Carolina Havenstein

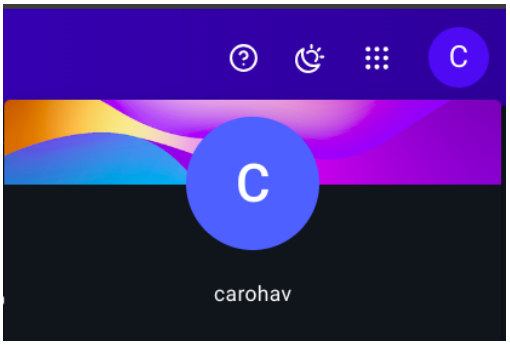
**TP 2 - Docker**

1- Instalar Docker Community Edition

```
caro@e14:~$ docker version
Client: Docker Engine - Community
 Version:           27.0.2
 API version:       1.46
 Go version:        go1.21.11
 Git commit:        912c1dd
 Built:             Wed Jun 26 18:47:16 2024
 OS/Arch:           linux/amd64
 Context:           default

Server: Docker Engine - Community
 Engine:
  Version:          27.0.2
  API version:      1.46 (minimum version 1.24)
  Go version:       go1.21.11
  Git commit:       e953d76
  Built:            Wed Jun 26 18:47:16 2024
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.18
  GitCommit:        ae71819c4f5e67bb4d5ae76a6b735f29cc25774e
 runc:
  Version:          1.7.18
  GitCommit:        v1.1.13-0-g58aa920
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
caro@e14:~$
```

2- Explorar DockerHub




3 - Obtener la imagen busybox

```
caro@e14:~$ docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
ec562eabd705: Pull complete
Digest: sha256:9ae97d36d26566ff84e8893c64a6dc4fe8ca6d1144bf5b87b2b85a32def253
c7
Status: Downloaded newer image for busybox:latest
docker.io/library/busybox:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scou
t quickview busybox
```

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

```
caro@e14:~$ docker images
REPOSITORY                     TAG            IMAGE ID       CREATED
 SIZE
proyectoarqsw2_frontend        latest         666ffcba5570   2 weeks ago
 1.27GB
proyectoarqsw2_arqsw2-hotels   latest         a7ee7309eedd   3 weeks ago
 1.36GB
proyectoarqsw2_arqsw2-uba      latest         09b42b577bdb   3 weeks ago
 1.23GB
proyectoarqsw2_arqsw2-search2  latest         0e26ca3bae1e   3 weeks ago
 1.26GB
proyectoarqsw2_arqsw2-db       latest         cfab6fc5d286   3 weeks ago
 586MB
mongo                          latest         a31b196b207d   6 weeks ago
 796MB
nginx                          latest         fffffc90d343   7 weeks ago
 188MB
solr                           latest         2a5ae62edb49   2 months ago
 581MB
rabbitmq                       3-management   64c5d4475635   5 months ago
 251MB
arqsw1-frontend                latest         980d7501498a   8 months ago
 1.18GB
arqsw1v2_frontend              latest         980d7501498a   8 months ago
 1.18GB
arqsw1-backend                 latest         0070edb8c5dd   8 months ago
 1.19GB
arqsw1v2_backend               latest         0070edb8c5dd   8 months ago
 1.19GB
arqsw1-db                      latest         f114d595fe09   8 months ago
 596MB
arqsw1v2_arqsw1-db             latest         f114d595fe09   8 months ago
 596MB
busybox                        latest         65ad0d468eb1   15 months ago
 4.26MB
memcached                      1.6.16         fdff4547b1b7   24 months ago
 89.2MB
```

4 - Ejecutando contenedores

Ejecutar un contenedor utilizando el comando run de docker:

```
caro@e14:~$ docker run busybox
```

No se obtuvo ningún resultado porque al no especificar ningún comando adicional, el contenedor se ejecuta y termina inmediatamente.
Especificamos algún comando a correr dentro del contenedor, ejecutar por ejemplo:

```
caro@e14:~$ docker run busybox echo "Hola Mundo"
Hola Mundo
```

Ver los contenedores ejecutados con el comando “docker ps”
Ver todos los contenedores con el comando “docker ps -a”

```
caro@e14:~$ docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED       STATUS
     PORTS                                       NAMES
ae57f1ff0102   memcached:1.6.16   "docker-entrypoint.s…"   2 weeks ago   Up 8 da
ys   0.0.0.0:11211->11211/tcp, :::11211->11211/tcp   memcached
caro@e14:~$ docker ps -a
CONTAINER ID   IMAGE                          COMMAND                  CREATED
             STATUS                         PORTS
             NAMES
48eeba9541c4   busybox                        "echo 'Hola Mundo'"      About a
minute ago   Exited (0) About a minute ago
             wizardly_goldstine
ead1ba07da79   busybox                        "sh"                     2 minute
s ago        Exited (0) 2 minutes ago
             reverent_borg
```

5 - Ejecutar en modo interactivo

```
caro@e14:~$ docker run -it busybox sh
/ # ls
bin    dev    etc    home   lib    lib64  proc   root   sys    tmp    usr    var
/ #
/ #
/ # ps
PID   USER     TIME  COMMAND
    1 root      0:00 sh
    8 root      0:00 ps
/ # uptime
 17:49:14 up 8 days,  4:10,  0 users,  load average: 0.35, 0.86, 0.92
/ # ls -l
total 40
drwxr-xr-x    2 root     root         12288 May 18  2023 bin
drwxr-xr-x    5 root     root           360 Aug 13 17:47 dev
drwxr-xr-x    1 root     root          4096 Aug 13 17:47 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  405 root     root             0 Aug 13 17:47 proc
drwx------    1 root     root          4096 Aug 13 17:47 root
dr-xr-xr-x   13 root     root             0 Aug 13 17:47 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
```

```
/ # ls
bin    etc    lib    proc   sys    usr
dev    home   lib64  root   tmp    var
/ # mkdir nuevodir
/ # ls
bin       etc       lib       nuevodir  root      tmp       var
dev       home      lib64     proc      sys       usr
/ # cd nuevodir
/nuevodir # exit
caro@e14:~$
```

```
caro@e14:~$ docker run -it busybox sh
/ # free
              total        used        free      shared  buff/cache   available
Mem:        7830024     4650228     1025068      768076     2154728     1819472
Swap:      12024308     2938880     9085428
/ #
```

6 - Borrando contenedores terminados

Obtener la lista de contenedores

```
caro@e14:~$ docker ps -a
CONTAINER ID   IMAGE                         COMMAND                  CREATED
         STATUS                     PORTS
     NAMES
b432dff167fe   busybox                       "sh"                     30 secon
ds ago   Exited (0) 18 seconds ago
     loving_stonebraker
55734ce31bb2   busybox                       "sh"                     6 minute
s ago    Exited (0) 3 minutes ago
     eager_haslett
48eeba9541c4   busybox                       "echo 'Hola Mundo'"      11 minut
es ago   Exited (0) 11 minutes ago
     wizardly_goldstine
ead1ba07da79   busybox                       "sh"                     12 minut
es ago   Exited (0) 12 minutes ago
     reverent_borg
```

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

```
caro@e14:~$ docker rm reverent_borg
reverent_borg
caro@e14:~$ docker ps -a
CONTAINER ID   IMAGE                         COMMAND                  CREATE
D              STATUS                        PORTS
                 NAMES
b432dff167fe   busybox                       "sh"                     About
a minute ago   Exited (0) About a minute ago
                 loving_stonebraker
55734ce31bb2   busybox                       "sh"                      7 minu
tes ago        Exited (0) 4 minutes ago
                 eager_haslett
48eeba9541c4   busybox                       "echo 'Hola Mundo'"      12 min
utes ago       Exited (0) 12 minutes ago
                 wizardly_goldstine
8b01f884c029   proyectoarqsw2_frontend       "docker-entrypoint.s…"   2 week
s ago          Exited (1) 2 weeks ago
                 proyectoarqsw2-frontend-1
```

Borrar todos los contenedores que no estén corriendo:

```
caro@e14:~$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
b432dff167fe1f7635fcf0a4c6faf8c80be808d573673541f0f7f90c7d789312
55734ce31bb233e63331bd42ab99d8c88a74408ea0a25f08a0dbdb82e82a296e
48eeba9541c4178735d98671a5f25174ad70ec305bdbabf8fb79a27746279868
8b01f884c0291b941038cfe74bba806238019518fb199d4e6d17d61553ca6b59
91166a04e565872891cd4f527ce0dc81115709075983985f5f2e03384335a964
e964c125cf04b0a0c55a5c27d446996f48e9f6631d03f44b26585ce4ae7d2cd3
e1cd4e3179e078edaa36045e6b989b9a3d189e62448922795d0799e00f058778
0e260bb4a06d738ff90b542b7e4260ebd624aeafae1706126cadd1ca4f749e6e
bf8b596a535936facf5ba7282c076ce8c7431bdf573138dfc6963483719823cd
11332456eb1ffb4364e8e58b39b1d524d1498dab661f62632a4f51771f47a13e
84482021b35c2aa124c324ada4de4283940cc563558f52ee4fa37ff2ae25f7de
f8a4d68ad3b350cebe5f8dfc53f644d054cd8f0f12d99d568941edeb9b38563b
7430ccfc73d78c1cdc3de782b7f239534da23a934f36d212a93b56646b912b13
d99aceb2e7d77dfbbfd5dba287c9ba237a35f4c97b92a83f862d320b1162c819

Total reclaimed space: 3.604MB
```

7 - Construir una imagen

Leer https://docs.docker.com/engine/reference/builder/

Describir las instrucciones:

FROM: inicializa una nueva etapa de build y setea una imagen base para instrucciones posteriores.

RUN: ejecuta cualquier comando para crear una nueva layer sobre la imagen actual. La layer agregada se usa en el siguiente paso del Dockerfile.

ADD: copia archivos nuevos o directorios de <src> y los agrega al sistema de archivos de la imagen en el path <dest>. Archivos y directorios pueden ser copiados desde el “build context”, un url remoto o un repositorio de Git.

COPY: copia archivos nuevos o directorios de <src> y los agrega al sistema de archivos de la imagen en el path <dest>. Archivos y directorios pueden ser copiados desde el “build context”, “build stage”, “named context” o una imagen.

EXPOSE: le informa a docker que el contenedor al ejecutarse escucha en el puerto de red especificado. Permite especificar el protocolo a utilizar (TCP o UDP), teniendo como default el protocolo TCP.

CMD: setea el comando para ser ejecutado al correr el contenedor desde una imagen. Solo puede haber una instrucción CMD en un Dockerfile, si hay más de una, solo la última hace efecto.

ENTRYPOINT: permite configurar un contenedor que va a correr como un ejecutable.

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

Clonar repositorio y crear imagen etiquetándola con un nombre. El punto final le indica a Docker que use el dir actual:

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker build -t mywebapi .
[+] Building 40.4s (18/18) FINISHED                          docker:default
 => [internal] load build definition from Dockerfile                    0.0s
 => => transferring dockerfile: 810B                                    0.0s
 => [internal] load metadata for mcr.microsoft.com/dotnet/sdk:7.0       2.2s
 => [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:7.0    1.2s
 => [internal] load .dockerignore                                       0.0s
 => => transferring context: 2B                                         0.0s
 => [build 1/7] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:d32bd65c  26.2s
 => => resolve mcr.microsoft.com/dotnet/sdk:7.0@sha256:d32bd65cf5843f4  0.0s
 => => sha256:5e263829ce76bc29ff631384bd5e93356d2 180.57MB / 180.57MB  24.0s
 => => sha256:c549dbd140e5a09bcc7e4b1da89ed4b96a0340 13.99MB / 13.99MB  6.9s
 => => sha256:a291948a5e5ceb50e3eaa75cb1e6377660146b38 2.01kB / 2.01kB  0.0s
 => => sha256:ea4f5eec20952a885a89566fc35cf3295b322837 5.29kB / 5.29kB  0.0s
 => => sha256:22689bb63f95af71c8c0079742ff0afa570fa6 25.37MB / 25.37MB  2.6s
 => => sha256:d32bd65cf5843f413e81f5d917057c82da99737c 1.79kB / 1.79kB  0.0s
 => => extracting sha256:22689bb63f95af71c8c0079742ff0afa570fa6cfc30ec  0.4s
 => => extracting sha256:5e263829ce76bc29ff631384bd5e93356d28625461ca5  1.9s
 => => extracting sha256:c549dbd140e5a09bcc7e4b1da89ed4b96a0340a730303  0.2s
 => [internal] load build context                                       0.0s
 => => transferring context: 3.89kB                                     0.0s
 => [base 1/2] FROM mcr.microsoft.com/dotnet/aspnet:7.0@sha256:c7d9ee6  0.0s
 => CACHED [base 2/2] WORKDIR /app                                      0.0s
 => CACHED [final 1/2] WORKDIR /app                                     0.0s
 => [build 2/7] WORKDIR /src                                            0.1s
 => [build 3/7] COPY [SimpleWebAPI/SimpleWebAPI.csproj, SimpleWebAPI/]  0.1s
 => [build 4/7] RUN dotnet restore "SimpleWebAPI/SimpleWebAPI.csproj"   4.9s
 => [build 5/7] COPY . .                                                0.1s
 => [build 6/7] WORKDIR /src/SimpleWebAPI                               0.0s
 => [build 7/7] RUN dotnet build "SimpleWebAPI.csproj" -c Release -o /  3.0s
 => [publish 1/1] RUN dotnet publish "SimpleWebAPI.csproj" -c Release   3.6s
 => [final 2/2] COPY --from=publish /app/publish .                      0.1s
 => exporting to image                                                  0.0s
 => => exporting layers                                                 0.0s
 => => writing image sha256:85936f9306a407da5fb5438e2045397a000c959b45  0.0s
 => => naming to docker.io/library/mywebapi                             0.0s

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

Revisar Dockerfile y explicar cada línea

**Primera etapa: base**

**FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base**

Usa la imagen oficial de .NET 7.0 (ASP.NET Core) como la base de esta etapa. Esta imagen contiene el entorno de tiempo de ejecución (runtime) necesario para ejecutar aplicaciones ASP.NET Core. AS base le da un nombre a esta etapa del Dockerfile, lo cual permite reutilizarla más adelante.

**WORKDIR /app**

Establece el directorio de trabajo dentro del contenedor en /app. Todos los comandos posteriores se ejecutarán dentro de este directorio.

**EXPOSE 80**

**EXPOSE 443**

**EXPOSE 5254**

Informa a Docker que el contenedor escuchará en los puertos 80, 443, y 5254. Estos son los puertos que se esperan utilizar para HTTP, HTTPS y otro puerto específico para tu aplicación.

**Segunda etapa: build**

**FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build**

Usa la imagen oficial del SDK de .NET 7.0, que incluye herramientas de desarrollo como el compilador, para compilar la aplicación. AS build le da un nombre a esta etapa.

**WORKDIR /src**

Establece el directorio de trabajo en /src, donde se copiarán los archivos del proyecto.

**COPY ["SimpleWebAPI/SimpleWebAPI.csproj", "SimpleWebAPI/"]**

Copia el archivo del proyecto .csproj al contenedor en el directorio SimpleWebAPI. Esto permite restaurar las dependencias sin copiar todo el código fuente.

**RUN dotnet restore "SimpleWebAPI/SimpleWebAPI.csproj"**

Restaura las dependencias del proyecto especificado en el archivo .csproj, descargándolas de NuGet.

**COPY . .**

Copia todo el código fuente y otros archivos necesarios al contenedor.

**WORKDIR "/src/SimpleWebAPI"**

Cambia el directorio de trabajo al directorio del proyecto dentro del contenedor.

**RUN dotnet build "SimpleWebAPI.csproj" -c Release -o /app/build**

Compila el proyecto en modo Release y coloca los archivos compilados en el directorio /app/build dentro del contenedor.

**Tercera etapa: Publish**

**FROM build AS publish**

Usa la imagen de la etapa build como punto de partida.
AS publish le da un nombre a esta nueva etapa.

**RUN dotnet publish "SimpleWebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=false**

Publica la aplicación, creando una versión optimizada para producción. La opción /p:UseAppHost=false se utiliza para evitar incluir un ejecutable nativo, lo cual es útil en algunos entornos de contenedores. Los archivos publicados se colocan en /app/publish.

**Cuarta etapa: Final**

**FROM base AS final**

Usa la imagen base (la primera etapa) como punto de partida para la etapa final del contenedor.AS final le da un nombre a esta etapa.

**WORKDIR /app**

Establece el directorio de trabajo en /app, que es donde se copiarán los archivos publicados.

**COPY --from=publish /app/publish .**

Copia los archivos publicados desde la etapa publish al directorio /app en la imagen final.

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

Define el punto de entrada para el contenedor. Cuando el contenedor se inicie, ejecutará dotnet SimpleWebAPI.dll, que arrancará la aplicación.

**#CMD ["/bin/bash"]**

Este comando está comentado (con #), lo que significa que no se ejecuta. Si se activara, indicaría que se quiere iniciar una sesión de bash en lugar de ejecutar la aplicación.

Ver imágenes disponibles

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker images
REPOSITORY                    TAG            IMAGE ID       CREATED
 SIZE
mywebapi                      latest         85936f9306a4   6 days ago
 216MB
```

Ejecutar un contenedor con nuestra imagen

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker run mywebapi
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
```

Subir imagen a nuestra cuenta de dockerhub

- inicia sesión en Docker Hub

Para poder iniciar sesión también fue necesario

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ gpg --gen-key
```

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ gpg --list-keys
```

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ pass init 6D6...
```

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker login
Log in with your Docker ID or email address to push and pull images from Dock
er Hub. If you don't have a Docker ID, head over to https://hub.docker.com/ t
o create one.
You can log in with your password or a Personal Access Token (PAT). Using a l
imited-scope PAT grants better security and is required for organizations usi
ng SSO. Learn more at https://docs.docker.com/go/access-tokens/

Username: carohav
Password:
Login Succeeded
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$
```

Etiquetar la imagen a subir con tu nombre de usuario de Docker Hub y el nombre de la imagen

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker tag mywebapi carohav/mywe
bapi:latest
```

Subir la Imagen etiquetada a Docker Hub

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker push carohav/mywebapi:lat
est
The push refers to repository [docker.io/carohav/mywebapi]
234a9667dab2: Pushed
5f70bf18a086: Mounted from library/golang
5fb319cfbc84: Pushed
270f7fde987a: Pushed
4d29f6e29d10: Pushed
b4ec6db9c251: Pushed
ba941484fbe1: Pushed
123eef91533f: Pushed
latest: digest: sha256:4ca8f15efa3b5567974dcea64d050d95458013f4a1ba2abe1d9fea
63e490bd03 size: 1995
```

Verificar la subida

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker pull carohav/mywebapi:lat
est
latest: Pulling from carohav/mywebapi
Digest: sha256:4ca8f15efa3b5567974dcea64d050d95458013f4a1ba2abe1d9fea63e490bd
03
Status: Image is up to date for carohav/mywebapi:latest
docker.io/carohav/mywebapi:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scou
t quickview carohav/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.

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker run --name myapi -d myweb
api
```

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker ps -a
CONTAINER ID   IMAGE              COMMAND                  CREATED       STAT
US                    PORTS                                          NAMES
463414bad468   mywebapi           "dotnet SimpleWebAPI…"   7 hours ago   Up 7
 hours                80/tcp, 443/tcp, 5254/tcp                      pricele
```

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.

**Not Found**

The requested URL was not found on this server.

*Apache/2.4.52 (Ubuntu) Server at localhost Port 80*

Paramos y removemos el contenedor

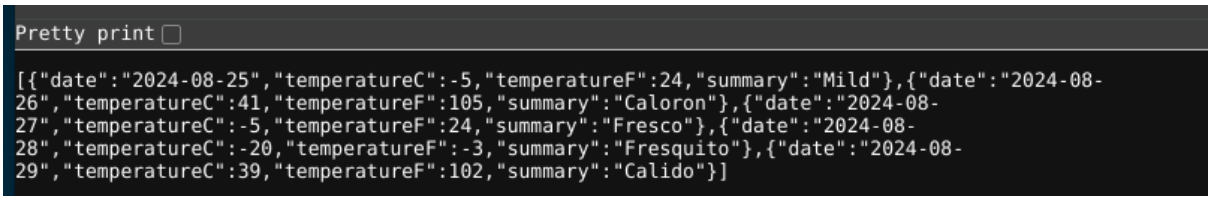
```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker kill myapi
```

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker rm myapi
myapi
```

Vamos a volver a correrlo otra vez, pero publicando el puerto 8080

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker run --name myapi -d -p 8080:80 mywebapi
cb1e140af5d074d18ff09bbf5ef3cecd51f5b0a1e9d50f4eb3eb733af5f93fc3
```

vamos a http://localhost:8080/WeatherForecast y ahora si devuelve datos




9 - Modificar Dockerfile para soportar bash

Modificamos dockerfile para que entre en bash sin ejecutar automáticamente la app.

```
#ENTRYPOINT ["dotnet", "SimpleWebAPI.dll"]
CMD ["/bin/bash"]
```

Rehacemos la imagen

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker build -t mywebapi .
[+] Building 5.8s (18/18) FINISHED                              docker:default
 => [internal] load build definition from Dockerfile                      0.0s
 => => transferring dockerfile: 812B                                      0.0s
 => [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:7.0      1.2s
 => [internal] load metadata for mcr.microsoft.com/dotnet/sdk:7.0         1.2s
 => [internal] load .dockerignore                                         0.0s
 => => transferring context: 2B                                           0.0s
 => [build 1/7] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:d32bd65cf    0.0s
 => [base 1/2] FROM mcr.microsoft.com/dotnet/aspnet:7.0@sha256:c7d9ee6    0.0s
 => [internal] load build context                                         0.0s
 => => transferring context: 4.67kB                                       0.0s
 => CACHED [build 2/7] WORKDIR /src                                       0.0s
 => CACHED [build 3/7] COPY [SimpleWebAPI/SimpleWebAPI.csproj, SimpleW    0.0s
 => CACHED [build 4/7] RUN dotnet restore "SimpleWebAPI/SimpleWebAPI.c    0.0s
 => [build 5/7] COPY . .                                                  0.1s
 => [build 6/7] WORKDIR /src/SimpleWebAPI                                 0.0s
 => [build 7/7] RUN dotnet build "SimpleWebAPI.csproj" -c Release -o /    3.0s
 => [publish 1/1] RUN dotnet publish "SimpleWebAPI.csproj" -c Release     1.5s
 => CACHED [base 2/2] WORKDIR /app                                        0.0s
 => CACHED [final 1/2] WORKDIR /app                                       0.0s
 => CACHED [final 2/2] COPY --from=publish /app/publish .                 0.0s
 => exporting to image                                                    0.0s
 => => exporting layers                                                   0.0s
 => => writing image sha256:2fd4aad2dd8e56b393e69af9e6fb2c5e476a83b76a    0.0s
 => => naming to docker.io/library/mywebapi                               0.0s

What's next:
    View a summary of image vulnerabilities and recommendations → docker scou
t quickview
```

Corremos contenedor en modo interactivo exponiendo puerto

```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker run -it --rm -p 8080:80 m
ywebapi
root@b567643d086b:/app#
```

vamos a http://localhost:8080/WeatherForecast y vemos que no se ejecuta automáticamente.

ejecutamos app

```
root@b567643d086b:/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
```

y vemos que funciona

```
[{"date":"2024-08-25","temperatureC":-12,"temperatureF":11,"summary":"Sweltering"},{"date":"2024-08-
26","temperatureC":50,"temperatureF":121,"summary":"Scorching"},{"date":"2024-08-
27","temperatureC":33,"temperatureF":91,"summary":"Sweltering"},{"date":"2024-08-
28","temperatureC":21,"temperatureF":69,"summary":"Helado"},{"date":"2024-08-
29","temperatureC":18,"temperatureF":64,"summary":"Mild"}]
```

salimos del contenedor

```
root@b567643d086b:/app# exit
exit
```

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:

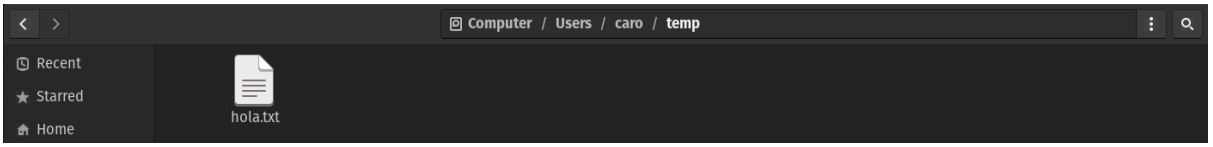
```
caro@e14:~/Documents/tp_ingsw3/SimpleWebAPI$ docker run -it --rm -p 8080:80 -
v /Users/caro/temp:/var/temp  mywebapi
```

```
root@07b86db57bb2:/app# ls -l /var
total 40
drwxr-xr-x 2 root root  4096 Jan 28  2024  backups
drwxr-xr-x 1 root root  4096 May 13 00:00  cache
drwxr-xr-x 1 root root  4096 May 13 00:00  lib
drwxrwsr-x 2 root staff 4096 Jan 28  2024  local
lrwxrwxrwx 1 root root     9 May 13 00:00  lock -> /run/lock
drwxr-xr-x 1 root root  4096 May 29 16:10  log
drwxrwsr-x 2 root mail  4096 May 13 00:00  mail
drwxr-xr-x 2 root root  4096 May 13 00:00  opt
lrwxrwxrwx 1 root root     4 May 13 00:00  run -> /run
drwxr-xr-x 2 root root  4096 May 13 00:00  spool
drwxr-xr-x 2 root root  4096 Aug 24 21:10 'temp'$'\302\240'
drwxrwxrwt 2 root root  4096 Jan 28  2024  tmp
```

```
root@07b86db57bb2:/app# ls -l /var/temp\302\240/
total 0
root@07b86db57bb2:/app# touch /var/temp\302\240/hola.txt
```

Verificar que el archivo se haya creado en el directorio del guest y del host

```
root@07b86db57bb2:/var# cd temp\302\240/
root@07b86db57bb2:/var/temp # ls
hola.txt
```




11 - Utilizando una base de datos

```
caro@e14:~$ mkdir $HOME/.postgres
```

```
caro@e14:~$ docker run --name my-postgres -e POSTGRES_PASSWORD=mysecretpasswo
rd -v $HOME/.postgres:/var/lib/postgresql/data -p 5432:5432 -d postgres:9.4
45d30f232c46e99a94bdbbd369cdfabb14d361ceb0bb9cc256e602e82d379315
```

```
caro@e14:~$ docker exec -it my-postgres /bin/bash
root@45d30f232c46:/# psql -h localhost -U postgres
psql (9.4.26)
Type "help" for help.
```

```
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privile
ges
-----------+----------+----------+------------+------------+-----------------
------
 postgres  | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres
     +
           |          |          |            |            | postgres=CTc/pos
tgres
 template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres
     +
           |          |          |            |            | postgres=CTc/pos
tgres
(3 rows)
```

```
postgres=# create database test;
CREATE DATABASE
postgres=# \connect test
You are now connected to database "test" as user "postgres".
test=# create table tabla_a (mensaje varchar(50));
CREATE TABLE
test=# insert into tabla_a (mensaje) values('Hola mundo!');
INSERT 0 1
test=# select * from tabla_a;
   mensaje
-------------
 Hola mundo!
(1 row)
```

```
test=# \q
root@45d30f232c46:/# exit
exit
```

Explicar que se logro con el comando docker run y docker exec ejecutados en este ejercicio.
docker run: para iniciar un contenedor Docker basado en la imagen de PostgreSQL.

docker exec: para ejecutar comandos dentro de un contenedor Docker que ya está en funcionamiento. Permite interactuar con un contenedor que ya está corriendo

12 - Hacer el punto 11 con Microsoft MySQL server
Armar un contenedor con SQL Server
Traigo la imagen

```
caro@e14:~$ docker pull mcr.microsoft.com/mssql/server:2022-latest
2022-latest: Pulling from mssql/server
e7945123d2a2: Pull complete


18a53d1b3bd7: Pull complete


d2a9a15297bf: Pull complete


Digest: sha256:c1aa8afe9b06eab64c9774a4802dcd032205d1be785b1fd51e1c0151e7586b
74
Status: Downloaded newer image for mcr.microsoft.com/mssql/server:2022-latest
mcr.microsoft.com/mssql/server:2022-latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scou
t quickview mcr.microsoft.com/mssql/server:2022-latest
```

Creo un contenedor
Crear BD, Tablas y ejecutar SELECT

```
caro@e14:~$ docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=Anilorac.9" -p 1500
0:1433 -d mcr.microsoft.com/mssql/server:2022-latest
760cfdc099c8b940edd7e1ee49461e6b3a814e490161bf786c88ebaeda8364a3
caro@e14:~$ docker ps
CONTAINER ID   IMAGE                                        COMMAND
        CREATED          STATUS          PORTS
          NAMES
760cfdc099c8   mcr.microsoft.com/mssql/server:2022-latest   "/opt/mssql/bin/p
erm…"   22 seconds ago   Up 21 seconds   0.0.0.0:15000->1433/tcp, :::15000->1
433/tcp   youthful_haibt
caro@e14:~$ docker exec -it youthful_haibt /opt/mssql-tools18/bin/sqlcmd -S l
ocalhost -U SA -P 'Anilorac.9' -C
1>
```

```
1> CREATE DATABASE ing;
2> GO
1> USE ing;
2> GO
Changed database context to 'ing'.
```

```
1> CREATE TABLE alumnos (id INT PRIMARY KEY, nombre NVARCHAR(50) NOT NULL);
2> GO
```

```
1> SELECT * FROM alumnos;
2> GO
id          nombre
----------- --------------------------------------------------

(0 rows affected)
```

```
1> INSERT INTO alumnos (id, nombre) VALUES (123, 'caro');
2> GO

(1 rows affected)
1> SELECT * FROM alumnos;
2> GO
id          nombre
----------- --------------------------------------------------
        123 caro

(1 rows affected)
```