



---

# PRODUCTO 1. PREPARACIÓN ENTORNO DE DESAROLLO

---

Devops y cloud computing



Adrián Juguera Aquilino  
Eric Moliner Pérez  
Hector Carrasco Bernad

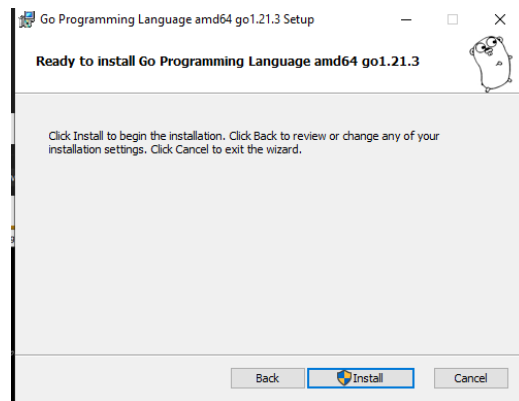
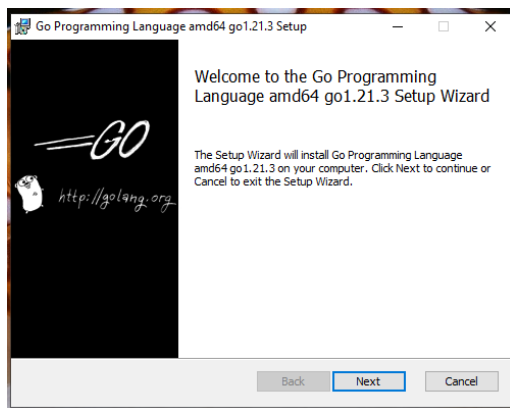
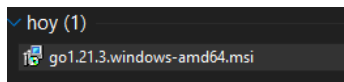
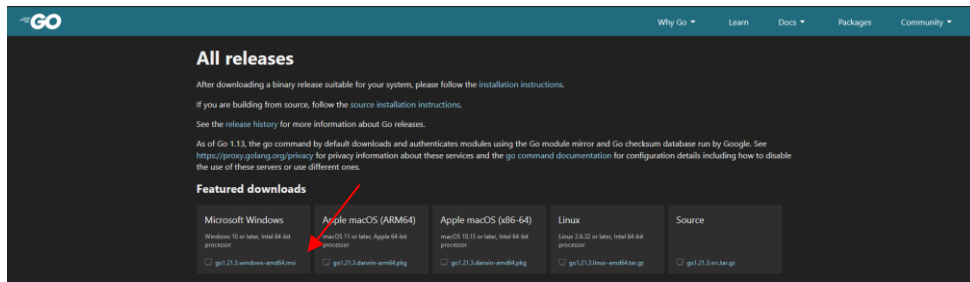
## Índice

Preparación de entorno desde Windows.....	2
<b>Paso 1: Instalar Golang:</b> .....	2
<b>Paso 2: Crear una web con Go:</b> .....	2
<b>Paso 3: Instalar Docker y crear fichero:</b> .....	4
Preparación del entorno desde Ubuntu.....	6
<b>Instalación de Golang</b> .....	6
<b>Creación de la web con Go</b> .....	6
<b>Construcción del dockerfile</b> .....	7
<b>Construir la imagen Docker</b> .....	7
<b>Comprobación del funcionamiento del servidor web</b> .....	8

## Preparación de entorno desde Windows

### Paso 1: Instalar Golang:

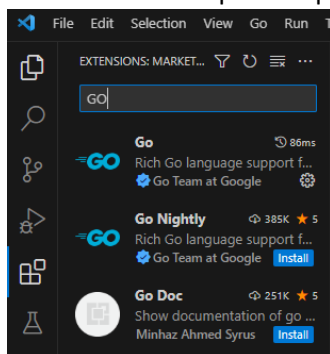
- Descargaremos e instalaremos en nuestro equipo Golang desde el sitio web oficial de Golang (<https://golang.org/dl/>).



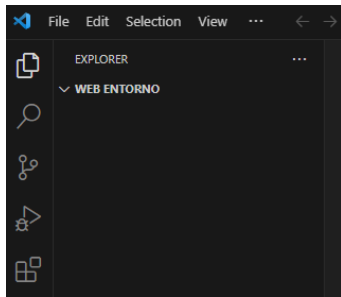
### Paso 2: Crear una web con Go:

- Necesitamos crear el archivo main.go, utilizaremos el programa Visual Studio Code por ejemplo.

Para ello, buscaremos la extensión oficial de Go, buscando mismamente go en los cuadraditos de la parte izquierda.



- Crearemos un proyecto dándole en Archivo, Open Folder para crear una carpeta que en nuestro caso se llamará web entorno



- Dentro de la carpeta crearemos el archivo main.go y creamos el siguiente programa:

```

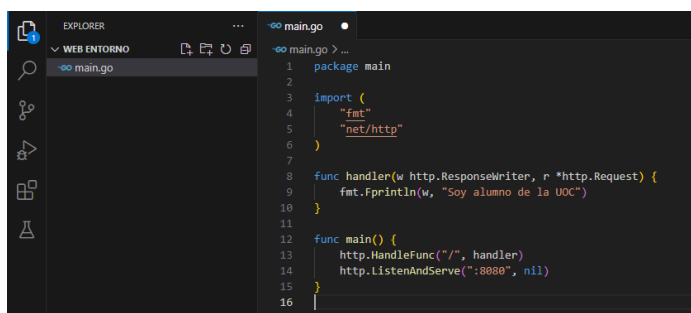
package main

import (
    "fmt"
    "net/http"
)

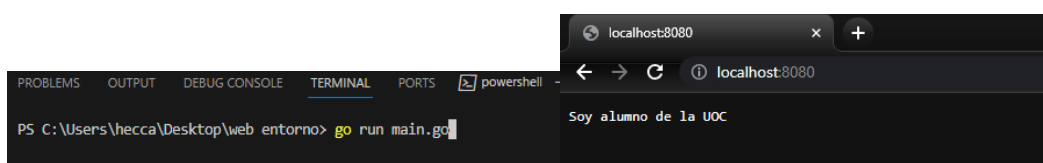
func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "Soy alumno de la UOC")
}

func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}

```

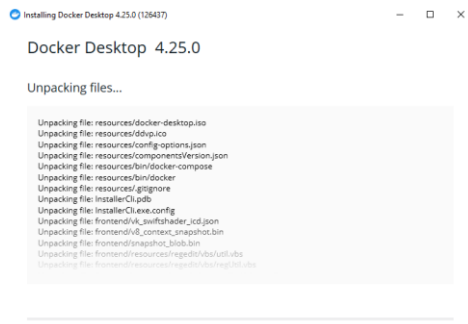


- Abriremos la terminal dentro de VS Code y ejecutaremos la aplicación go añadiendo el comando `go run main.go`:
- Esto iniciará la aplicación web en el puerto 8080. Podemos acceder a ella en un navegador visitando `http://localhost:8080`.

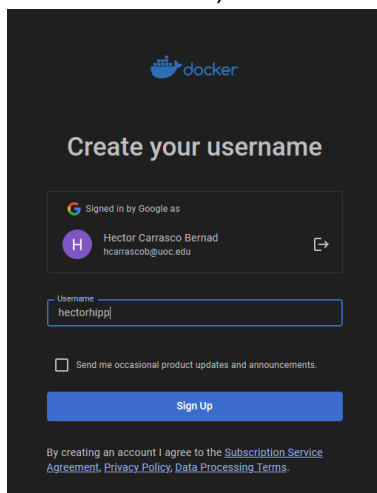


### Paso 3: Instalar Docker y crear fichero:

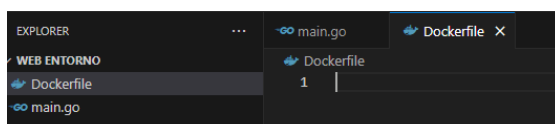
- Descargaremos e instalaremos Docker desde el sitio web oficial de Docker <https://www.docker.com/get-started>.



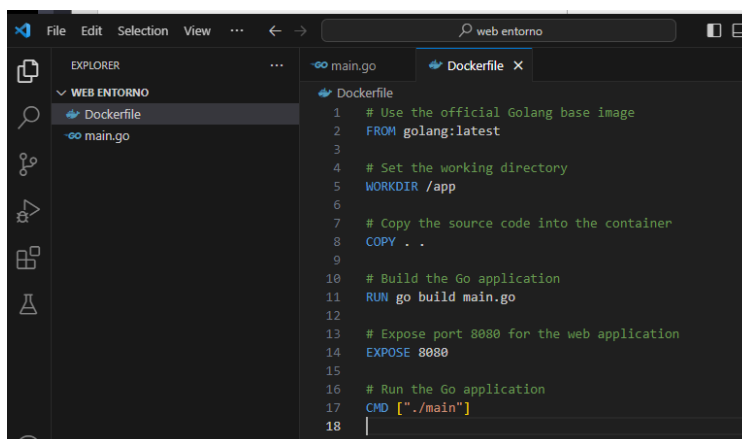
- Una vez instalado, creamos una cuenta, por ejemplo esta:



- Dentro del mismo directorio en VS Code, crearemos un nuevo archivo nombrándolo Dockerfile sin extensión.



- Agregaremos el contenido del Dockerfile y guardaremos el archivo.



- Abrimos un terminal dentro del mismo directorio del Dockerfile y ejecutamos el siguiente comando: `docker build -t mi-aplicacion-go .`  
Esto construirá la imagen Docker llamada mi aplicación go.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hecca\Desktop\web entorno> docker build -t mi-aplicacion-go .
[+] Building 15.3s (5/9)                                docker:default
=> [1/4] FROM docker.io/library/golang:latest@sha256:b113af1e 13.4s
=> => sha256:16418b73e8cedef61f8a3ecef6cab7 67.00MB / 67.00MB 7.6s
=> => extracting sha256:8457fd5474e70835e4482983a566235d892d5 2.5s
=> => sha256:d0b897dfff2699368980e221be9948aa5e30 155B / 155B 4.8s
=> => extracting sha256:13baa2029dde87a21b87127168a0fb50a007c0 0.7s
=> => extracting sha256:325c5bf4c2f26c11380501bec4b6eef8a3ea35 2.3s
=> => extracting sha256:c185020e1367a154f8bf1d26293b7c6fac0624 2.1s
=> => extracting sha256:16418b73e8cedef61f8a3ecef6cab70f5f7ae 1.9s
=> [internal] load build context                        0.0s

```

- Después de que la imagen sea creada con éxito, tendremos que ejecutar un contenedor a partir de la imagen utilizando el comando Docker run.  
`docker run -p 8080:8080 mi-aplicacion-go.`

```

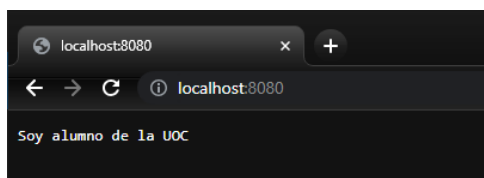
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

=> [4/4] RUN go build main.go 5.8s
=> exporting to image 0.3s
=> => exporting layers 0.3s
=> => writing image sha256:03e28f090ecb7c05372cae6a250d434f37 0.0s
=> => naming to docker.io/library/mi-aplicacion-go 0.0s

What's Next?
View a summary of image vulnerabilities and recommendations + docker
scout quickview
PS C:\Users\hecca\Desktop\web entorno> docker run -p 8080:8080 mi-apli
cacion-go

```

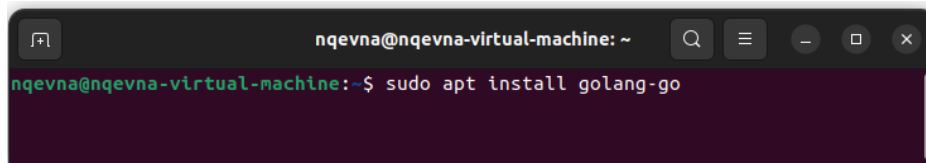
- Comprobamos que la aplicación Docker esta funcionando poniendo en nuestro navegador <http://localhost:8080>



Search						
<input type="checkbox"/>	Name	Tag	Status	Created	Size	Actions
<input type="checkbox"/>	mi-aplicacion-go	latest	In use	5 minutes ago	884.88 MB	<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>
	03e28f090ecc					

## Preparación del entorno desde Ubuntu

### Instalación de Golang



```
nqevna@nqevna-virtual-machine: ~  
nqevna@nqevna-virtual-machine:~$ sudo apt install golang-go
```

### Creación de la web con Go

```
1 package main  
2  
3 import (  
4     "fmt"  
5     "net/http"  
6 )  
7  
8 func handler(w http.ResponseWriter, r *http.Request) {  
9     switch r.Method {  
10        case http.MethodGet:  
11            fmt.Fprintf(w, "Soy alumno de la UOC - Método GET")  
12        case http.MethodPost:  
13            fmt.Fprintf(w, "Soy alumno de la UOC - Método POST")  
14        default:  
15            http.Error(w, "Método no soportado", http.StatusMethodNotAllowed)  
16        }  
17    }  
18  
19 func main() {  
20     http.HandleFunc("/", handler)  
21     http.ListenAndServe(":8080", nil)  
22 }  
23
```

- package main: Este es el punto de entrada de la aplicación. En Go, la función main() dentro del package main es el primer punto de ejecución cuando se arranca el programa.
- import: Aquí importamos dos paquetes que necesitaremos.
  - fmt: Este paquete nos proporciona funciones para formatear texto.
  - net/http: Este paquete nos permite construir servidores HTTP.
- func handler: Definimos una función llamada handler que será llamada cada vez que haya una petición HTTP a nuestro servidor.
  - w http.ResponseWriter: Este es el objeto que permite enviar respuestas al cliente.
  - r \*http.Request: Este objeto contiene información sobre la petición HTTP recibida, como el método (GET, POST, etc.), las cabeceras, etc.
- switch r.Method: Evaluamos el método de la petición HTTP (GET, POST, etc.) y cambiamos el mensaje para reflejar cómo se ha realizado.
- func main(): Esta es la función principal que se ejecuta al iniciar el programa.

http.HandleFunc("/", handler): Esto registra la función handler para ser llamada cada vez que se accede a la ruta / en nuestro servidor.

http.ListenAndServe(":8080", nil): Esto arranca el servidor en el puerto 8080. Las peticiones serán manejadas por las funciones que hayamos registrado con HandleFunc.

## Construcción del dockerfile

Para la construcción de la imagen se han utilizado 2 etapas:

- Etapa de construcción: En esta etapa, a la que se asigna el alias build-env, se procede a:
  - Importar la imagen golang:alpine, que es una imagen basada en Linux Alpine (un SO muy ligero) con la capacidad de compilar Go.
  - Se establece el directorio de trabajo en la ruta /app del contenedor.
  - Se copian los archivos existentes en la misma carpeta que el Dockerfile en el directorio de trabajo del contenedor.
  - Se ejecuta el comando para compilar el ejecutable.
- Etapa final: Una vez compilado el ejecutable, se procede a:
  - Importar la imagen alpine, esta vez sin Golang integrado para que la imagen final pese menos.
  - Establecer el directorio de trabajo en la ruta /app del contenedor.
  - Copiar el archivo main desde la anterior etapa a la nueva en el directorio actual.
  - La instrucción EXPOSE en un Dockerfile se utiliza para informar a Docker que la aplicación dentro del contenedor escuchará en uno o varios puertos de red específicos en tiempo de ejecución. Es una forma de documentar qué puertos deberían ser expuestos o mapeados para permitir la comunicación externa con el contenedor, pero no estarán automáticamente disponibles para la máquina host. Es necesario especificar explícitamente los puertos que se quieren mapear al iniciar el contenedor.
  - Especificar el comando que se debe ejecutar cuando se inicia el contenedor. En este caso, estamos ejecutando nuestro binario de aplicación, main.

## Construir la imagen Docker

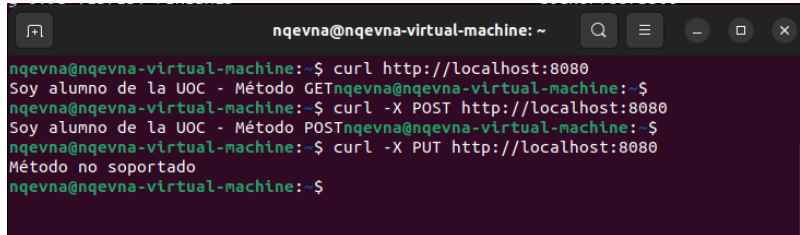
Desde la terminal, nos desplazamos al directorio donde está el Dockerfile y procedemos a buildearlo y a iniciarlo, en este caso etiquetándolo como devops al crear la build y mapeando los puertos 8080 para acceder al servidor web que se está ejecutando en el contenedor.

```
nqevna@nqevna-virtual-machine:~/Documents/DevOps$ sudo docker build -t devops .
[sudo] password for nqevna:
[+] Building 0.9s (13/13) FINISHED                                docker:default
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                     0.0s
=> [internal] load build definition from Dockerfile               0.0s
=> => transferring dockerfile: 390B                                0.0s
=> [internal] load metadata for docker.io/library/golang:alpine  0.9s
=> [internal] load metadata for docker.io/library/alpine:latest  0.0s
=> [build-env 1/4] FROM docker.io/library/golang:alpine@sha256:926f7f7e1 0.0s
=> [stage-1 1/3] FROM docker.io/library/alpine@sha256:eece025e432126ce23 0.0s
=> [internal] load build context                                  0.0s
=> => transferring context: 84B                                     0.0s
=> CACHED [stage-1 2/3] WORKDIR /app                              0.0s
=> CACHED [build-env 2/4] WORKDIR /app                            0.0s
=> CACHED [build-env 3/4] COPY . .                                0.0s
=> CACHED [build-env 4/4] RUN go build -o main .                  0.0s
=> CACHED [stage-1 3/3] COPY --from=build-env /app/main .        0.0s
=> exporting to image                                             0.0s
=> => exporting layers                                             0.0s
=> writing image sha256:d09913d82d12b9d9d1d2101f21062fef610e0aa68aefb 0.0s
=> => naming to docker.io/library/devops                          0.0s
nqevna@nqevna-virtual-machine:~/Documents/DevOps$ docker run -p 8080:8080 devops
docker: permission denied while trying to connect to the Docker daemon socket at unix:/
ock: connect: permission denied.
See 'docker run --help'.
nqevna@nqevna-virtual-machine:~/Documents/DevOps$ sudo docker run -p 8080:8080 devops
```



## Comprobación del funcionamiento del servidor web

Una vez corriendo el contenedor, podemos mediante la herramienta cURL hacer distintas peticiones para asegurarnos de que el servidor funciona correctamente:

A terminal window titled 'nqevna@nqevna-virtual-machine: ~' with standard window controls. It shows a series of curl commands and their outputs. The first command is 'curl http://localhost:8080', which returns 'Soy alumno de la UOC - Método GET'. The second is 'curl -X POST http://localhost:8080', returning 'Soy alumno de la UOC - Método POST'. The third is 'curl -X PUT http://localhost:8080', returning 'Método no soportado'.

```
nqevna@nqevna-virtual-machine: ~  
nqevna@nqevna-virtual-machine:~$ curl http://localhost:8080  
Soy alumno de la UOC - Método GETnqevna@nqevna-virtual-machine:~$  
nqevna@nqevna-virtual-machine:~$ curl -X POST http://localhost:8080  
Soy alumno de la UOC - Método POSTnqevna@nqevna-virtual-machine:~$  
nqevna@nqevna-virtual-machine:~$ curl -X PUT http://localhost:8080  
Método no soportado  
nqevna@nqevna-virtual-machine:~$
```