

Producto 1. Preparación entorno de desarrollo

Preparación en Windows con Docker UI g




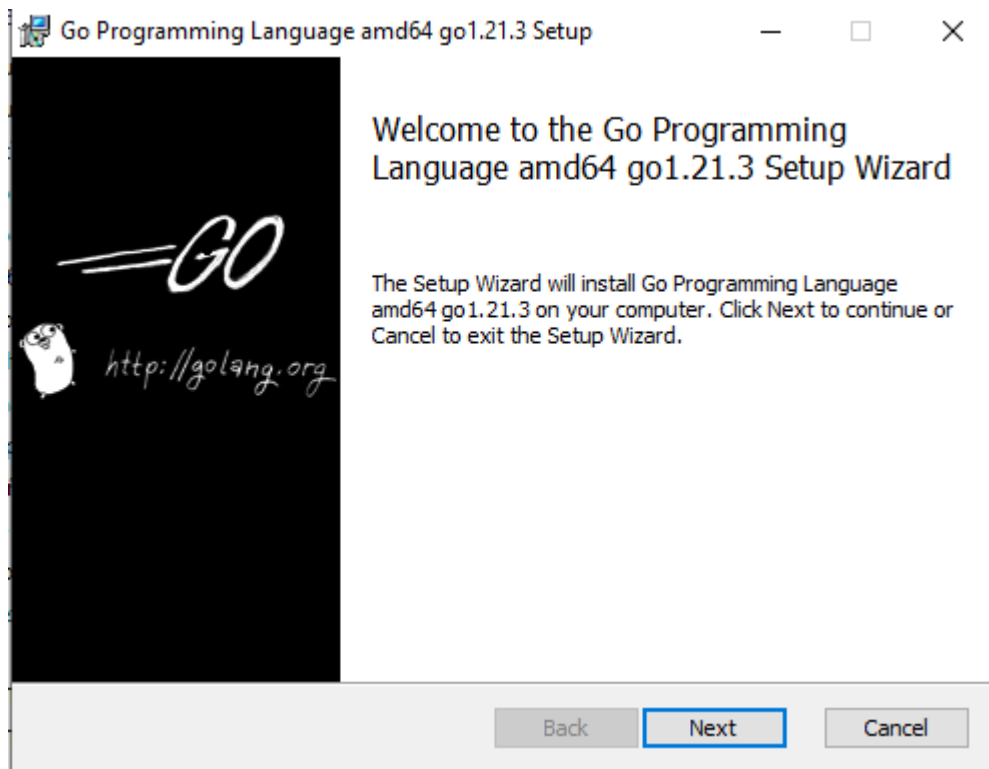


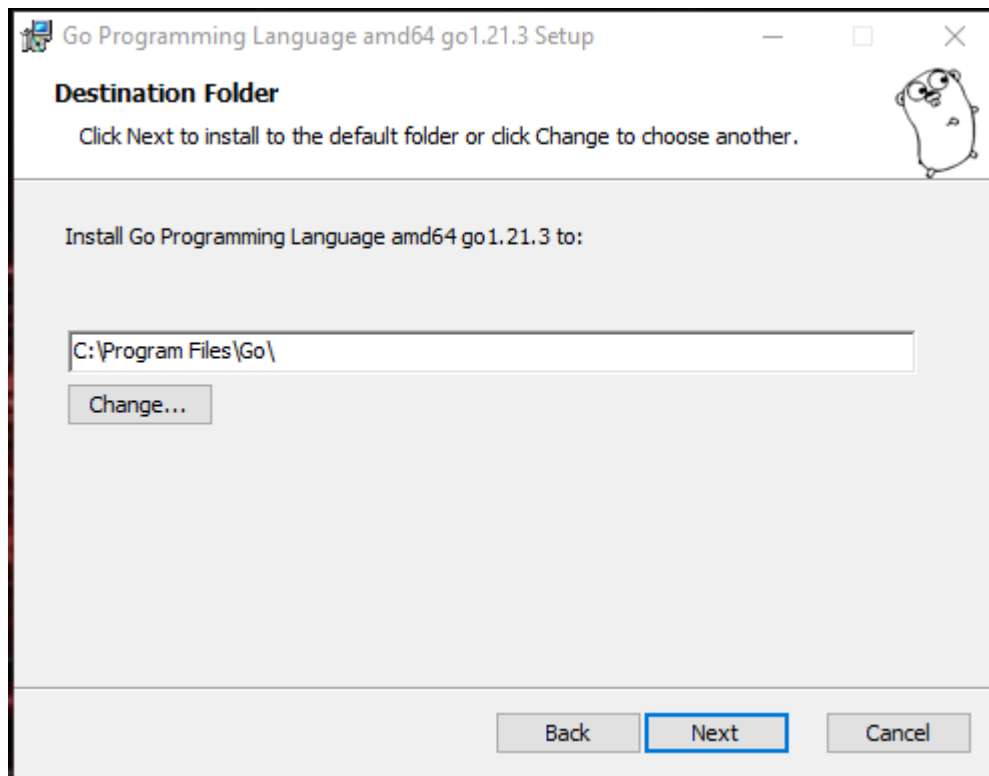
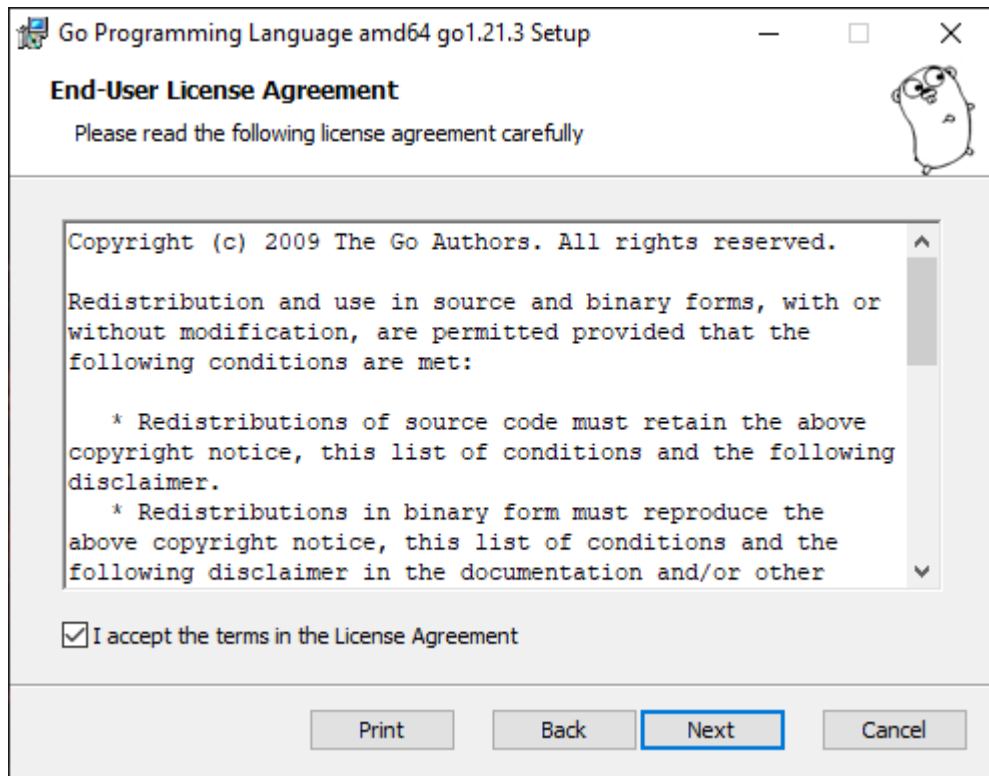
1. Instalar Golang

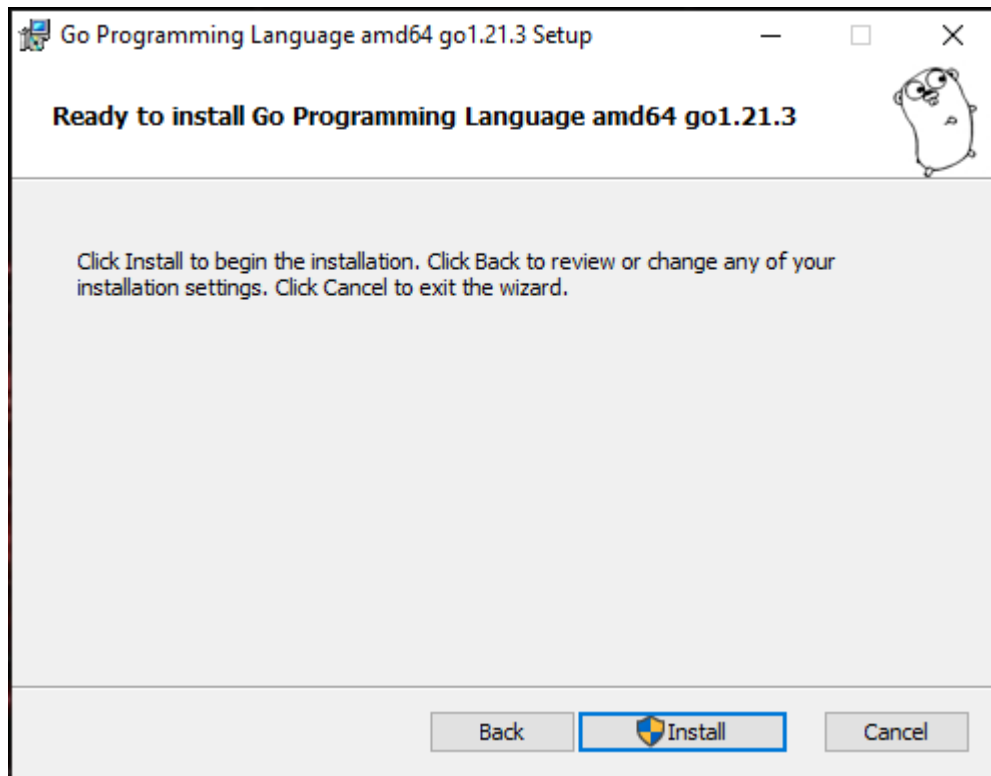
Primero para instalar golang, accederemos a su web oficial y nos descargamos el archivo ejecutable, en nuestro caso vamos a instalarlo en entorno Windows.

<https://go.dev/dl/>

 go1.21.3.windows-amd64.msi	16/10/2023 18:41	Paquete de Windo...	59.836 KB
--	------------------	---------------------	-----------







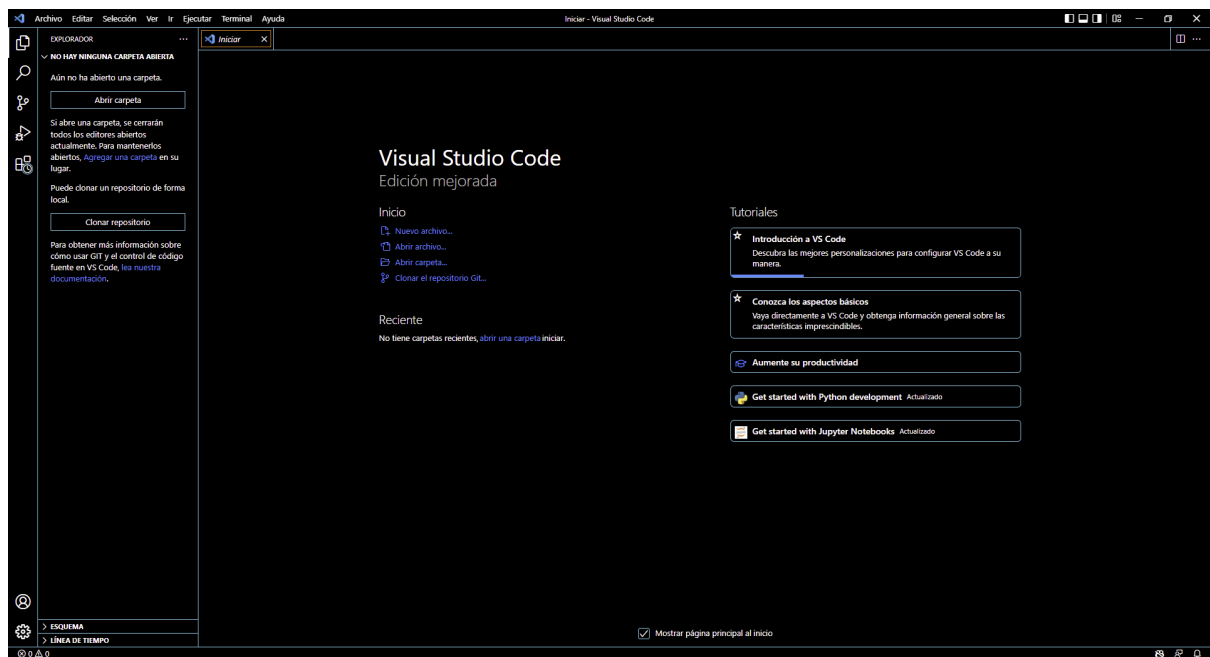
Una vez instalado go lo unico que tendremos que hacer es abrir un terminal de power shell y ejecutar el siguiente comando **go version**

A screenshot of a Windows PowerShell terminal window. The title bar says "Windows PowerShell". The terminal has a dark blue background with white text. The text in the terminal is as follows:
PS C:\Users\Eric> go version
go version go1.21.3 windows/amd64
PS C:\Users\Eric>
There is a small upward arrow icon on the right side of the terminal window, indicating a scroll bar.

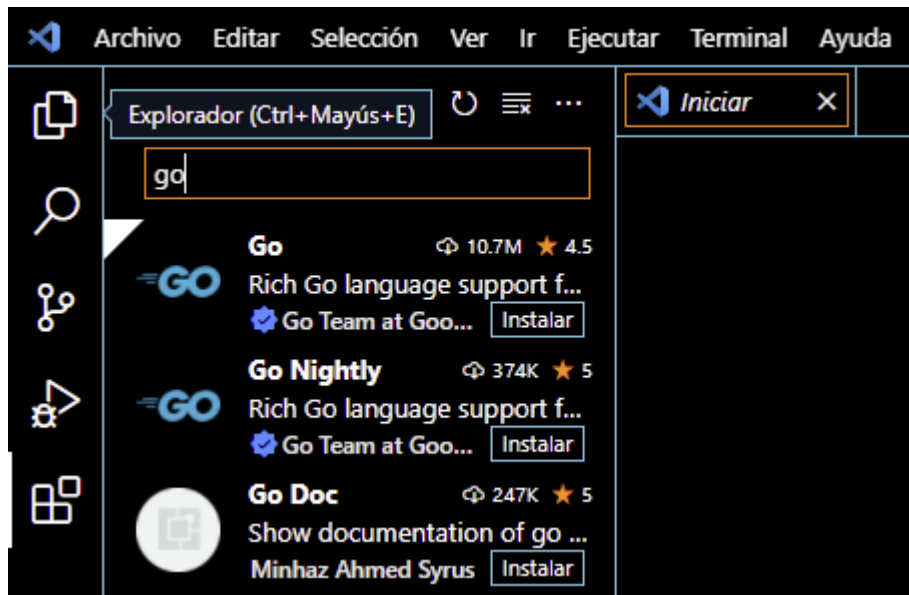
```
Seleccionar Windows PowerShell
PS C:\Users\Eric> go env
set GO111MODULE=
set GOARCH=amd64
set GOBIN=
set GOCACHE=C:\Users\Eric\AppData\Local\go-build
set GOENV=C:\Users\Eric\AppData\Roaming\go\env
set GOEXE=.exe
set GOEXPERIMENT=
set GOFLAGS=
set GOHOSTARCH=amd64
set GOHOSTOS=windows
set GOINSECURE=
set GOMODCACHE=C:\Users\Eric\go\pkg\mod
set GONOPROXY=
set GONOSUMDB=
set GOOS=windows
set GOPATH=C:\Users\Eric\go
set GOPRIVATE=
set GOPROXY=https://proxy.golang.org,direct
set GOROOT=C:\Program Files\Go
set GOSUMDB=sum.golang.org
set GOTMPDIR=
set GOTOOCHAIN=auto
set GOTOOOLDIR=C:\Program Files\Go\pkg\tool\windows_amd64
set GOVCS=
set GOVERSION=go1.21.3
set GCCGO=gccgo
set GOAMD64=v1
set AR=ar
set CC=gcc
set CXX=g++
set CGO_ENABLED=0
set GOMOD=NUL
set GOWORK=
set CGO_CFLAGS=-O2 -g
set CGO_CPPFLAGS=
set CGO_CXXFLAGS=-O2 -g
set CGO_FFLAGS=-O2 -g
set CGO_LDFLAGS=-O2 -g
set PKG_CONFIG=pkg-config
set GOGCCFLAGS=-m64 -fno-caret-diagnostics -Qunused-arguments -Wl,-no-gc-sections -fmessage-length=0 -ffile-prefix-map=C:\Users\Eric\AppData\Local\Temp\go-build2171087456=/tmp/go-build -gno-record-gcc-switches
PS C:\Users\Eric>
```

2. Visual studio code

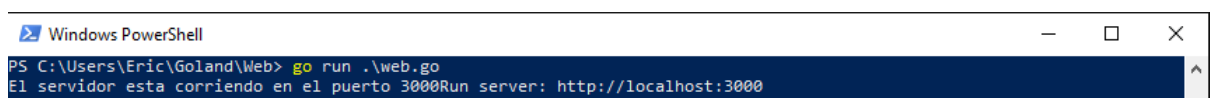
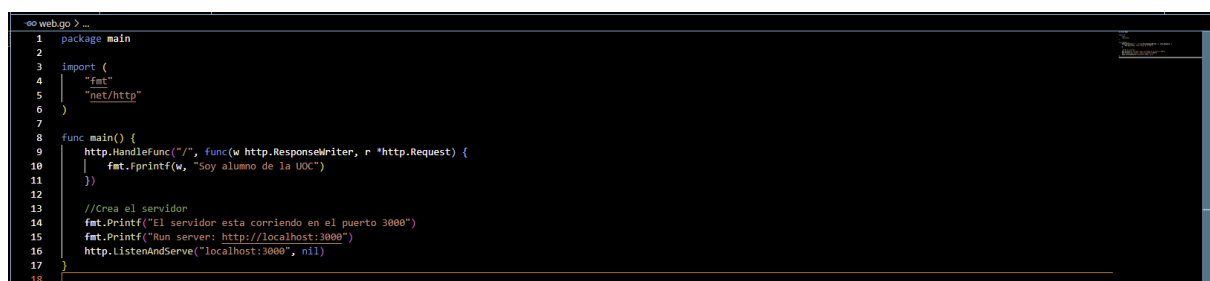
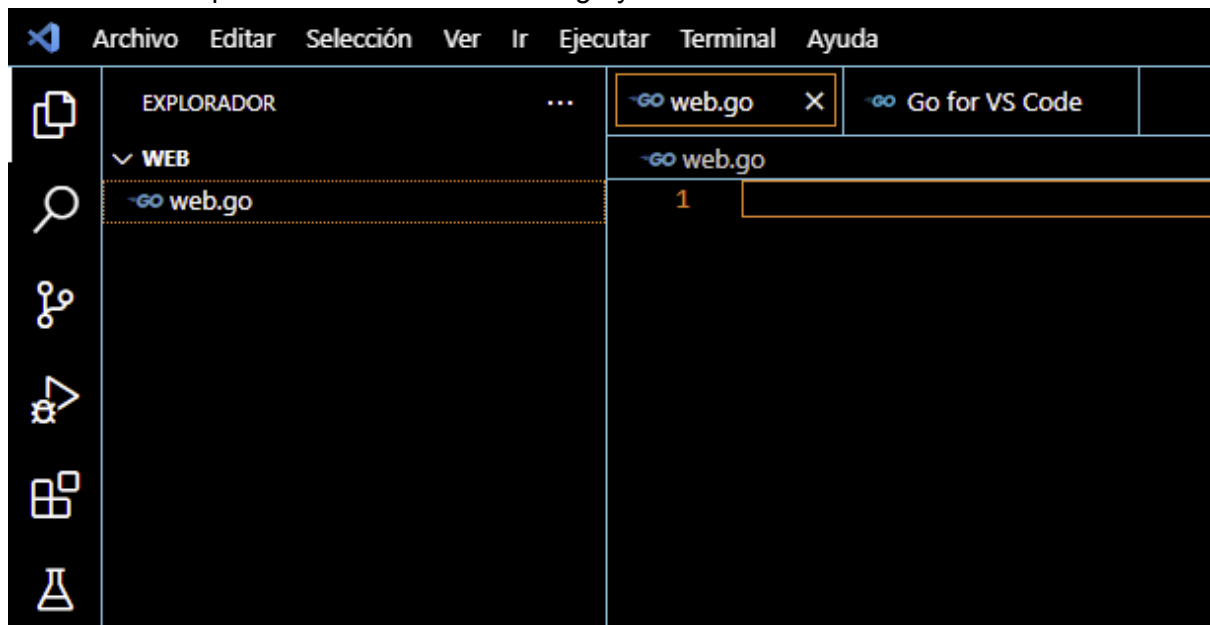
Para empezar a programar en go necesitaremos un programa para crear nuestros proyectos, en mi caso voy a utilizar visual studio code



Dentro de visual studio code instalaremos la extensión de go, podemos descargarlos desde las extensiones o creando nuestro archivo y llamando.



Primero crearemos una carpeta para nuestro proyecto, en mi caso la he llamado web, dentro de la carpeta crearemos un archivo .go y ahí crearemos nuestra web.



← → ↻ ⓘ localhost:3000

Soy alumno de la UOC

Tutoriales

<https://www.youtube.com/watch?v=Y9rejuvgvOE>

<https://www.youtube.com/watch?v=G8Du1EOuoLY>

3. Instalar Docker

Antes de instalar Docker desktop vamos a tener que crearnos una cuenta, nosotros utilizaremos Docker desktop

<https://www.docker.com/products/personal/>



Create your account

Signing up for Docker is fast and free.

Email

emolinerpe@uoc.edu

Username

EricUOC

Password

.....

☐

Send me occasional product updates and announcements.

This site is protected by reCAPTCHA and the Google [Privacy Policy](#) and [Terms of Service](#) apply.

Sign up

By creating an account I agree to the [Subscription Service Agreement](#), [Privacy Policy](#), [Data Processing Terms](#).

Already have an account? [Sign in](#)

hub.docker.com

docker hub

Search Docker Hub

ExploreRepositoriesOrganizationsHelp

Upgrade


ericuoc

Welcome to Docker


Download the desktop application

Download for Windows


Also available for Mac and Linux



Create a Repository
Push container images to a repository on Docker Hub.




Docker Hub Basics
Watch the guide on how to create and push your first image into a Docker Hub repository.




Language-Specific Guides
Learn how to containerize language-specific applications using Docker.


Access the world's largest library of container images




nginx




mongoDB




alpine




node




redis




busybox
Official




ubuntu
Official




python
Official

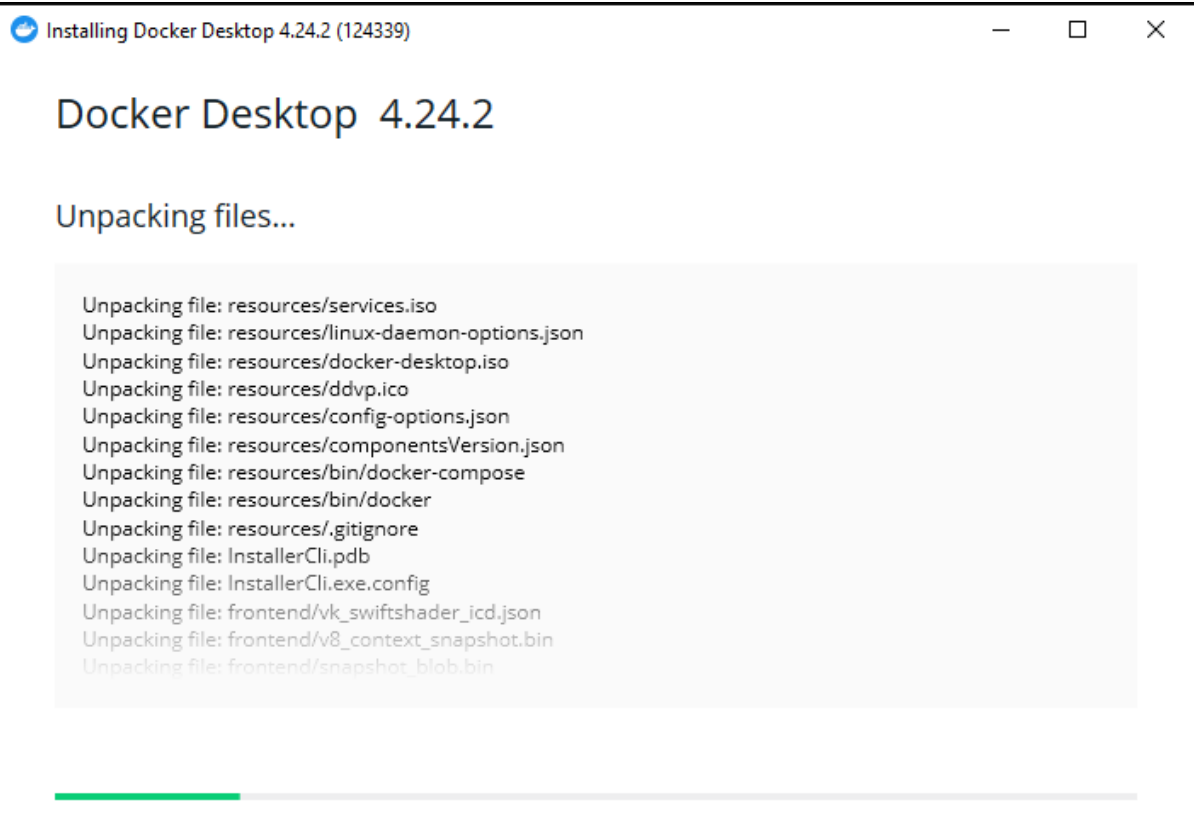
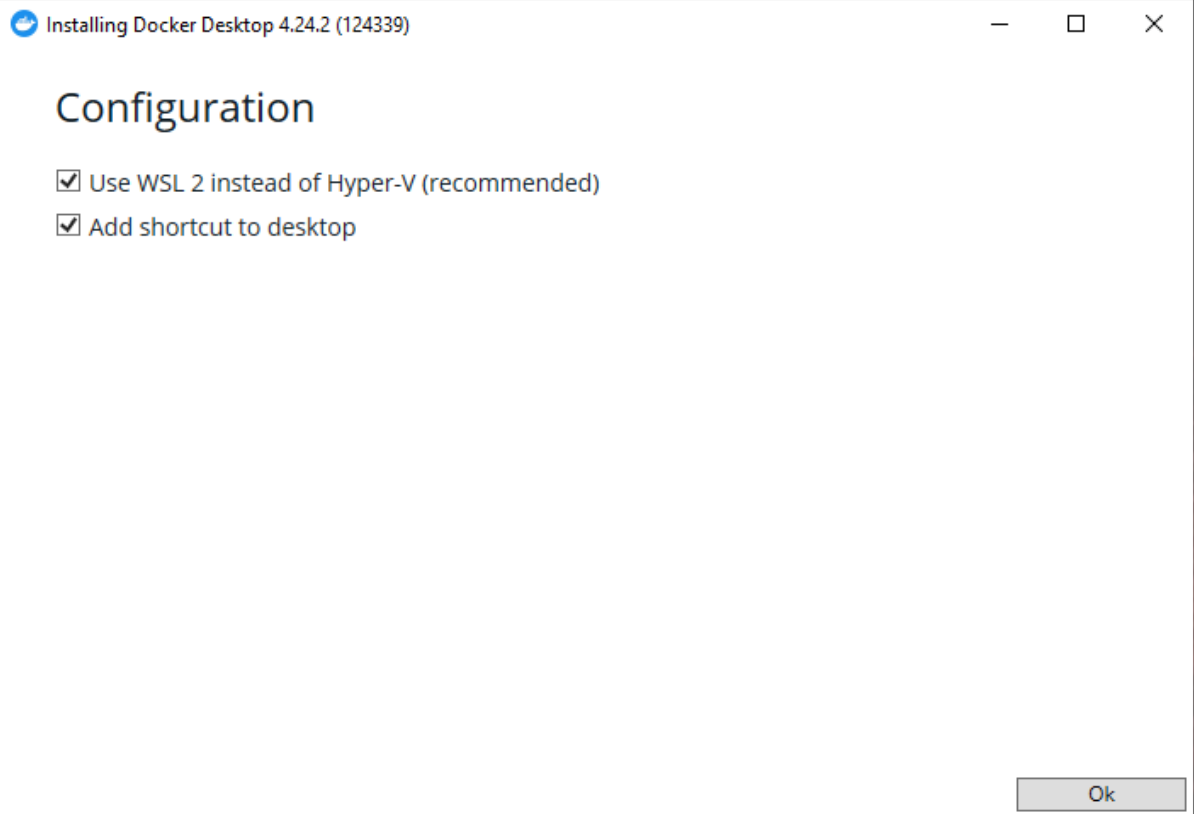


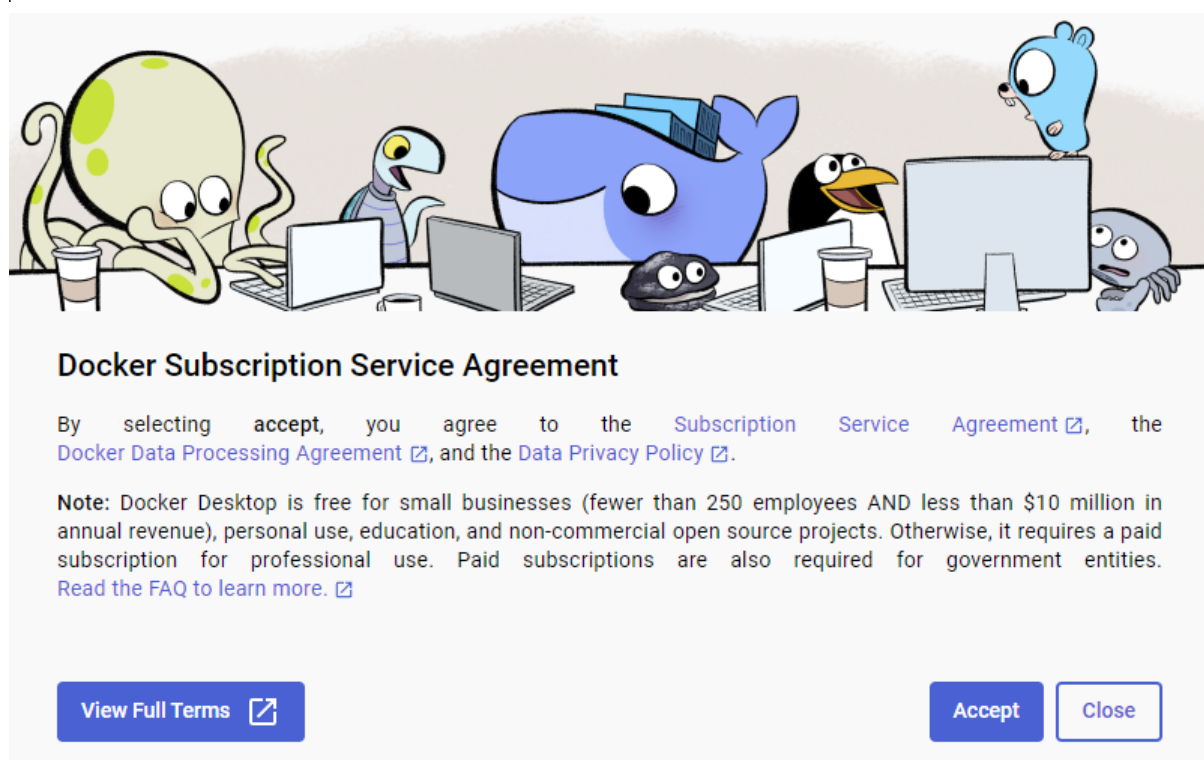
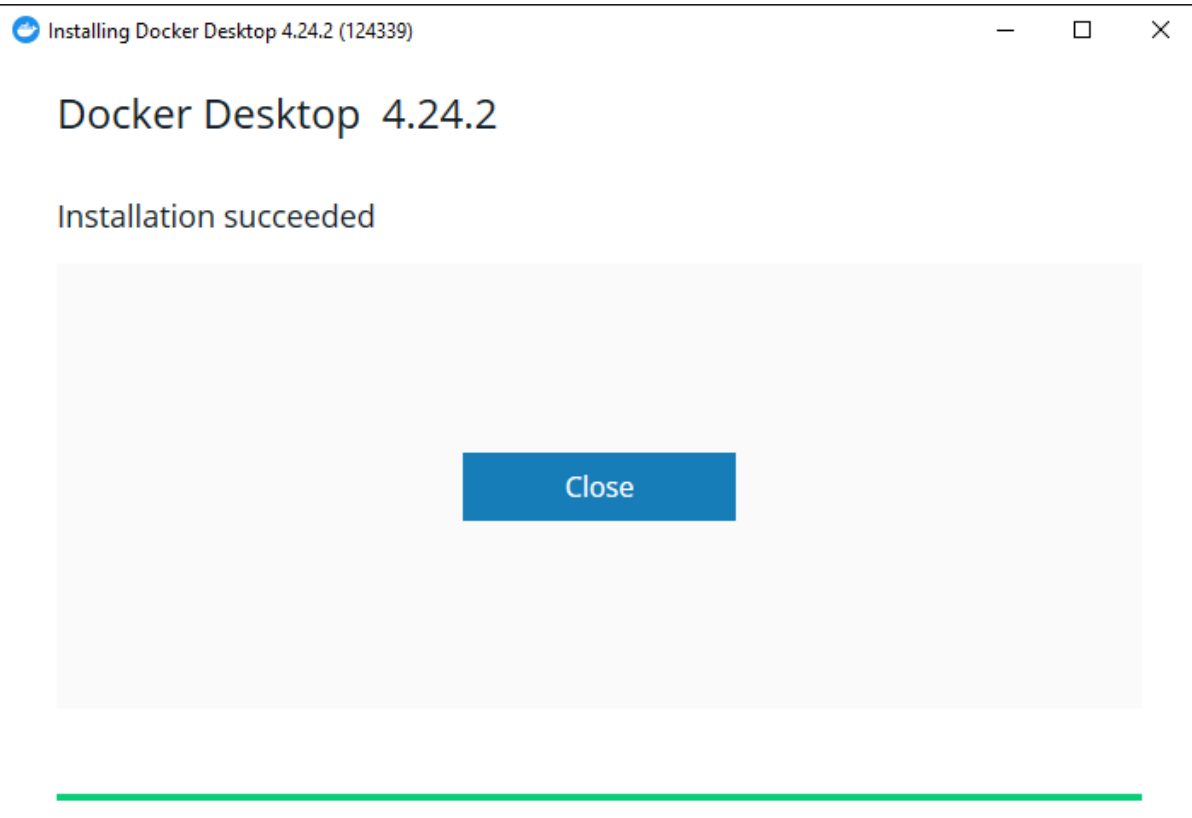
postgres
Official



httpd
Official

 Docker Desktop Installer.exe	16/10/2023 20:12	Aplicación	569.409 KB
--	------------------	------------	------------







Una vez instalado docker, lo primero que vamos ha hacer es instalar una imagen para asi poder crear un contenedor con nuestra aplicación.


Search
goland


Images (50) Containers (0) Volumes (0) Extensions (0) Docs (0)


Hub images (50) Remote repositories (0) Local images (0)


 golandsh/multi-server 347 · 0


 marktmilligan/goland 10K+ · 0 Tag latest Pull Run


 suckowbiz/goland 3.9K · 1

 rycus86/goland 2.9K · 6

 ericpaulsen/goland 10K+ · 0

 golandy/queuefunction 152 · 0

 golandsh/multi-client 135 · 0

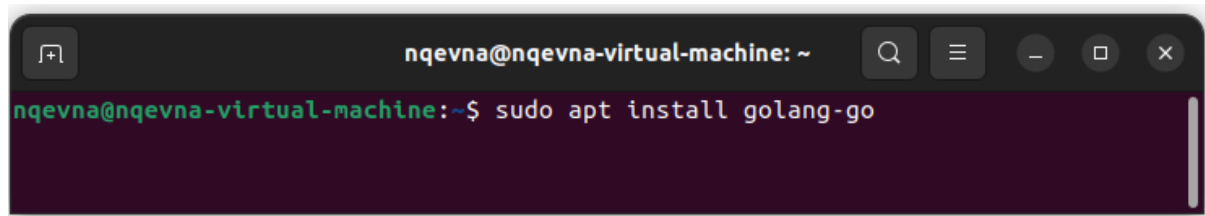
 golandsh/multi-worker 83 · 0

to open to navigate ESC to close SCROLL for more results Give feedback

En nuestro caso vamos a instalar una imagen de goland.

Preparación en Ubuntu sin UI

Instalación de Golang



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

Creación de la web con Go

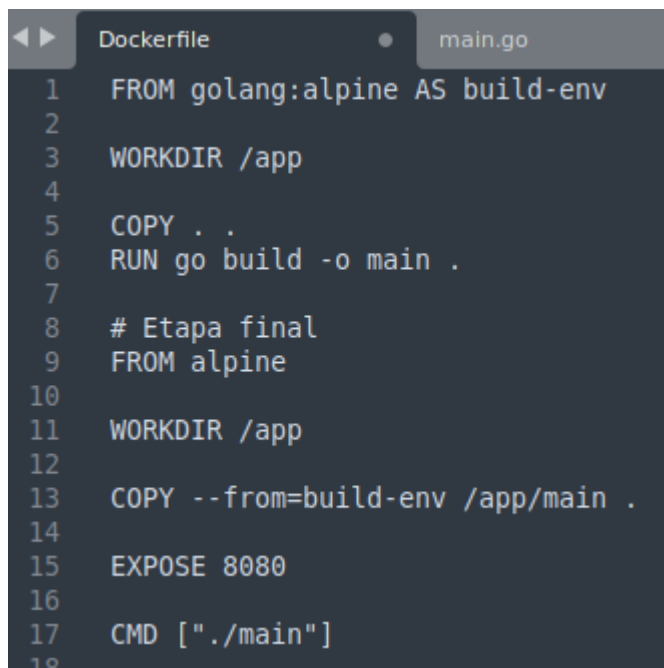


```
1  package main  
2  
3  import (  
4      "fmt"  
5      "net/http"  
6  )  
7  
8  func handler(w http.ResponseWriter, r *http.Request) {  
9      switch r.Method {  
10         case http.MethodGet:  
11             fmt.Fprintf(w, "Soy alumno de la UOC - Método GET")  
12         case http.MethodPost:  
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



```

1  FROM golang:alpine AS build-env
2
3  WORKDIR /app
4
5  COPY . .
6  RUN go build -o main .
7
8  # Etapa final
9  FROM alpine
10
11 WORKDIR /app
12
13 COPY --from=build-env /app/main .
14
15 EXPOSE 8080
16
17 CMD ["/main"]
18

```

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

- **Etapas 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.
- **Etapas 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 0.8s
=> [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:

```
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:~$
```