

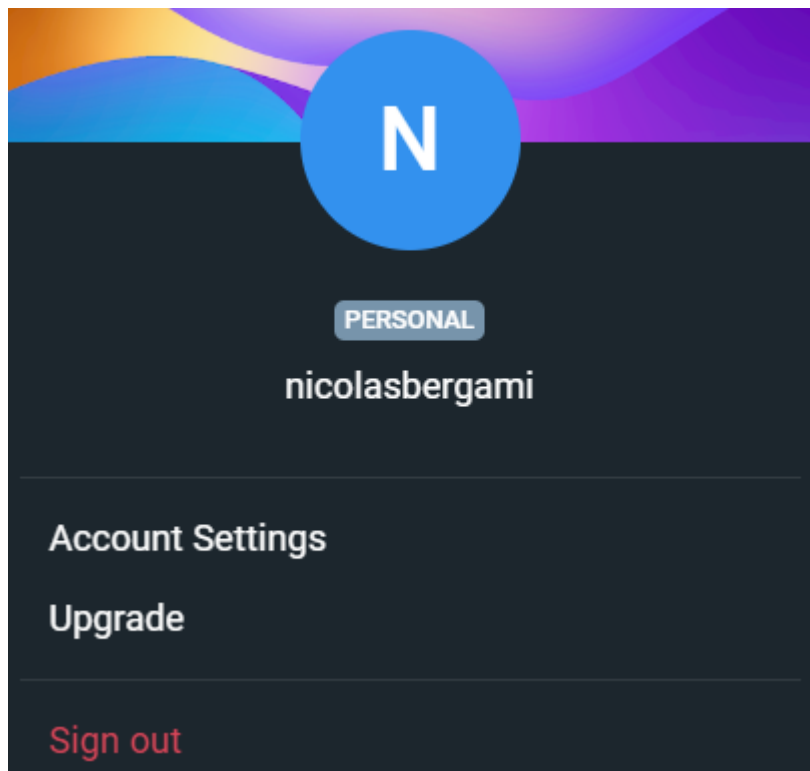
TP2-DOCKER

☐ Verificar Docker Version

```
C:\Users\nicol>docker version
Client:
 Version:           26.1.4
 API version:       1.45
 Go version:        go1.21.11
 Git commit:        5650f9b
 Built:             Wed Jun  5 11:29:54 2024
 OS/Arch:           windows/amd64
 Context:           desktop-linux
error during connect: Get "http://%2F%2F.%2Fpipe%2FdockerDesktopLinuxEngine/v1.45/version": open //./pipe/dockerDesktopLinuxEngine: The system cannot find the file specified.

C:\Users\nicol>
```

2) Registrarse en Docker



3) Obtener la imagen BusyBox: Ejecutar el siguiente comando, para bajar una imagen de DockerHub

☐ docker pull busybox

```
C:\Users\nicol>docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
ec562eabd705: Pull complete
Digest: sha256:9ae97d36d26566ff84e8893c64a6dc4fe8ca6d1144bf5b87b2b85a32def253c7
Status: Downloaded newer image for busybox:latest
docker.io/library/busybox:latest

What's next:
View a summary of image vulnerabilities and recommendations → docker scout quickview busybox
```

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

☐ docker images

```
C:\Users\nicol>docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ucc-as2-final-frontend	latest	e6c0064d2ebc	4 weeks ago	1.58GB
ucc-as2-final-containers-api	latest	440b70fe4001	4 weeks ago	1.29GB
ucc-as2-final-user-res-api	latest	51c1c0b2d144	4 weeks ago	1.3GB
ucc-as2-final-search-api	latest	5ee00b6671d2	4 weeks ago	1.27GB
ucc-as2-final-hotels-api	latest	4da922a11123	4 weeks ago	1.37GB
proyectofinal-frontend	latest	c65687f8da34	4 weeks ago	1.23GB
<none>	<none>	549a33905794	4 weeks ago	1.23GB
<none>	<none>	28922ed70953	4 weeks ago	1.23GB
<none>	<none>	ce24840253a3	4 weeks ago	1.23GB
<none>	<none>	a570c522ade1	4 weeks ago	1.23GB
<none>	<none>	37685a8f41b6	4 weeks ago	1.23GB
<none>	<none>	504e559c4dc8	4 weeks ago	1.22GB
<none>	<none>	71ad786cc169	4 weeks ago	1.22GB
<none>	<none>	a30bc28b7f73	4 weeks ago	1.22GB
proyectofinal-usersdb	latest	327b3af2e5c7	4 weeks ago	583MB
<none>	<none>	e789e2cbc8da	5 weeks ago	1.22GB
proyectofinal-users	latest	fdf6c3b9a769	5 weeks ago	1.29GB
proyectofinal-consumer	latest	35043d6badfe	5 weeks ago	1GB
proyectofinal-back	latest	e839b7be66df	5 weeks ago	1.36GB
proyectofinal-messages	latest	08f14d5d03b1	5 weeks ago	1.36GB
proyectofinal-search	latest	1547fe9e22e4	5 weeks ago	1.26GB
<none>	<none>	3dcaa90853a	5 weeks ago	583MB
mysql	latest	7ce93a845a8a	5 weeks ago	586MB
arqsw2	latest	329a9cce9ae2	8 weeks ago	1.77GB

☐ Ejecutar un contenedor utilizando el comando run

☐ docker run busybox

```
C:\Users\nicol>docker run busybox

C:\Users\nicol>
```

El comando `docker run busybox` no mostró ningún resultado porque BusyBox, al no recibir un comando específico, se ejecuta y finaliza inmediatamente sin realizar ninguna acción visible. Para ver un resultado, puedes pasar un

comando como `docker run busybox ls`, que listará el contenido del directorio raíz del contenedor antes de cerrarse.



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

- ☐ `docker run busybox echo "Hola Mundo"`
- ☐ Ver los contenedores ejecutados utilizando el comando `ps`:
- ☐ `docker ps`
- ☐ Vemos que no existe nada en ejecución, correr entonces:
- ☐ `docker ps -a`
- ☐ Mostrar el resultado y explicar que se obtuvo como salida del comando anterior.

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

C:\Users\nicol>docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
0f30cfcf8d9e   busybox    "echo 'Hola Mundo'"     26 seconds ago Exited (0)    25 seconds ago competent_carver
```

El comando `docker ps -a` te muestra que el contenedor basado en BusyBox se ejecutó con éxito, imprimió "Hola Mundo" y luego se detuvo. El estado `Exited (0)` indica que el contenedor terminó su tarea sin errores.

El comando muestra todos los contenedores, activos o inactivos, lo cual es útil para verificar que un contenedor se ejecutó como esperabas, incluso si terminó rápidamente.

5)Ejecutando en modo interactivo

- ☐ Ejecutar el siguiente comando
`docker run -it busybox sh`
- ☐ Para cada uno de los siguientes comandos dentro de contenedor, mostrar los resultados:

`ps`
`uptime`
`free`
`ls -l /`

- ☐ Salimos del contenedor con:

exit

```
C:\Users\nicol>docker run -it busybox sh
/ #
/ #
/ # ps
PID    USER      TIME  COMMAND
   1   root         0:00   sh
   8   root         0:00   ps
/ # uptime
18:27:37 up 38 min,  0 users,  load average: 0.00, 0.00, 0.00
/ # free
              total        used        free      shared  buff/cache   available
Mem:        7936348        607940       6878608         3428     449800      7093080
Swap:        2097152           0       2097152
/ # ls -l /
total 40
drwxr-xr-x  2 root    root          12288 May 18  2023 bin
drwxr-xr-x  5 root    root           360 Aug 27 18:26 dev
drwxr-xr-x  1 root    root          4096 Aug 27 18:26 etc
drwxr-xr-x  2 nobody nobody        4096 May 18  2023 home
drwxr-xr-x  2 root    root          4096 May 18  2023 lib
lrwxrwxrwx  1 root    root           3 May 18  2023 lib64 -> lib
dr-xr-xr-x 262 root    root           0 Aug 27 18:26 proc
drwx----- 1 root    root          4096 Aug 27 18:27 root
dr-xr-xr-x 11 root    root           0 Aug 27 18:26 sys
drwxrwxrwt  2 root    root          4096 May 18  2023 tmp
drwxr-xr-x  4 root    root          4096 May 18  2023 usr
drwxr-xr-x  4 root    root          4096 May 18  2023 var
/ # exit

C:\Users\nicol>
```

6) Borrando contenedores terminados

- ☐ Obtener la lista de contenedores

docker ps -a

- ☐ Para borrar podemos utilizar el id o el nombre (autogenerado si no se especifica) de contenedor que se desee, por ejemplo:

docker rm elated_lalande

- ☐ Para borrar todos los contenedores que no estén corriendo, ejecutar cualquiera de los siguientes comandos:

docker rm \$(docker ps -a -q -f status=exited)

docker container prune

```

C:\Users\nicol>docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
2e629581b0a2   busybox   "sh"                    3 minutes ago Exited (0)    2 minutes ago          jolly_shaw
0f30cfcf8d9e   busybox   "echo 'Hola Mundo'"    12 minutes ago Exited (0)    12 minutes ago          competent_carver

C:\Users\nicol>docker rm elated_lalande
Error response from daemon: No such container: elated_lalande

C:\Users\nicol>docker rm jolly_shaw
jolly_shaw

C:\Users\nicol>docker rm $(docker ps -a -q -f status=exited)
unknown shorthand flag: 'a' in -a
See 'docker rm --help'.

C:\Users\nicol>docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
0f30cfcf8d9e7c29e4aa4f2bbc7765807233645cec83e62f53b113a763f4c5d9

Total reclaimed space: 0B

C:\Users\nicol>docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
C:\Users\nicol>

```

Conceptos de DockerFile

- ☐ Leer <https://docs.docker.com/engine/reference/builder/>
- ☐ Describir las instrucciones
 - FROM
 - RUN
 - ADD
 - COPY
 - EXPOSE
 - CMD
 - ENTRYPOINT

1. FROM

Descripción:

FROM es la instrucción más básica y es necesaria en cada Dockerfile.

Establece la imagen base desde la cual se construirá la nueva imagen de Docker. Todas las instrucciones subsiguientes en el Dockerfile se aplican sobre esta imagen base.

2. RUN

Descripción:

RUN ejecuta un comando en la imagen en construcción y guarda el resultado en la capa resultante de la imagen. Es comúnmente usado para instalar paquetes de software o realizar tareas de configuración.

3. ADD

Descripción:

ADD copia archivos o directorios desde una fuente (en el sistema local o una URL) a un destino en el sistema de archivos del contenedor. Además, puede

descomprimir archivos automáticamente si son de tipo comprimido reconocido (.tar, .gz, .bz2, etc.).

4. COPY

Descripción:

COPY copia archivos o directorios desde la fuente en el sistema de archivos del host Docker al sistema de archivos del contenedor. A diferencia de ADD, no soporta URLs ni descomprime archivos.

5. EXPOSE

Descripción:

EXPOSE documenta que el contenedor escucha en puertos específicos en tiempo de ejecución. No expone realmente los puertos; es solo una instrucción informativa para el usuario que crea el contenedor.

6. CMD

Descripción:

CMD proporciona comandos predeterminados para ejecutar cuando un contenedor se inicia. Si se proporciona un argumento en la línea de comando docker run, este comando reemplazará a CMD. Solo puede haber una instrucción CMD por Dockerfile. Si se especifican múltiples, solo se utilizará la última.

7. ENTRYPOINT

Descripción:

ENTRYPOINT configura una aplicación que se ejecuta siempre en un contenedor. A diferencia de CMD, ENTRYPOINT no se sobrescribe cuando se pasan argumentos adicionales a docker run, sino que esos argumentos se pasan como argumentos al comando ENTRYPOINT. Puedes combinar ENTRYPOINT y CMD para definir un comando base con argumentos predeterminados.

- ☐ 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

docker build -t mywebapi .

```
C:\Users\nicol\OneDrive\Documentos\universidad\Ingsoft3\IngSoft3\TP2>git clone https://github.com/ingsoft3ucc/SimpleWebAPI
Cloning into 'SimpleWebAPI'...
remote: Enumerating objects: 150, done.
remote: Counting objects: 100% (150/150), done.
remote: Compressing objects: 100% (115/115), done.
Remote: Total 150 (delta 60), reused 33 (delta 9), pack-reused 0 (from 0)
Receiving objects: 100% (150/150), 28.41 KiB | 808.00 KiB/s, done.
Resolving deltas: 100% (60/60), done.
```

```
C:\Users\nicol\OneDrive\Documentos\universidad\Ingsoft3\IngSoft3\TP2\SimpleWebAPI>docker build -t mywebapi .
[+] Building 382.1s (18/18) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 810B
=> [internal] load metadata for mcr.microsoft.com/dotnet/sdk:7.0
=> [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:7.0
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [build 1/7] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:d32bd65cf5843f413e81f5d917057c82da99737cb1637e905a 347.0s
=> => resolve mcr.microsoft.com/dotnet/sdk:7.0@sha256:d32bd65cf5843f413e81f5d917057c82da99737cb1637e905a1a4bc2e7 0.1s
=> => sha256:a291948a5e5ceb50e3eaa75cbe6377660146b385e475938d6855e220afaaeed 2.01kB / 2.01kB
=> => sha256:728328ac3bde9b85225b1f0d60f5c149f5635a191f5d8eae00e095d36ef9fd 31.43MB / 31.43MB
=> => sha256:534ba947de6ac79fd6168f4a93847954b23bab2782700bbfff7f31e61a03e8d4 32.46MB / 32.46MB
=> => sha256:d32bd65cf5843f413e81f5d917057c82da99737cb1637e905a1a4bc2e7ec6c8d 1.79kB / 1.79kB
=> => sha256:ea4f5e0c20952a885a89566fc35cf3295b3228375b715a0f1af9e5a3c0c2eebf 5.29kB / 5.29kB
=> => sha256:82bb7a80de578404d92b5ae5e67f3de90eab30027694d2609be35ad25b09e3bc 14.97MB / 14.97MB
```

☐ Revisar Dockerfile y explicar cada línea

The screenshot shows the Docker Desktop interface for the 'mywebapi:latest' image. The left sidebar contains navigation options: Containers, Images (selected), Volumes, Builds, Docker Scout, and Extensions. The main panel displays the 'Layers (16)' for the image, which is 215.66 MB in size and was created 1 second ago. The layers are listed as follows:

Layer ID	Command	Size
0	ADD file:9b38b383dd93169a663ee...	80.66 MB
1	CMD ["bash"]	0 B
2	ENV ASPNETCORE_URLS=http://+...	0 B
3	RUN /bin/sh -c apt-get update && a...	36.24 MB
4	ENV DOTNET_VERSION=7.0.20	0 B
5	COPY /dotnet /usr/share/dotnet # ...	73.26 MB
6	RUN /bin/sh -c ln -s /usr/share/dot...	24 B
7	ENV ASPNET_VERSION=7.0.20	0 B
8	COPY /shared/Microsoft.AspNetCo...	21.83 MB
9	WORKDIR /app	0 B
10	EXPOSE map[80/tcp:[]]	0 B

Explicación de cada línea:

1. ENTRYPOINT ["dotnet", "SimpleWebAPI.dll"]:

Define el comando que se ejecutará cuando se inicie el contenedor. En este caso, ejecuta la aplicación .NET Core utilizando el archivo SimpleWebAPI.dll.

2. COPY /app/publish . # buildkit:

Copia el contenido del directorio /app/publish al directorio de trabajo del contenedor. # buildkit
parece ser un comentario relacionado con la herramienta de compilación utilizada.

3. WORKDIR /app:

Establece el directorio de trabajo dentro del contenedor. A partir de este punto, todas las acciones se realizarán en el directorio /app.

4. EXPOSE map[5254/tcp:{}]:

Expone el puerto 5254 del contenedor. Esto indica que el contenedor está preparado para recibir tráfico TCP en ese puerto.

5. EXPOSE map[443/tcp:{}]:

Expone el puerto 443, que es el puerto estándar para HTTPS.

6. EXPOSE map[80/tcp:{}]:

Expone el puerto 80, el puerto estándar para HTTP.

7. WORKDIR /app (repetido):

Nuevamente establece el directorio de trabajo en /app.

8. COPY /shared/Microsoft.AspNetCore.App /usr/share/dotnet/shared/...:

Copia los archivos de la aplicación compartida de ASP.NET Core al directorio especificado dentro del contenedor.

9. ENV ASPNET_VERSION=7.0.20:

Establece la variable de entorno ASPNET_VERSION a 7.0.20. Esto especifica la versión de ASP.NET Core que se utilizará.

10. RUN /bin/sh -c ln -s /usr/share/dotnet/dotnet /usr/bin/dotnet # buildkit:

Crea un enlace simbólico (ln -s) para que dotnet esté disponible en el contenedor. Esto facilita la ejecución de comandos dotnet.

11. COPY /dotnet /usr/share/dotnet # buildkit:

Copia los archivos de dotnet desde el directorio fuente al directorio de destino en el contenedor.

12. ENV DOTNET_VERSION=7.0.20:

Establece la variable de entorno DOTNET_VERSION a 7.0.20. Esto define la versión de .NET que se utilizará en el contenedor.

13. RUN /bin/sh -c apt-get update && apt-get install -y...:

Actualiza los paquetes del sistema y luego instala el paquete necesario (apt-get install). Este

comando asegura que el contenedor tenga todas las dependencias necesarias.

14. ``ENV ASPNETCORE_URLS=http://+:80`

`DOTNET_RUNNING_IN_CONTAINER=true`**:`

- Establece dos variables de entorno:

Configura la aplicación para que escuche en el puerto 80 de todas las interfaces (``+``).

Indica que la aplicación está ejecutándose dentro de un contenedor Docker, lo cual puede

cambiar algunos comportamientos por defecto de la aplicación.

15. ``/bin/sh -c #(nop) CMD ["bash"]`**:`

- Este comando es utilizado por Docker internamente para definir la instrucción ``CMD``. Aquí

indica que, si se inicia el contenedor sin especificar otro comando, se abrirá una sesión de bash.

16. ``/bin/sh -c #(nop) ADD file:9b3bB3...`**:`

- El comando ``ADD`` copia archivos del host al sistema de archivos del contenedor. En este

caso, parece estar añadiendo un archivo específico con el identificador, que probablemente es un

archivo de la aplicación o una dependencia importante.

☐ Ver imágenes disponibles

```
C:\Users\nicol>docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mywebapi	latest	19734a3a8873	10 minutes ago	216MB
ucc-as2-final-frontend	latest	e6c0064d2ebc	4 weeks ago	1.58GB
ucc-as2-final-containers-api	latest	440b70fe4001	4 weeks ago	1.29GB
ucc-as2-final-user-res-api	latest	51c1c0b2d144	4 weeks ago	1.3GB
ucc-as2-final-search-api	latest	5ee00b6671d2	4 weeks ago	1.27GB
ucc-as2-final-hotels-api	latest	4da922a11123	4 weeks ago	1.37GB
proyectofinal-frontend	latest	c65687f8da34	4 weeks ago	1.23GB
<none>	<none>	549a33905794	4 weeks ago	1.23GB
<none>	<none>	28922ed70953	4 weeks ago	1.23GB
<none>	<none>	ce24840253a3	4 weeks ago	1.23GB
<none>	<none>	a570c522ade1	4 weeks ago	1.23GB
<none>	<none>	37685a8f41b6	4 weeks ago	1.23GB
<none>	<none>	504e559c4dc8	4 weeks ago	1.22GB
<none>	<none>	71ad786cc169	4 weeks ago	1.22GB
<none>	<none>	a30bc28b7f73	4 weeks ago	1.22GB
proyectofinal-usersdb	latest	327b3af2e5c7	4 weeks ago	583MB
<none>	<none>	e789e2cbc8da	5 weeks ago	1.22GB
proyectofinal-users	latest	fdf6c3b9a769	5 weeks ago	1.29GB
proyectofinal-consumer	latest	35043d6badfe	5 weeks ago	1GB
proyectofinal-back	latest	e839b7be66df	5 weeks ago	1.36GB
proyectofinal-messages	latest	08f14d5d03b1	5 weeks ago	1.36GB
proyectofinal-search	latest	1547fe9e22e4	5 weeks ago	1.26GB

☐ Ejecutar un contenedor con nuestra imagen

☐ Subir imagen a nuestra cuenta de dockerhub

7.1 Inicia sesión en Docker Hub

Primero, asegúrate de estar autenticado en Docker Hub desde tu terminal:
docker login

```
C:\Users\nicol>docker login
Authenticating with existing credentials...
Login Succeeded

C:\Users\nicol>
```

7.2 Etiquetar la imagen a subir con tu nombre de usuario de Docker Hub y el nombre de la imagen. Por ejemplo:

```
docker tag <nombre_imagen_local>
<tu_usuario_dockerhub>/<nombre_imagen>:<tag>
```

7.3 Subir la Imagen

Para subir la imagen etiquetada a Docker Hub, utiliza el comando docker push:

```
docker push <tu_usuario_dockerhub>/<nombre_imagen>:<tag>
```

7.4 Verificar la Subida

```
docker pull <tu_usuario_dockerhub>/<nombre_imagen>:<tag>
```

```
C:\Users\nicol>docker login
Authenticating with existing credentials...
Login Succeeded

C:\Users\nicol>docker tag mywebapi nicolasbergami/mywebapi:latest

C:\Users\nicol>docker push nicolasbergami/mywebapi:latest
The push refers to repository [docker.io/nicolasbergami/mywebapi]
3585453156b2: Pushed
5f70bf18a086: Mounted from library/golang
af51b0c14af2: Pushed
270f7fde987a: Pushed
4d29f6e29d10: Pushed
b4ec6db9c251: Pushed
ba941484fbe1: Pushed
123eef91533f: Pushed
latest: digest: sha256:96dc0a634c0364ee562d247174f9accf563dab960236bb468e2f1414572a3696 size: 1995

C:\Users\nicol>docker pull nicolasbergami/mywebapi:latest
latest: Pulling from nicolasbergami/mywebapi
Digest: sha256:96dc0a634c0364ee562d247174f9accf563dab960236bb468e2f1414572a3696
Status: Image is up to date for nicolasbergami/mywebapi:latest
```

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

- Procedemos entonces a parar y remover este contenedor:
docker kill myapi - docker rm myapi
- Vamos a volver a correrlo otra vez, pero publicando el puerto 80
docker run --name myapi -d -p 80:80 mywebapi

```
C:\Users\nicol>docker run --name myapi -d mywebapi
d2f9741cbe5d600de02bd41af1ea0a652b76c0028cceda129749031297530b72

C:\Users\nicol>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
d2f9741cbe5d   mywebapi  "dotnet SimpleWebAPI..." 6 seconds ago  Up 5 seconds  80/tcp, 443/tcp, 5254/tcp          myapi

C:\Users\nicol>docker kill myapi
myapi

C:\Users\nicol>- docker rm myapi
"- no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

C:\Users\nicol>docker rm myapi
myapi

C:\Users\nicol>docker run --name myapi -d -p 80:80 mywebapi
62360a54dc9d90cfe225110e855bb45d3d252867f5dd8c8e2b418e66f523a499

C:\Users\nicol>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
62360a54dc9d   mywebapi  "dotnet SimpleWebAPI..." 7 seconds ago  Up 6 seconds  443/tcp, 0.0.0.0:80->80/tcp, 5254/tcp  myapi

C:\Users\nicol>
```

Accedamos nuevamente a <http://localhost/WeatherForecast> y vemos que nos devuelve datos.

```
1  [
2      {
3          "date": "2024-08-28",
4          "temperatureC": 0,
5          "temperatureF": 32,
6          "summary": "Caloron"
7      },
8      {
9          "date": "2024-08-29",
10         "temperatureC": -13,
11         "temperatureF": 9,
12         "summary": "Balmy"
13     },
14     {
15         "date": "2024-08-30",
16         "temperatureC": 42,
17         "temperatureF": 107,
18         "summary": "Sweltering"
19     },
20     {
21         "date": "2024-08-31",
22         "temperatureC": 25,
23         "temperatureF": 76,
24         "summary": "Calido"
25     },
26     {
27         "date": "2024-09-01",
28         "temperatureC": 43,
29         "temperatureF": 109,
30         "summary": "Sweltering"
31     }
32 ]
```

9) Modificar Dockerfile para soportar bash

Modificamos dockerfile para que entre en bash sin ejecutar automaticamente la app

#ENTRYPOINT ["dotnet", "SimpleWebAPI.dll"]

CMD ["/bin/bash"]

```
C:\> FACULTAD > IngSw3 > IngSw3 > TP2 > SimpleWebAPI > Dockerfile > ...
1  #See https://aka.ms/containerfastmode to understand how Visual Studio uses this Dockerfile
2
3  FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
4  WORKDIR /app
5  EXPOSE 80
6  EXPOSE 443
7  EXPOSE 5254
8
9  FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
10 WORKDIR /src
11 COPY ["SimpleWebAPI/SimpleWebAPI.csproj", "SimpleWebAPI/"]
12 RUN dotnet restore "SimpleWebAPI/SimpleWebAPI.csproj"
13 COPY . .
14 WORKDIR "/src/SimpleWebAPI"
15 RUN dotnet build "SimpleWebAPI.csproj" -c Release -o /app/build
16
17 FROM build AS publish
18 RUN dotnet publish "SimpleWebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=false
19
20 FROM base AS final
21 WORKDIR /app
22 COPY --from=publish /app/publish .
23 #ENTRYPOINT ["dotnet", "SimpleWebAPI.dll"]
24 CMD ["/bin/bash"]
25 #CMD ["/bin/bash"]
26
27
28
```

- Rehacemos la imagen

docker build -t mywebapi .

```
C:\Users\nicol\OneDrive\Documentos\universidad\Ingsoft3\IngSoft3\TP2\SimpleWebAPI>docker build -t mywebapi .
2024/08/27 20:56:04 http2: server: error reading preface from client //./pipe/dockerDesktopLinuxEngine: file has already
been closed
[+] Building 6.4s (18/18) FINISHED                                                                                                     docker:desktop
linux
=> [internal] load build definition from Dockerfile
0.0s
=> => transferring dockerfile: 810B
0.0s
=> [internal] load metadata for mcr.microsoft.com/dotnet/sdk:7.0
1.4s
=> [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:7.0
1.4s
=> [internal] load .dockerignore
0.0s
=> => transferring context: 2B
0.0s
=> [build 1/7] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:d32bd65cf5843f413e81f5d917057c82da99737cb1637e905a1a4bc2e7e
0.0s
=> [internal] load build context
0.0s
=> => transferring context: 3.64kB
0.0s
=> [base 1/2] FROM mcr.microsoft.com/dotnet/aspnet:7.0@sha256:c7d9ee6cd01afe9aa80642e577c7cec9f5d87f88e5d70bd36fd610720
0.0s
=> CACHED [build 2/7] WORKDIR /src
```

- Corremos contenedor en modo interactivo exponiendo puerto

docker run -it --rm -p 80:80 mywebapi

```

root@e24d68eb76b6:/app#
root@e24d68eb76b6:/app# dotnet SimpleWebAPI.dll
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app

```

- Navegamos a <http://localhost/weatherforecast>
- Vemos que no se ejecuta automáticamente
- Ejecutamos app:

dotnet SimpleWebAPI.dll

```

[
  {
    "date": "2024-08-28",
    "temperatureC": 22,
    "temperatureF": 71,
    "summary": "Helado"
  },
  {
    "date": "2024-08-29",
    "temperatureC": 49,
    "temperatureF": 120,
    "summary": "Bracing"
  },
  {
    "date": "2024-08-30",
    "temperatureC": 3,
    "temperatureF": 37,
    "summary": "Scorching"
  },
  {
    "date": "2024-08-31",
    "temperatureC": 31,
    "temperatureF": 87,
    "summary": "Sweltering"
  },
  {
    "date": "2024-09-01",
    "temperatureC": 3,
    "temperatureF": 37,
    "summary": "Caloron"
  }
]

```

10) Montando volúmenes

- Hasta este punto los contenedores ejecutados no tenían contacto con el exterior, ellos corrían en su propio entorno hasta que terminaran su ejecución. Ahora veremos cómo montar un volumen dentro del contenedor para visualizar por ejemplo archivos del sistema huésped:
- Ejecutar el siguiente comando, cambiar myusuario por el usuario que corresponda. En Mac puede utilizarse /Users/miusuario/temp):
docker run -it --rm -p 80:80 -v /Users/miuser/temp:/var/temp


mywebapi

- Dentro del contenedor correr

ls -l /var/temp

touch /var/temp/hola.txt

- Verificar que el Archivo se ha creado en el directorio del guest y del host.

Nombre	Fecha de modificación	Tipo	Tamaño
 hola.txt	27/8/2024 18:17	Documento de tex...	0 KB

11)Utilizando una base de datos

- Levantar una base de datos PostgreSQL
 - Ejecutar sentencias utilizando esta instancia
 - Conectarse a la base utilizando alguna IDE (Dbeaver - <https://dbeaver.io/>, Azure DataStudio -<https://azure.microsoft.com/es-es/products/data-studio>, etc).
- Interactuar con los objetos creados.
- Explicar que se logro con el comando docker run y docker exec ejecutados en este ejercicio

```
.postgres:/var/lib/postgresql/data -p 5432:5432 -d postgres:9.4
Unable to find image 'postgres:9.4' locally
9.4: Pulling from library/postgres
619014d83c02: Pull complete
7ec0fe6664f6: Pull complete
9ca7ba8f7764: Pull complete
9e1155d037e2: Pull complete
febcb7f8870: Pull complete
8c78c79412b5: Pull complete
5a35744405c5: Pull complete
27717922e067: Pull complete
36f0c5255550: Pull complete
dbf0a396f422: Pull complete
ec4c06ea33e5: Pull complete
e8dd33eba6d1: Pull complete
51c81b3b2c20: Pull complete
2a03dd76f5d7: Pull complete
Digest: sha256:42a7a6a647a602efa9592edd1f56359800d079b93fa52c5d92244c58ac4a2ab9
Status: Downloaded newer image for postgres:9.4
docker: Error response from daemon: create $HOME/.postgres: "$HOME/.postgres" includes invalid characters for a local volume name, only "[a-zA-Z0-9][a-zA-Z0-9_.-]" are allowed. If you intended to pass a host directory, use absolute path.
See 'docker run --help'.
```

```
root@ff0e21124406:/# psql -h localhost -U postgres
psql (9.4.26)
Type "help" for help.

postgres=# create database test
postgres=# connect test
postgres=# create table tabla_a (mensaje varchar(50))
postgres=# insert into tabla_a (mensaje) values('Hola mundo!')
postgres=# select * from tabla_a
postgres=# ^\Quit
root@ff0e21124406:/# exit
```

Explicación de los Comandos Docker:

- **docker run:** Crea y ejecuta un nuevo contenedor con la configuración especificada. En este caso, estás creando un contenedor con PostgreSQL y configurando una contraseña para el usuario postgres.
- **docker exec:** Ejecuta un comando en un contenedor que ya está en funcionamiento. En este caso, docker exec te permite acceder a la interfaz de línea de comandos de PostgreSQL (psql) dentro del contenedor para que puedas interactuar con la base de datos.

12)Hacer el punto 11 con Microsoft SQL Server

- Armar un contenedor con SQL Server
- Crear BD, Tablas y ejecutar SELEC

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PO
RTS						
daf03c90d72e	mcr.microsoft.com/mssql/server:2019-latest	my-sqlserver	"/opt/mssql/bin/perm..."	19 seconds ago	Up 16 seconds	0.
0.0.0:1433->1433/tcp						
ff0e21124406	postgres:9.4	my-postgres	"docker-entrypoint.s..."	7 minutes ago	Up 7 minutes	0.
0.0.0:5432->5432/tcp						
755da6e80842	f709e71e4088		"dotnet SimpleWebAPI..."	3 hours ago	Up 3 hours	44
3/tcp, 5254/tcp, 0.0.0:8080->80/tcp		reverent_snyder				