TP 2 - DOCKER

1- Instalar Docker Community Edition
Ejecutar el siguiente comando para comprobar versiones de cliente y demonio.

```
PS C:\Users\sofia> docker version
Client:
 Cloud integration: v1.0.28
 Version:           20.10.17
 API version:       1.41
 Go version:        go1.17.11
 Git commit:        100c701
 Built:             Mon Jun  6 23:09:02 2022
 OS/Arch:           windows/amd64
 Context:           default
 Experimental:      true

Server: Docker Desktop 4.11.1 (84025)
 Engine:
  Version:          20.10.17
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.17.11
  Git commit:       a89b842
  Built:            Mon Jun  6 23:01:23 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.6
  GitCommit:        10c12954828e7c7c9b6e0ea9b0c02b01407d3ae1
 runc:
  Version:          1.1.2
  GitCommit:        v1.1.2-0-ga916309
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

2- Explorar DockerHub
Registrase en docker hub: https://hub.docker.com/
Familiarizarse con el portal
3- Obtener la imagen BusyBox
Ejecutar el siguiente comando, para bajar una imagen de DockerHub

```
PS C:\Users\sofia> docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
Digest: sha256:3fbc632167424a6d997e74f52b878d7cc478225cffac6bc977eedfe51c7f4e79
Status: Image is up to date for busybox:latest
docker.io/library/busybox:latest
```

Verificar qué versión y tamaño tiene la imagen bajada, obtener una lista de imágenes locales:

```
PS C:\Users\sofia> docker images
REPOSITORY                          TAG                 IMAGE ID       CREATED         SIZE
example-voting-app_result           latest              8d55cdf23633   7 days ago      266MB
example-voting-app_worker           latest              504fbb564b6b   7 days ago      194MB
example-voting-app_vote             latest              c80544d2fad5   7 days ago      164MB
postgres                            15-alpine           ab8fb914369e   11 days ago     237MB
redis                               alpine              6c09b0364aa8   2 weeks ago     30.2MB
<none>                              <none>              e5ff3ac494fc   2 weeks ago     216MB
<none>                              <none>              bd3e342bcaf4   2 weeks ago     216MB
mywebapi                            latest              7e69e58d5556   2 weeks ago     216MB
busybox                             latest              a416a98b71e2   5 weeks ago     4.26MB
```

4- Ejecutando contenedores

Especificamos algún comando a correr dentro del contendor, ejecutar por ejemplo:

```
PS C:\Users\sofia> docker run busybox echo "Hola Mundo"
Hola Mundo
PS C:\Users\sofia>
```

Ver los contendores ejecutados utilizando el comando ps:

```
PS C:\Users\sofia> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
PS C:\Users\sofia>
```

Vemos que no existe nada en ejecución, correr entonces:

```
PS C:\Users\sofia> docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS                          PORTS     NAMES
d13e21ae8898   busybox   "echo 'Hola Mundo'"      38 seconds ago       Exited (0) 36 seconds ago                 blissful_pasteur
e5222ddab950   busybox   "sh"                     About a minute ago   Exited (0) About a minute ago             stupefied_golick
```

El comando docker ps -a se utiliza para mostrar una lista de todos los contenedores de Docker en el sistema, incluyendo los que están detenidos o en ejecución. La salida del comando incluye:

CONTAINER ID: El identificador único asignado a cada contenedor.
IMAGE: La imagen de Docker que se utilizó para crear el contenedor.
COMMAND: El comando que se ejecuta dentro del contenedor.
CREATED: La fecha y hora en que se creó el contenedor.
STATUS: El estado actual del contenedor (en ejecución, detenido, etc.).
PORTS: Los puertos expuestos y mapeados desde el contenedor al host.
NAMES: El nombre asignado al contenedor.

5- Ejecutando en modo interactivo

Para cada uno de los siguientes comandos dentro de contenedor, mostrar los resultados:

```
/ # ps
PID   USER     TIME  COMMAND
    1 root      0:00 sh
   11 root      0:00 ps
/ # free
              total        used        free      shared  buff/cache   available
Mem:        3666808      609652     1768112        1736     1289044     2896996
Swap:       1048576           0     1048576
/ # uptime
 16:30:33 up 7 min,  0 users,  load average: 0.04, 0.04, 0.00
/ # ls -l /
total 40
drwxr-xr-x    2 root     root         12288 Jul 17 18:30 bin
drwxr-xr-x    5 root     root           360 Aug 23 16:29 dev
drwxr-xr-x    1 root     root          4096 Aug 23 16:29 etc
drwxr-xr-x    2 nobody   nobody        4096 Jul 17 18:30 home
drwxr-xr-x    2 root     root          4096 Jul 17 18:30 lib
lrwxrwxrwx    1 root     root             3 Jul 17 18:30 lib64 -> lib
dr-xr-xr-x  297 root     root             0 Aug 23 16:29 proc
drwx------    1 root     root          4096 Aug 23 16:29 root
dr-xr-xr-x   11 root     root             0 Aug 23 16:29 sys
drwxrwxrwt    2 root     root          4096 Jul 17 18:30 tmp
drwxr-xr-x    4 root     root          4096 Jul 17 18:30 usr
drwxr-xr-x    4 root     root          4096 Jul 17 18:30 var
```

6- Borrando contendores terminados

Para borrar todos los contendores que no estén corriendo, ejecutar cualquiera de los siguientes comandos:

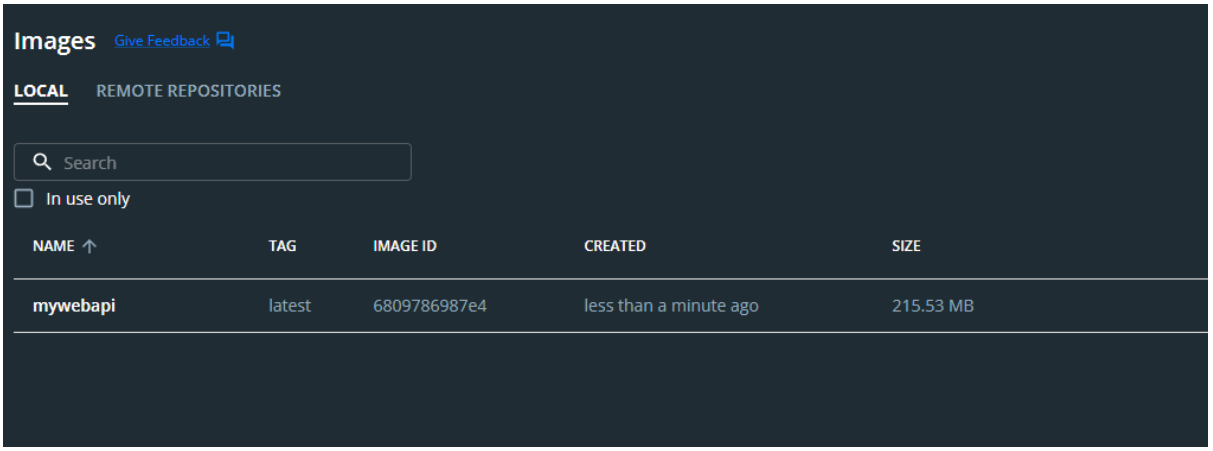
```
PS C:\Users\sofia> docker rm $(docker ps -a -q -f status=exited)
c82d65b7ef62
d13e21ae8898
e5222ddab950
360a8257018f
eb4bad43a7e0
b26a4c894205
d5aa20d5ae7b
f7de19897cfc
242f79e6988b
3e120bda7a9f
d9615be38a28
05468a216409
3cf36d1c6640
PS C:\Users\sofia> docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
PS C:\Users\sofia>
```

7- Construir una imagen

A partir del código https://github.com/ingsoft3ucc/SimpleWebAPI crearemos una imagen.

Clonar repo

Crear imagen etiquetándola con un nombre. El punto final le indica a Docker que use el dir actual



8- Publicando puertos
En el caso de aplicaciones web o base de datos donde se interactúa con estas aplicaciones a través de un puerto al cual hay que acceder, estos puertos están visibles solo dentro del contenedor. Si queremos acceder desde el exterior deberemos exponerlos.

Ejecutar la siguiente imagen, en este caso utilizamos la bandera -d (detach) para que nos devuelva el control de la consola:
docker run --name myapi -d mywebapi
Ejecutamos un comando ps:

Vemos que el contendor expone 3 puertos el 80, el 5254 y el 443, pero si intentamos en un navegador acceder a http://localhost/WeatherForecast no sucede nada.

Procedemos entonces a parar y remover este contenedor:

```
PS C:\Users\sofia\OneDrive\Escritorio\SimpleWebAPI> docker kill myapi
myapi
PS C:\Users\sofia\OneDrive\Escritorio\SimpleWebAPI> docker rm myapi
myapi
```

Vamos a volver a correrlo otra vez, pero publicando los puertos 80 y 5254

Accedamos nuevamente a http://localhost/WeatherForecast y a http://localhost/swagger/index.html y expliquemos que sucede.