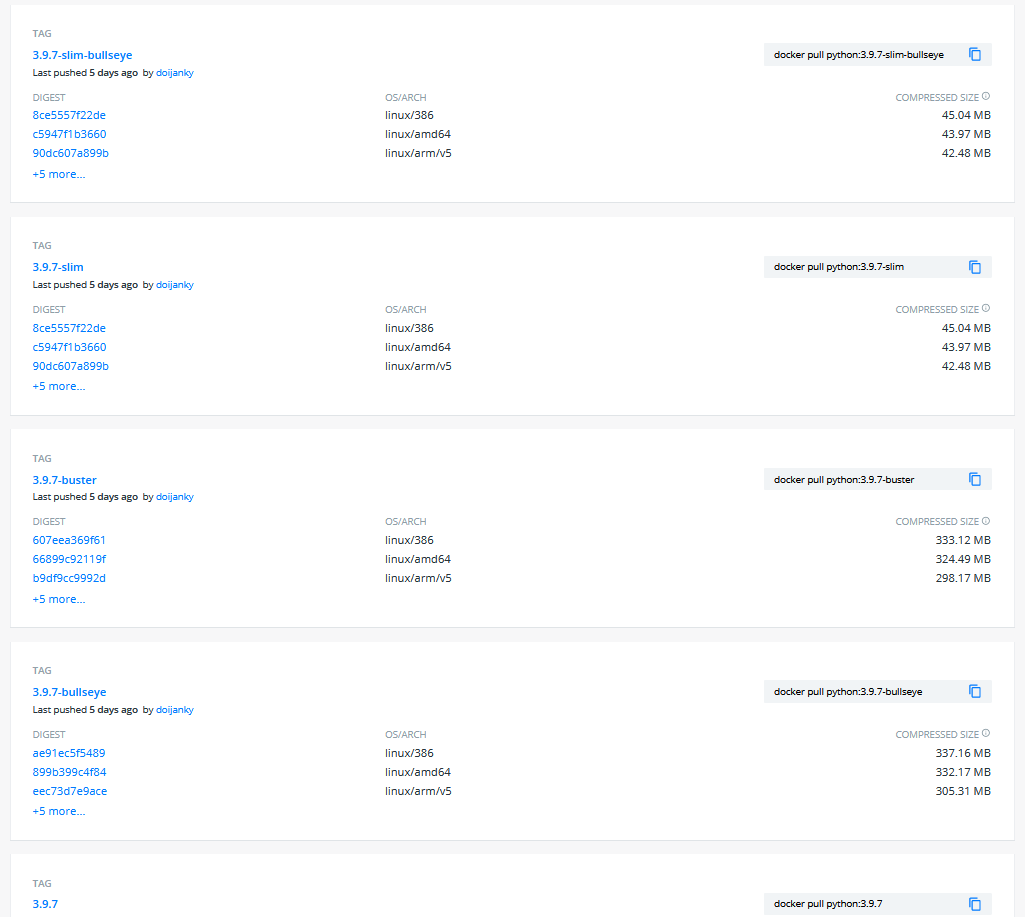
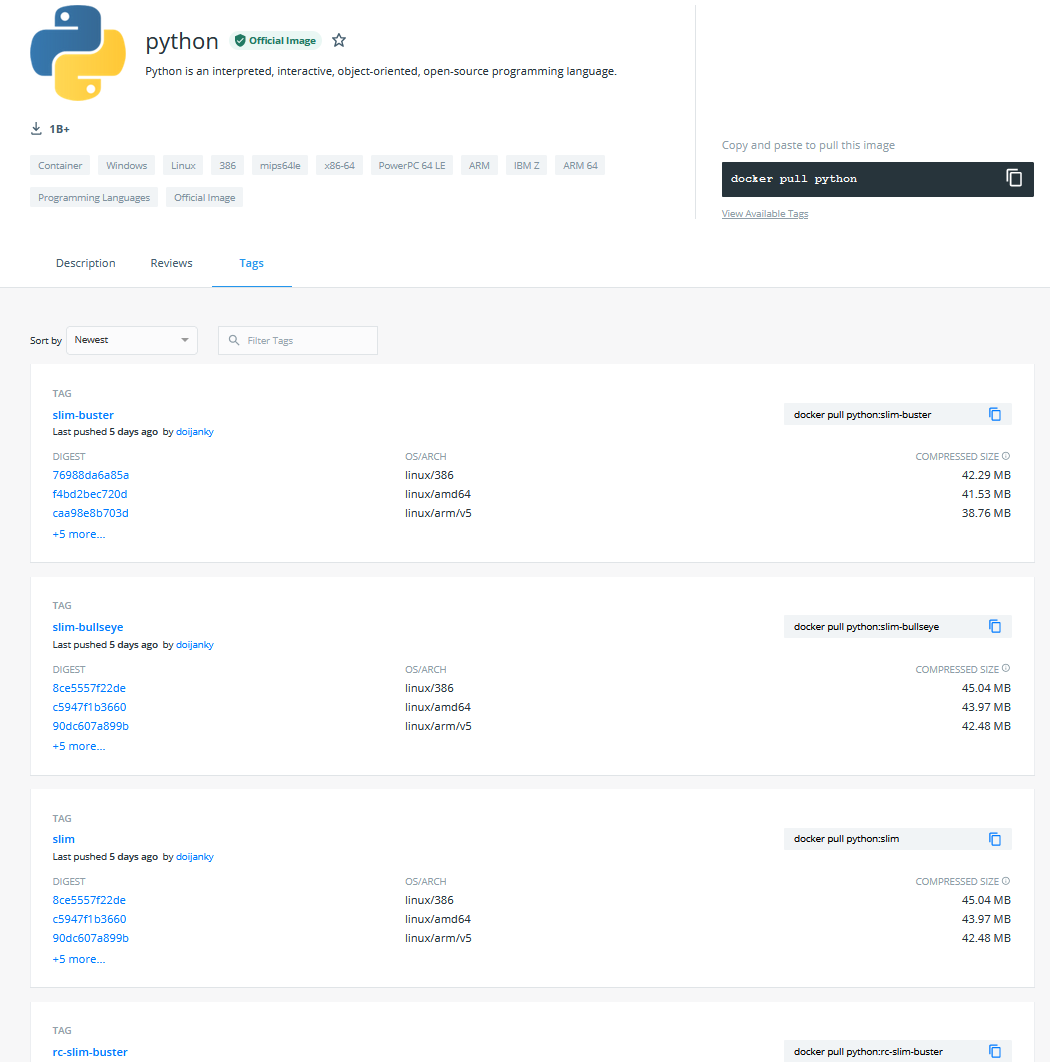
FROM ubuntu:latest

FROM python:3

Como se puede apreciar, las imágenes están separadas por un “:”. Esto se debe a que se organizan por nombre y etiqueta.

*<nombre>:<etiqueta>*

Se puede ver de varias formas. El nombre puede ser el nombre de la imagen misma y la etiqueta la versión de esa imagen, como está organizado, por ejemplo, el ejemplo anterior de Python:



También se puede organizar por repositorios, y elementos dentro de cada uno. Por ejemplo, más adelante en otro tutorial se nombrará el Google Container Registry, un repositorio de imágenes Docker. Ahí, se podrán agrupar los elementos utilizando esta nomenclatura, p.e.:

*<grupo>:<elemento>*

Una vez entendido el concepto de imagen, vamos a la creación de una nueva. Ya hemos comentado que el primer paso es seleccionar la base para el contenedor. Lo siguiente es ver si esa base necesita algo más. Es posible que la imagen base tenga lo necesario para ejecutar aplicaciones, pero no librerías adicionales que necesita nuestra aplicación.

Para hacer cambios en la imagen base (instalar nuevo software, cambiar recursos, etc.) Dockerfile nos habilita la opción **RUN**. Con esta opción, se ejecutará el comando que pongamos a continuación. Si ponemos *RUN apt-get install python*, se instalará Python en la imagen base. Hay que tener en cuenta que no se modifica la imagen base. Los cambios que hagamos no se hacen en la base, si no que Docker la descarga, crea una nueva, y ahí es donde se instalaría Python, en la nueva imagen que se cree.

Después, como ya hemos comentado, un contenedor Docker es una unidad “aislada”, por lo que deberá tener todo lo necesario. Eso incluye el código fuente de nuestra aplicación (el archivo .jar, .jar, .py, etc.). Para que el contenedor pueda ejecutar ese archivo, debemos dárselo. Para ello se utilizan los comandos **COPY** o **ADD**. Estos nos permiten copiar archivos desde nuestro ordenador a dentro del contenedor.

*COPY/ADD <path al archivo con código fuente> <path dentro del contenedor>*

Por último, tenemos el comando **CMD**. En este se especifica el comando que se ejecutará cuando se arranque el contenedor. Aquí es donde añadiremos la orden de arranque de la aplicación, el despliegue de un servidor… Tiene dos formas de expresarse: como el comando RUN, es decir, poner el comando separados por espacios, o en una lista en la que cada elemento es un argumento del comando.

*CMD arranca la aplicación*

*CMD [“arranca”, “la”, “aplicación”]*

# Despliegue de contenedores Docker

Una vez tengamos el Dockerfile definido, podremos crear nuestra imagen (para después que se cree el contenedor a partir de ella). Para ello, utilizaremos el comando [*docker build*](https://docs.docker.com/engine/reference/commandline/build/). Si el archivo se llama Dockerfile podremos utilizar la primera opción, ya que Docker buscará un archivo que se llame de esa forma. Si no, nos bastará con utilizar la segunda opción y especificarle donde está el archivo Dockerfile.

> docker build -t <nombre imagen>:<etiqueta imagen> <path a los archivos> (1)

> docker build -t <nombre imagen>:<etiqueta imagen> -f <path al Dockerfile> <path a los archivos> (2)

Para ver la imagen creada, y todas las demás en nuestro dispositivo, basta con ejecutar este comando:

> docker images

Si queremos borrar la imagen creada:

> docker rmi <nombre imagen>:<etiqueta>

Para crear el contenedor, es importante tener en cuenta donde se va a desplegar. Si lo vamos a hacer en Docker, es muy sencillo. Con el comando[*docker run*](https://docs.docker.com/engine/reference/commandline/run/) Docker utiliza la imagen que le especifiquemos para crear el contenedor y arrancarlo. Conviene que le pongamos un nombre, por si necesitamos encontrarlo después, ya que si no Docker le pone uno aleatorio.

> docker run --name <nombre del contenedor> <nombre imagen>:<etiqueta>

Ahora, podremos ver los contenedores desplegados (1) o eliminar el que deseemos (2):

> docker container ls

> docker rm <nombre contenedor>

# Ejemplo práctico

## Creación de la aplicación práctica

Vamos a crear una aplicación que consta de dos contenedores. El primero va a crear 10 números aleatorios cada 5 segundos y se los va a ir enviando al segundo, el cual va a ir recibiéndolos y sumándolos para conseguir el resultado final. Con este ejemplo veremos en práctica lo comentado anteriormente: como crear y desplegar un contenedor. Además, analizaremos como comunicar dos contenedores.

Comenzaremos con la creación del primer elemento, el encargado de crear los números, el cual le llamaremos ***appCrearNumero***. Para crearlo, el primero paso será desarrollar el archivo Dockerfile. Ya tenemos el código fuente, el cual podemos ver que está desarrollado en Python. Por lo tanto, utilizaremos una imagen base que tenga Python. Para que se entiendan las etiquetas de las imágenes base, utilizaremos diferentes bases para los dos elementos. La aplicación appCrearNumero utiliza dos librerías adicionales: una para enviar mensajes por HTTP y otra para crear objetos JSON. Como hemos visto los contenedores nos brindan un nivel de personalización muy alto, por lo que hay diversas formas de comunicarlos entre ellos. Nosotros el que más utilizamos es HTTP, y es por eso que necesita esas librerías.

Para instalar estas librerías las añadiremos en un archivo de texto y las instalaremos desde ahí. En Python se puede instalar librerías de forma muy sencilla con la herramienta [*pip*](https://ubunlog.com/pip-instalacion-conceptos-basicos-ubuntu-20-04/). Con la opción de RUN añadiremos el comando para la instalación. A continuación, copiaremos el archivo con el código fuente y lo dejaremos en la carpeta inicial. Al estar basada en Linux, la carpeta inicial de la imagen base de Python es “/”. Si analizáis el código fuente de appCrearNumero, podéis comprobar que envía un mensaje a un destino especificado en las variables de entorno. Esta es una forma de personalizar los contenedores.

Una vez creada la imagen, no podremos acceder al código fuente, sin tener que crear otra nueva. Para cambiar el código, se pueden utilizar las variables de entorno. Los valores de estas se pueden modificar en cualquier etapa (creación de la imagen y creación del contenedor). En este caso las modificaremos en el Dockerfile. Deberemos asegurarnos de que el nombre que añadamos en la variable *HOST* y el nombre que le demos al contenedor de appSuma son los mismos. Por último, añadiremos el comando de arranque de la aplicación, en este caso, el que ejecutará el archivo con el código fuente. Al final, este será el Dockerfile de appCrearNumero.

FROM python:alpine3.7

COPY requirements1.txt ./

RUN pip install --no-cache-dir -r requirements1.txt

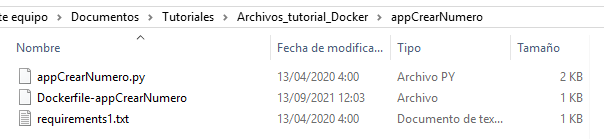
COPY appCrearNumero.py /

ENV APPSUMA\_HOST=app-suma

ENV APPSUMA\_PORT=6000

CMD ["python", "/appCrearNumero.py"]

Ahora vamos a crear su imagen a partir de este Dockerfile. Añadiremos todos los archivos necesarios a una carpeta, la cual nos quedará así:



Ejecutaremos el comando para crear la imagen en el mismo directorio:

> docker build -t aplicacion1:app-crear-numero -f Dockerfile-appCrearNumero .

*Nota: El último punto es importante, quiere decir que los archivos están en el mismo directorio donde se ejecuta el comando.*

A continuación, crearemos el segundo elemento, al que llamaremos ***appSuma***. Seguiremos el mismo procedimiento, empezando por el Dockerfile. Utilizaremos una imagen base diferente para ver como hay muchas opciones, pero a nosotros nos ofrecerá lo mismo que la de appCrearNumero. Para instalar la librería adicional lo haremos igual que antes, aunque ahora se instalará [Flask](https://flask.palletsprojects.com/en/2.0.x/). Esta es una herramienta muy útil para programar servidores en Python. También copiaremos el archivo con el código fuente.

Cuando se arrancar un servidor de Flask, busca el código fuente y el puerto donde se va a ejecutar en las variables de entorno. Para modificarlas y añadir nuestra aplicación, utilizaremos la opción ENV, la cual nos habilita añadir o modificar las variables de entorno de dentro del contenedor. En el caso de flask, como se puede ver en el Dockerfile, se modifican las variables *FLASK\_APP* y *FLASK\_RUN\_PORT*. Por último, añadiremos el comando de arranque del servidor Flask.

FROM python:3.7.0-alpine3.8

COPY requirements2.txt ./

RUN pip install --no-cache-dir -r requirements2.txt

COPY appSuma.py .

ENV FLASK\_APP=appSuma.py

ENV FLASK\_RUN\_PORT=6000

CMD flask run --host=0.0.0.0

Con los archivos en el directorio, podremos ejecutar el comando para crear la imagen:

> docker build -t aplicacion1:app-suma -f Dockerfile-appSuma .

## Despliegue de la aplicación práctica

Para desplegarlo tendremos que tener en cuenta donde se va a hacer. Si es solo en Docker, habrá que analizar cómo comunicar las aplicaciones. Utilizaremos HTTP, pero hay que especificar el destino. Se puede añadir una dirección IP fija y funcionaria, pero eso no es muy funcional. En Docker se puede crear una red donde añadir los contenedores y así utilizar el nombre de ellos, convirtiendo la comunicación mucho más flexible.

*Nota: En el tutorial de Kubernetes se puede comprobar que la comunicación va ir de forma muy parecida.*

Las [redes en Docker](https://dockertips.com/algo_sobre_redes) se pueden gestionar de forma muy sencilla. Con el primero comando crearemos la red, y con el segundo añadiremos todos los contenedores que queramos.

> docker network create <nombre de la red>

> docker network connect <nombre de la red> <nombre del contenedor>

Notad que tenemos que añadir el nombre del contenedor, no el nombre de la imagen. Para eso, [crearemos el contenedor](https://docs.docker.com/engine/reference/commandline/container_create/) sin arrancarlo, así podremos añadirlo a la red para cuando se arranque, pueda comunicarse.

> docker container create --name <nombre del contenedor> <nombre de la imagen>

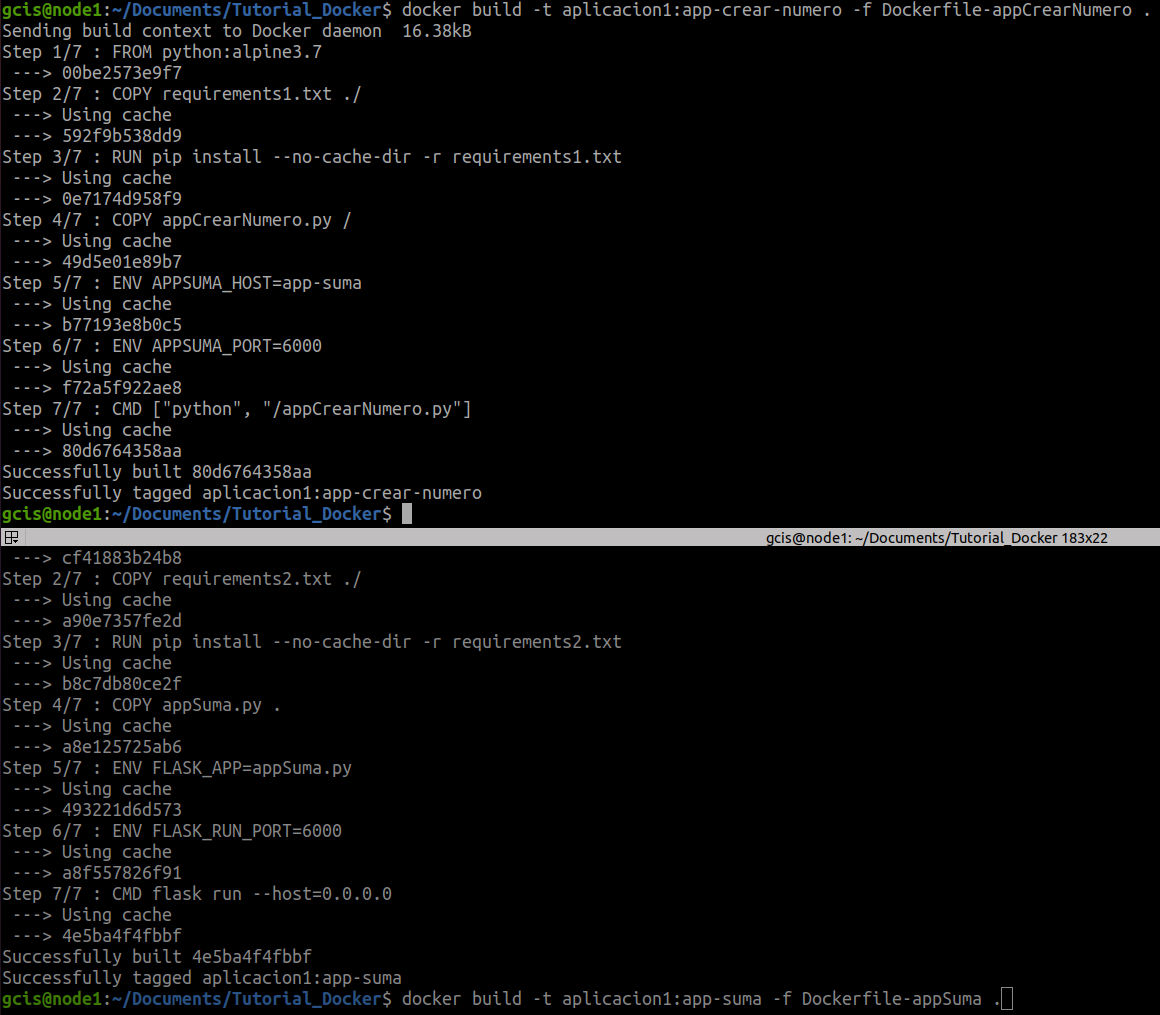
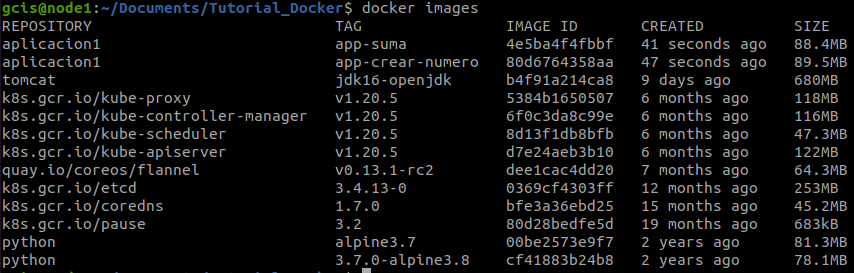
## Puesta en marcha

Vamos a poner la aplicación en marcha. Como hemos comentado anteriormente, lo primero será crear las imágenes de appCrearNumero y appSuma. Para ello, llevaremos todos los ficheros a un directorio (para hacerlo más fácil), y crearemos las imágenes:

> docker build -t aplicacion1:app-crear-numero -f Dockerfile-appCrearNumero .

> docker build -t aplicacion1:app-suma -f Dockerfile-appSuma .

Este será el resultado:



Después, crearemos los contenedores, pero sin arrancarlos:

> docker container create --name app-suma aplicacion1:app-suma

> docker container create --name app-crear-numero aplicacion1:app-crear-numero

Una vez hecho, los añadiremos a la misma red:

> docker network create red-tutorial

> docker network connect red-tutorial app-suma

> docker network connect red-tutorial app-crear-numero

Para arrancar los contenedores, utilizaremos [*docker start*](https://docs.docker.com/engine/reference/commandline/start/). Al principio del tutorial hemos visto como con *docker run* se podía hacer. No lo vamos a hacer con esa forma ya que la opción *run* crea y arranca el contenedor de seguido, por lo que no nos daría tiempo a meterlo en la red. Aun así, es la mejor opción para arrancar contenedores. Con la opción *start* Docker arrancará los contenedores, pero no nos enseñará sus consolas. Para eso utilizaremos el comando [*docker logs*](https://docs.docker.com/engine/reference/commandline/logs/).

> docker start app-suma

> docker start app-crear-numero

> docker logs -f app-suma

> docker logs -f app-crear-numero

*Nota: Mejor arrancar antes appSuma, para cuando se arranque appCrearNumero el servidor ya este funcional.*

Para borrar los contenedores basta con usar *docker rm <nombre contenedor>*, pero si no deja ya que el contenedor está arrancado, primero lo pararemos (*docker stop*) y después se podrá borrar.

> docker stop app-suma

> docker rm app-suma

> docker stop app-crear-numero

> docker rm app-crear-numero

Resultado final:

