Despliegue de Aplicaciones Web

Tema 4: Docker II

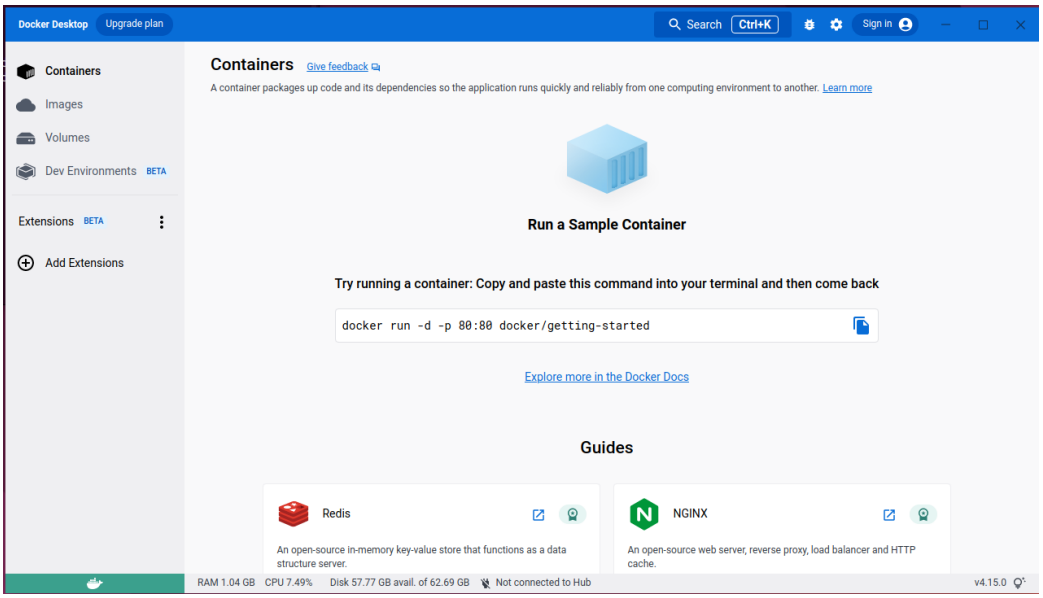
1. Introducción

En esta unidad se va a continuar viendo Docker, pero haciendo uso de Docker Desktop, una aplicación nativa Docker con todas las herramientas Docker disponibles y que ofrece una interfaz gráfica al usuario.

Esta herramienta está disponible para sistemas Windows, Linux y Mac. Por ello, lo primero que se va a hacer es instalar Docker Desktop siguiendo las instrucciones que se pueden encontrar en la página oficial de Docker https://docs.docker.com/desktop/ teniendo en cuenta el sistema operativo que tienes instalado. Después, para verificar que la instalación ha terminado de forma satisfactoria, se puede comprobar la versión instalada con el ya conocido docker --version, y ejecutar la imagen hello-world.

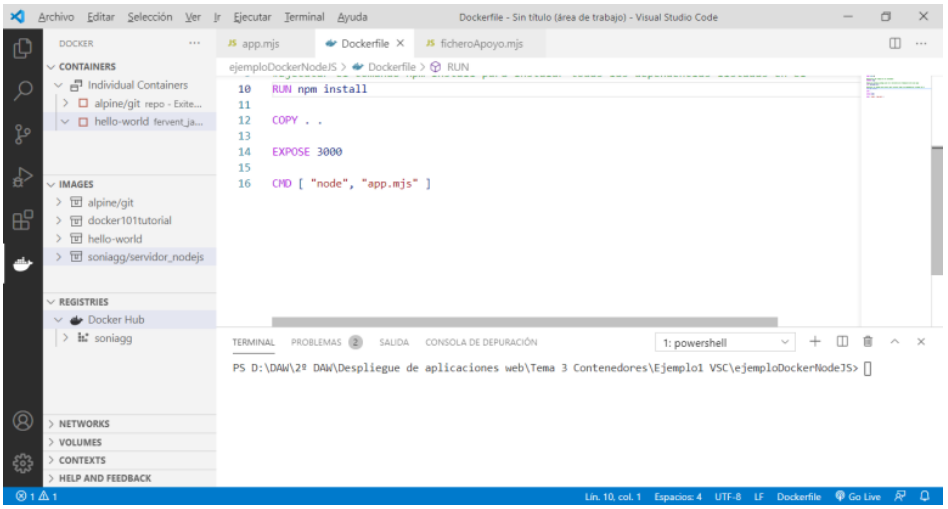
Al instalar Docker Desktop se está instalando, entre otras, Docker Engine, Docker CLI client y Docker Compose.

Una vez instalado, podemos acceder a través de la interfaz gráfica que provee:



2. Docker con VSC

Una vez instalado Docker Desktop, se va a empezar a utilizar VSC como IDE para los pequeños proyectos que se van a manejar. Es recomendable instalar la extensión de Docker, con la que se podrán ver contenedores e imágenes en el mismo VSC, que además será de ayuda a la hora de escribir instrucciones:



En el apartado REGISTRIES se puede realizar login en dockerhub.

3. Dockerfile

Ejercicio 1

Para este ejercicio vamos a hacer uso del proyecto "Ejemplo Docker NodeJS" del aula virtual. Esta aplicación crea un servidor web que escucha peticiones en el puerto 3000.

- Descarga el proyecto y ábrelo desde VSC.

Para poder ejecutarlo de forma local habría que:

- descargar NodeJS e instalarlo.
- instalar npm para poder tener todas las dependencias del proyecto indicadas en el fichero package.json.
- etc.

En nuestro caso, que es trabajar con Docker, ya sabemos que para construir la imagen del proyecto es necesario crear el Dockerfile con todos los pasos de descargas, dependencias, copias de código, etc. Vamos a repasar directivas Dockerfile ya conocidas y ver alguna nueva:

- En el fichero Dockerfile ya incluido en el proyecto:
  - ¿Qué hace la directiva FROM, que ya conoces, con el valor node:14?
  - Busca para que sirve la directiva WORKDIR.
  - ¿En qué directorio indica que se copie el fichero package.json?
  - La directiva RUN indica alguna acción con npm, ¿Cuál es esa acción?
  - Busca la funcionalidad de las directivas EXPOSE y CMD.

**Es importante entender la diferencia entre la instrucción RUN y la instrucción CMD:

- RUN: instrucción que se ejecuta al construir una imagen, para realizar una acción, creando una capa nueva.
- CMD: se encarga de pasar valores predeterminados a un contenedor, los cuales se ejecutarán una vez que el contenedor se inicialice. Solo puede haber una instrucción CMD en un Dockerfile, y debería ser la última instrucción del mismo. Si se incluye más de un CMD, solo el último CMD tendrá efecto.

**Por lo tanto, la diferencia entre CMD y RUN es que lo incluido con RUN se ejecutará al crear la imagen, mientras que lo incluido con CMD se ejecutará cuando se arranque un contenedor basado en la imagen.

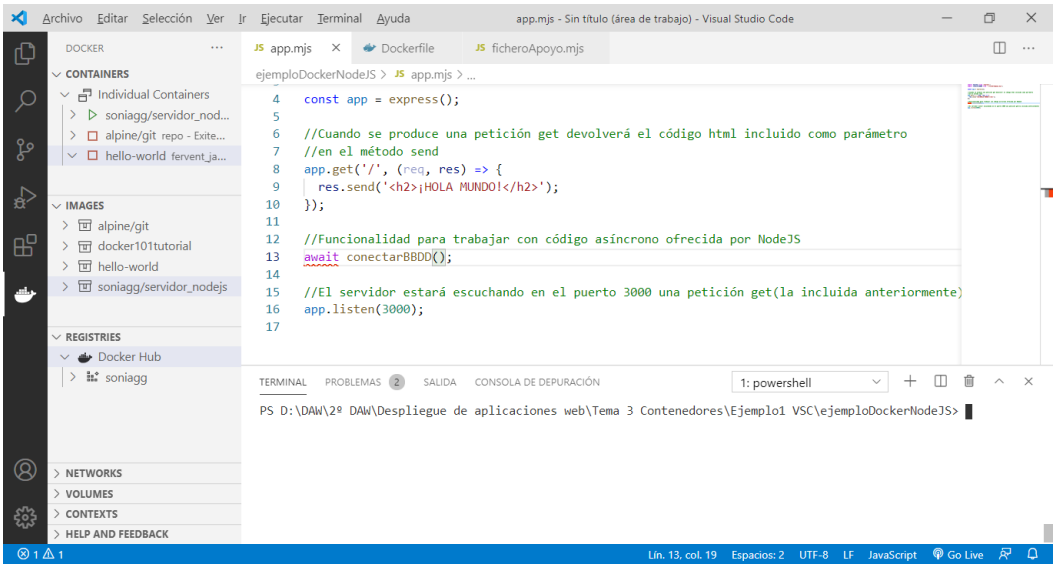
Continuamos:

Abrimos un terminal en el mismo VSC, y creamos la imagen a partir del fichero Dockerfile. Hay que asegurarse de que Docker está ejecutándose:

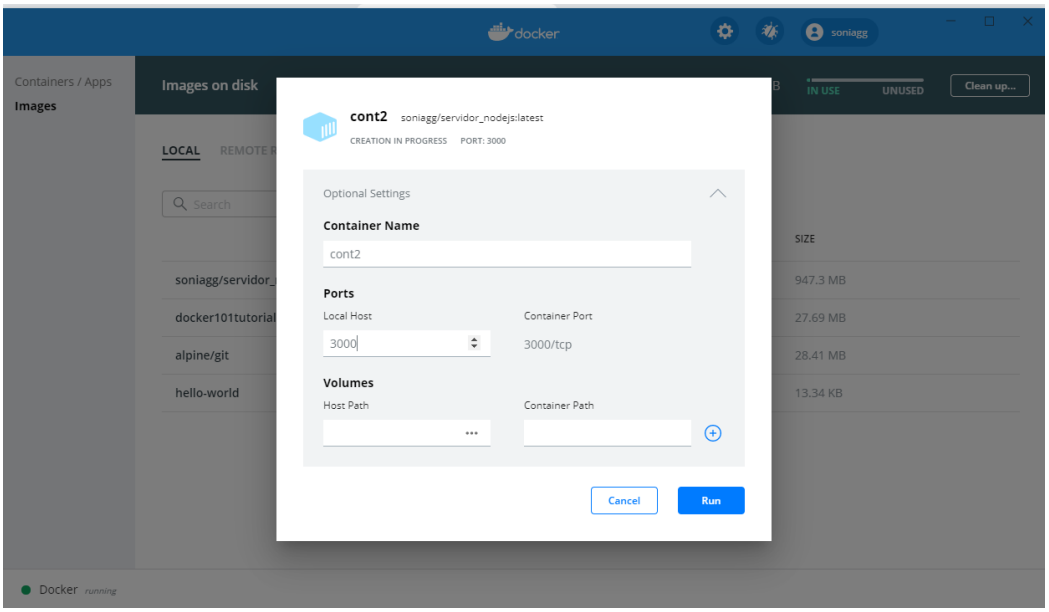
- Crea la imagen haciendo uso de alguno de los comandos docker ya conocidos.
- Comprueba que, efectivamente se ha creado. Puedes ver sus características tanto en la pestaña Docker de VSC como en Docker Desktop.

Para ejecutar un contenedor a partir de la imagen recién creada, existe la posibilidad de hacerlo desde diferentes puntos. Hay que tener en cuenta que será necesario publicar la exposición del puerto 3000 que se ha configurado para escuchar, ya que por defecto no hay conexión entre el contenedor y el sistema operativo host:

- Desde el propio terminal abierto en VSC, haciendo uso del comando docker run ya conocido (con todas las opciones que sean necesarias). Tras lanzarlo, se podrá ver en la pestaña de Docker, en el apartado CONTAINERS, el nuevo contenedor.



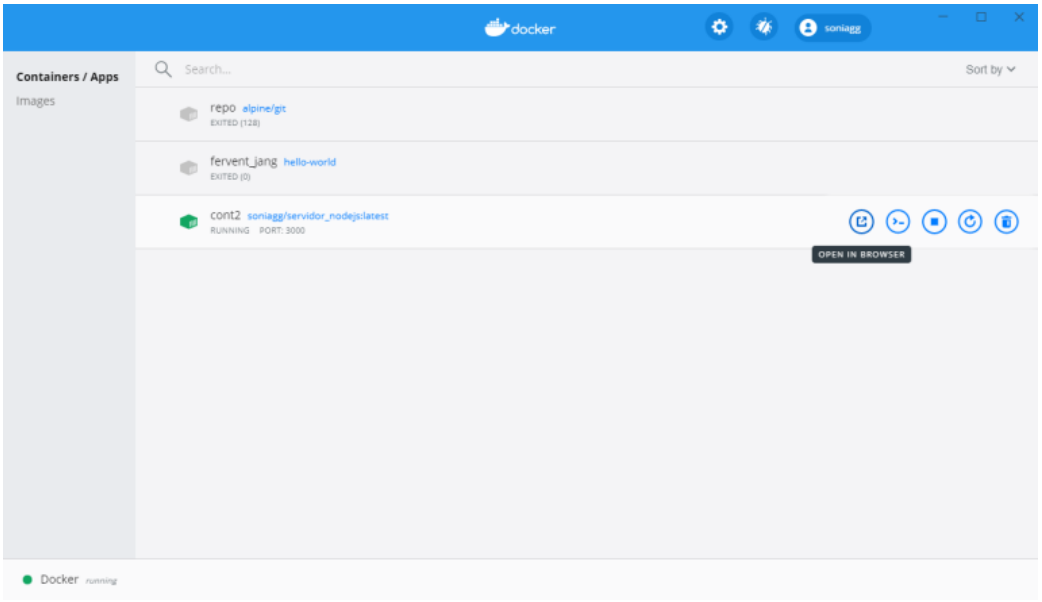
- Desde la pestaña Images en Docker Desktop. Posicionándose sobre la imagen a lanzar, aparecerá un botón RUN. Al pulsarlo, se permitirá completar otros datos que pueden ser necesarios para la ejecución:



Una vez arrancado el contenedor, al acceder desde el navegador a http://localhost:3000/, se mostrará el mensaje ¡HOLA MUNDO!.

Para parar el contenedor, ya conocemos el comando necesario. Además, desde VSC, es posible pararlo haciendo click con el botón derecho sobre el contenedor, en la sección CONTAINERS de la pestaña de Docker. Igualmente, para eliminar el contenedor se puede hacer desde la terminal o desde la pestaña de Docker.

Por último, desde Docker Desktop, es posible parar o eliminar un contenedor. E, incluso, es posible abrir el contenedor en el navegador con un solo clic:



- Ejecuta dos contenedores de la imagen del proyecto con el que estás trabajando:
  - Uno desde la terminal de VSC y otro desde Docker Desktop.
  - Haz accesible uno a través del puerto 3000 y otro del 3001(de la máquina host)
  - Intenta acceder a ellos desde el navegador.
  - Comprueba que ambos aparecen en la lista de contenedores tanto de VSC como de Docker Desktop.
  - Desde VSC, busca el nombre de cada uno de los contenedores.

- Una vez terminado el punto anterior:
  - Prueba a borrar la imagen utilizada desde Docker Desktop.
  - Si no te ha dejado en el paso anterior, realiza todas las acciones necesarias para poder después eliminarla.

Ejercicio 2

A partir de "Ejercicio 2 Docker" que encontrarás en el Aula Virtual, tienes que formar un fichero Dockerfile, para después poder crear una imagen a partir de la cual se puedan ejecutar contenedores con dicha aplicación. Algunos datos a tener en cuenta son:

- Es un proyecto NodeJS. En el código encontrarás algunos comentarios que pueden ayudarte a entender la funcionalidad de la aplicación.
- En el fichero package.json (fichero 100% nodeJS) se incluye la descripción de la aplicación nodeJS. De aquí, lo que nos interesa son las dependencias necesarias ya que habrá que incluirlas en el fichero Dockerfile.
- Acciones que es necesario definir en Dockerfile:
  - Primero habrá que indicar la imagen base de la que se va a partir, que en este caso será la oficial de nodeJS.
  - Como segunda acción a realizar, es necesario indicar el directorio de trabajo del contenedor. Este directorio de trabajo será donde se copien todos los archivos, carpetas, etc. necesarios, será también donde se ejecuten los comandos que hagan falta para el correcto funcionamiento de la aplicación, etc. En este caso, por ejemplo, se puede establecer como directorio de trabajo ./app

- Después, Docker necesitará saber que ficheros, carpetas, etc. será necesario incluir en la imagen. Teniendo en cuenta las matizaciones que se incluyen a continuación, se tendrá que detallar QUÉ hay que copiar en la imagen y DÓNDE:
  - Habrá que indicarle que copie todos los ficheros, carpetas, etc. necesarios del path externo al contenedor, que suele ser el mismo en el que se está creando el Dockerfile (Docker excluirá Dockerfile automáticamente y no lo copiará), al path dentro de la imagen que se quiera.
  - Toda imagen, dependiendo de la imagen base de la que parta, tendrá su sistema de ficheros. Suele ser recomendable no usar el directorio raíz para incluir el código y sí crear una subcarpeta propia (en este caso, el directorio de trabajo que se ha indicado en el paso anterior).
- Seguidamente, será necesario indicar la instalación de npm (esto se tiene que hacer para poder instalar todas las dependencias que se indican en el fichero package.json). Este paso hay que hacerlo ya que es lo mismo que tendría que hacerse si se fuera a ejecutar la aplicación sin contenedores.
- Este paso se realiza indicando que ha de ejecutarse un comando. El comando será npm install, y se ejecutará en el directorio de trabajo indicado en el segundo paso.
- Una vez establecida toda la configuración anterior, hay que dar la instrucción de que, cuando se ejecute un contenedor a partir de esta imagen, arranque el servidor creado. El arranque se indica con el comando node server.js (node es un comando propio de nodeJS, que se podrá ejecutar porque estamos creando la imagen a partir de la imagen base de node). Importante tener en cuenta que esta instrucción tendrá que ejecutarse al arrancar un contenedor y no al crear la imagen.
- Por último, para poder ver la aplicación, será necesario exponer el puerto concreto del contenedor a la máquina host. ** Esta instrucción, aunque aquí explicada la última, no debería estar en esta posición. Ponla en la posición correcta.

➢ Una vez definido el Dockerfile:
- Crea una imagen a partir del mismo.
- Ejecútala.
- Comprueba que la aplicación funciona, accediendo desde el navegador.
- Ejecuta de nuevo la imagen, sin parar ni eliminar el contenedor creado anteriormente.
- Comprueba que se puede acceder a la aplicación de cada contenedor desde el navegador.

4. Imágenes basadas en capas

Las imágenes, como ya se ha mencionado anteriormente, son de solo lectura. Una vez que se ha generado una imagen, el código interno no podrá modificarse. Si se realizan cambios en la aplicación, habrá que generar una imagen nueva.

Las imágenes, por otro lado, están basadas en capas:

- Cada instrucción incluida en Dockerfile es una capa, capa que Docker evaluará en la generación de la imagen.
- Al generar la imagen, se cacheará cada una de esas capas.
- Si se genera una nueva imagen a partir del mismo Dockerfile, y no se han realizado modificaciones, la generación tardará muy poco ya que Docker utilizará las capas cacheadas y no ejecutará de nuevo las instrucciones desde cero. Poniendo como ejemplo el ejercicio 2:

```
PS D:\DAW\2º DAW\Despliegue de aplicaciones web\Tema 3 Contenedores\Ejercicios\SolucionEjercicio2Docker> docker
image build -t soniagg/servidor_nodejs2 .
[+] Building 3.4s (10/10) FINISHED
 => [internal] load build definition from Dockerfile                                                0.1s
 => => transferring dockerfile: 32B                                                                 0.0s
 => [internal] load .dockerignore                                                                   0.1s
 => => transferring context: 2B                                                                     0.0s
 => [internal] load metadata for docker.io/library/node:latest                                      2.9s
 => [auth] library/node:pull token for registry-1.docker.io                                         0.0s
 => [1/4] FROM docker.io/library/node@sha256:e29e61c883c1f1a5f5d997c7713e0ea65ada79e5505d05b497331c055219  0.0s
 => [internal] load build context                                                                   0.1s
 => => transferring context: 185B                                                                   0.0s
 => CACHED [2/4] WORKDIR /app                                                                       0.0s
 => CACHED [3/4] COPY . .                                                                           0.0s
 => CACHED [4/4] RUN npm install                                                                    0.0s
 => exporting to image                                                                              0.1s
 => => exporting layers                                                                             0.0s
 => => writing image sha256:decbcebb08ff9cb3a56badd0c4262279d37026eee09c99141d4ac5b394ec78c4        0.0s
 => => naming to docker.io/soniagg/servidor_nodejs2                                                 0.0s
```

- Si se realiza un cambio por ejemplo, en código, cacheará las capas no modificadas que se encuentren ANTES de la capa que si hay que rehacer, y generará el resto a partir de la capa que si incluye cambios:

```
PS D:\DAW\2º DAW\Despliegue de aplicaciones web\Tema 3 Contenedores\Ejercicios\SolucionEjercicio2Docker> docker
image build -t soniagg/servidor_nodejs3 .

[+] Building 17.0s (10/10) FINISHED
 => [internal] load build definition from Dockerfile                                                0.2s
 => => transferring dockerfile: 590B                                                                0.1s
 => [internal] load .dockerignore                                                                   0.1s
 => => transferring context: 2B                                                                     0.0s
 => [internal] load metadata for docker.io/library/node:latest                                      5.5s
 => [auth] library/node:pull token for registry-1.docker.io                                         0.0s
 => [1/4] FROM docker.io/library/node@sha256:e29e61c883c1f1a5f5d997c7713e0ea65ada79e5505d05b497331c055219  0.0s
 => [internal] load build context                                                                   0.1s
 => => transferring context: 2.53kB                                                                 0.0s
 => CACHED [2/4] WORKDIR /app                                                                       0.0s
 => [3/4] COPY . .                                                                                  0.1s
 => [4/4] RUN npm install                                                                          10.1s
 => exporting to image                                                                              0.6s
 => => exporting layers                                                                             0.5s
 => => writing image sha256:af5a4d5ba3920adb74dcbd3da8941d05247b145fa77fb9de94b962f048d142b3        0.0s
 => => naming to docker.io/soniagg/servidor_nodejs3                                                 0.0s
```

Ejercicio 3

➢ Partiendo del ejercicio 2, haz las modificaciones que consideres necesarias para que, al realizar cambios en el código de la aplicación, la instalación de npm no se ejecute entera de nuevo, sino que tire de caché. Ten en cuenta que se necesitará la información almacenada en el fichero package.json para la instalación de npm.

5. Más comandos y opciones

docker attach

Comando que permite, indicando un contenedor concreto, vincular la salida, entrada y salida de errores estándar del mismo, al terminal en el que se ejecute. Se tendrá que incluir el id o el nombre del contenedor a vincular. Cuando se ejecuta un contenedor sin la opción -d, el contenedor se ejecuta, por defecto, en modo vinculado.

docker logs

Haciendo uso de este comando, se pueden ver los logs que se han imprimido de un contenedor en concreto. Se ha de indicar el contenedor mediante su nombre o su id.

Con la opción -f, se puede activar el modo seguimiento de logs.

Ejercicio 4

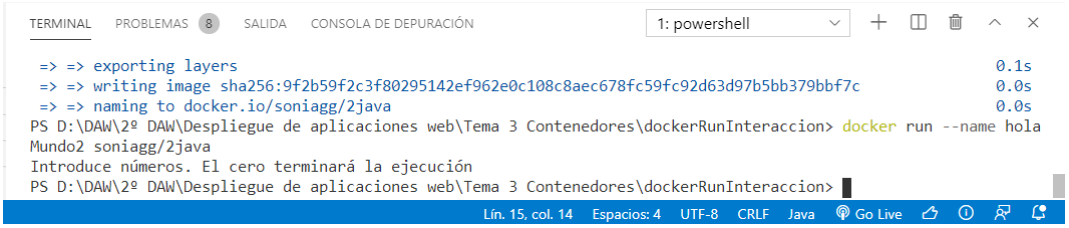
- Arranca, desde el terminal, un contenedor de la imagen creada en el ejercicio 3, no lo hagas en segundo plano:
  - Accede a la aplicación desde el navegador.
  - Comprueba que, cada vez que se modifica el objetivo, aparece un log en el terminal, que será el texto que muestra la línea console.log(nuevoObjetivo);
- Arranca ahora, un nuevo contenedor (sin parar el otro), esta vez en segundo plano:
  - Comprueba que no aparece ningún log.
- Ahora, para el segundo contenedor, actualiza el objetivo, desde el navegador, tres veces. Tras esas acciones, muestra los logs en el terminal con el comando correcto.
- Después, activa el seguimiento de logs en tiempo real. ¿Cómo saldrías del seguimiento?

docker run –it

Al ejecutar un contenedor, puede que la aplicación en concreto no sea accesible a través del navegador. En ese caso, se podrá ver en consola su ejecución, pero no interactuar con ella.

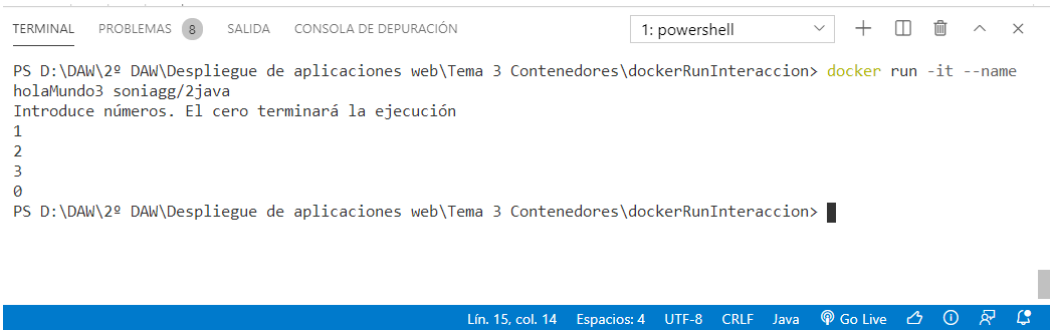
Con la opción -i se mantiene el canal de entrada (stdin) del contenedor, abriendo el contenedor en modo interactivo, y con -t, se asocia una tty (terminal). Es posible incluir las dos opciones como -it.

En la siguiente imagen se ve la ejecución de un programa que pide un número por pantalla pero que termina su ejecución sin permitir al usuario interactuar con la aplicación.



Sin embargo, ejecutando un contenedor de la misma imagen con las opciones -it, se mantiene el canal de comunicación abierto y no se para el contenedor hasta que no se da la

circunstancia necesaria para terminar la ejecución del programa.



Cuando la ejecución ha terminado, el contenedor pasa a estar parado. Si se reiniciara de la forma ya conocida (docker start), ya no existiría comunicación entre el programa y el usuario. Habría que reiniciar el contenedor en modo vinculado, haciendo uso de las opciones correctas para tener acceso tanto al canal de entrada, como al de salida y al de error.

docker run –-rm

Opción por la que se eliminará el contenedor cuando termine de ejecutarse. El contenedor no pasará de ejecución a parado, sino que desaparecerá.

Ejercicio 5

- Descarga el proyecto “Interacción con contenedores” que encontrarás en el aula virtual.
  - Vemos juntos el código, qué hace el programa y el fichero Dockerfile.
- Crea la imagen y arranca un contenedor de tal manera que se pueda hacer uso del programa.
- Una vez parado el contendor tras la ejecución, reinícialo.
  - ¿Qué opciones necesitas para poder volver a utilizar el programa? ¿Qué hace cada una de las opciones?
- Inicia un nuevo contenedor de la misma imagen con la opción --rm. Comprueba que, una vez terminada la ejecución del programa, el contenedor desaparece.

docker image inspect

Comando que, con el identificador de una imagen que se le pasa como argumento, mostrará información de la misma.

Se podrá ver información de las capas que conforman la imagen (las suyas y las de la imagen base), configuración para los futuros contenedores que se ejecuten a partir de ella, sistema operativo interno de la imagen, etc.