

Primeros pasos

Para que podamos empezar a hacer las configuraciones pertinentes en nuestro entorno de Docker, lo primero sería crear un proyecto y *dockerizarlo*, es decir, prepararlo para que pueda generar una imagen base del proyecto que nosotros creamos.

Los proyectos pueden estar en cualquier lenguaje y versión (Node, Python, Java, etc.).

¿Cómo creo un contenedor de mi aplicación?

Como mencionamos con anterioridad debemos dockerizar nuestro proyecto, pero cómo lo haremos?

Es importante que nosotros tengamos en la raíz de nuestro proyecto un archivo llamado Dockerfile, para eso solamente deberemos hacer:

```
touch Dockerfile
```

Esto ya creará el archivo vacío, es aquí donde nosotros vamos a empezar a agregar la lógica necesaria para que pueda correr de manera adecuada.

Si nos movemos a nuestro Dockerfile vamos a empezar creando el build de nuestra imagen.

¿Cómo configuro mi Dockerfile?

Build

La imagen de la build debe venir de algún lado, si en nuestro proyecto tenemos una arquitectura de node, ubuntu, python, o lo que sea vamos a indicarlo haciendo:

```
FROM node:latest
```

```
# Or
```

```
FROM Ubuntu:22.0
```

Aquí tenemos dos ejemplos, node y ubuntu, cómo funciona esta línea?

FROM: Origen de la imagen

node/ubuntu (etc.): Arquitectura de la imagen

:latest/:22.0: Versión de la arquitectura

OJO: Solo debemos usar un único FROM , a menos que estemos usando multi-stage builds (tema más avanzado). Por ahora, uno solo basta.

Si ya tenemos la imagen base, por ejemplo node:latest (Última versión disponible de node) vamos a continuar especificando la carpeta de trabajo de nuestra aplicación

```
WORKDIR /app
```

Con esto indicamos la carpeta de trabajo (La que dispara el proyecto, app por default en node)

Y seleccionamos los archivos necesarios para copiar, usualmente son los package, ya que estos tienen las configuraciones base del proyecto

```
COPY package*.json ./
```

Posteriormente haremos la instalación de las dependencias

```
RUN npm i
```

Copiamos el resto de los archivos

```
COPY . .
```

Exponemos el puerto de nuestra aplicación (Varia según nuestro caso específico)

```
EXPOSE 3000
```

Comando que correremos en el CMD

```
CMD ["npm", "start"]
```

Construyendo nuestra imagen

Para construir la imagen que definimos en el Dockerfile, usamos:

```
docker build ${context}
```

con el comando `docker build` estamos creando la imagen de nuestro proyecto, pero no es lo único que podemos hacer, ya que podemos marcar la ubicación en la que queremos guardar nuestra imagen haciendo uso de:

```
docker build -f image/Dockerfile image/
```

O ponerle una etiqueta

```
docker build -t image/nodejs .
```

Comprobamos que exista en nuestro repo

```
docker images image/nodejs
```

Una de las cosas que podemos hacer para mantener nuestro entorno más limpio, es el uso del `.dockerignore` para marcar aquellos archivos que no nos interese agregar al contenedor.

Podemos ver esto como una especie de `.gitignore`, pero orientado al contenedor de docker, ya que nosotros no deberíamos estar subiendo documentos temporales o generados al instalar dependencias, o generar caché, etc, por lo que podemos omitir estos.

¿Quieres conocer más información?

Si aún no nos queda claro lo que es un dockerfile, una imagen o cómo crearlo podemos usar la siguiente documentación en paralelo con esta.

<https://pilasguru.gitbooks.io/docker-guia-para-el-usuario/content/chapter03/04crear-dockerfile.html>

¿Cómo configuro mi dominio?

Suponiendo que ya tenemos un dominio comprado (si aún no lo tienes, puedes adquirir uno en [NameCheap](#) o con el proveedor de dominios de tu elección), y que contamos con conocimientos básicos sobre servidores, vamos a ver cómo configurarlo para que apunte a nuestro entorno de Docker.

También vamos a asumir que ya tenemos un **VPS (Servidor Privado Virtual)** activo — por ejemplo, contratado desde proveedores como DigitalOcean, AWS EC2, Vultr, etc.

En mi caso específico:

- **Proveedor de dominio:** NameCheap
- **Gestor DNS / Proxy inverso:** Cloudflare

OJO: Si no estás familiarizado con Cloudflare, puedes apoyarte con su documentación oficial: <https://developers.cloudflare.com/fundamentals/>

Veremos a continuación como hacer la conexión de nuestro dominio a los servicios de AWS para que podamos dar despliegue de nuestro proyecto.

Conectado mi dominio a AWS

Veremos que nuestro VPS será AWS, por lo que debemos hacer que este servicio pueda recibir las solicitudes cuándo alguien trate de entrar desde nuestro dominio, por ejemplo:

Flujo

Usuario 1 -> Accede a la web -> thinkguille.space -> apunta a nuestro proyecto desplegado en aws

¿Cómo ejecutamos nuestro flujo?

Para que nosotros podamos hacer ese redireccionamiento deberemos agregar registros tipo A a nuestro administrador de DNS

Administración de DNS para **thinkguille.space**

Permite revisar, agregar y editar registros de DNS. Las ediciones entrarán en vigor una vez guardadas.

Configuración de DNS: Completo ⓘ Importar y exportar ▼ ⚙ Configuración de la pantalla del Panel de control

Aquí vamos a agregar nuestro registro

REGISTRO TIPO A

Tipo: A

Nombre: @

Dirección IPv4: #Ip_pública_de_nuestro_servicio_EC2/VPS

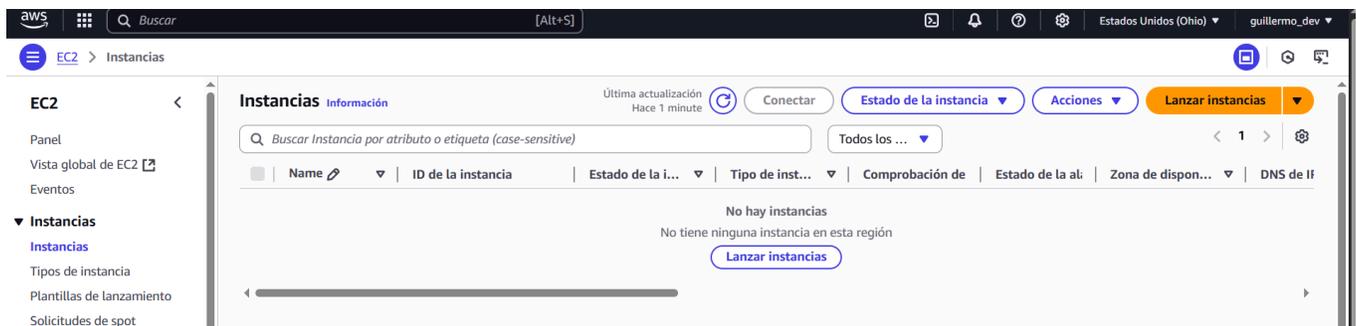
Estado de proxy: Solo DNS

¿Cómo consigo la IP pública de AWS?

Si ya tenemos una cuenta en AWS, podremos acceder a los servicios de EC2, aquí es dónde vamos a generar es ID pública para el despliegue y redirección de nuestro proyecto.

Paso 1. Crear Una Instancia de EC2

Lo primero que debemos hacer será escribir en el buscador de la consola de AWS **EC2**, para luego ir al apartado de instancias y empezar a crear nuestra instancia:



Al seleccionar el botón de **LANZAR INSTANCIAS** vamos a ver que se despliega una nueva vista, para esta vamos a llenar algunos campos cómo:

Campos De Instancia

Nombre: El nombre de nuestro servidor

Imagen De Aplicación: Ubuntu - Capa Gratuita

Tipo De Instancia: Default - Capa Gratuita

Par De Claves: Generadas al momento

Configuración De Red: Crear Grupo de seguridad - Permitir Tráfico SSH desde "dirección IP de nuestro dominio"

Almacenamiento: Default 8GB

Nombre:

Nombre y etiquetas [Información](#)

Nombre

servidor-archivero-digital

[Agregar etiquetas adicionales](#)

Imagen De Aplicación:

▼ Imágenes de aplicaciones y sistemas operativos (Imagen de máquina de Amazon) [Información](#)

Una AMI es una plantilla que contiene la configuración de software (sistema operativo, servidor de aplicaciones y aplicaciones) necesaria para lanzar la instancia. Busque o examine las AMI si no ve lo que busca a continuación.

🔍 *Busque en nuestro catálogo completo que incluye miles de imágenes de sistemas operativos y aplicaciones*

Inicio rápido



🔍 [Buscar más AMI](#)
Inclusión de AMI de AWS, Marketplace y la comunidad

Imágenes de máquina de Amazon (AMI)

Ubuntu Server 24.04 LTS (HVM), SSD Volume Type Apto para la capa gratuita ▼
ami-0d1b5a8c13042c939 (64 bits (x86)) / ami-019eeff96c2865995 (64 bits (Arm))
Virtualización: hvm Activado para ENA: true Tipo de dispositivo raíz: ebs

Descripción

Ubuntu Server 24.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).

Canonical, Ubuntu, 24.04, amd64 noble image

Arquitectura

64 bits (x86) ▼

ID de AMI

ami-0d1b5a8c13042c939

Fecha de publicación

2025-06-10

Nombre de usuario

ubuntu

Proveedor verificado

Tipo De Instancia:

▼ Tipo de instancia [Información](#) | [Obtener asesoramiento](#)

Tipo de instancia

t2.micro

Apto para la capa gratuita

Familia: t2 1 vCPU 1 GiB Memoria Generación actual: true

Bajo demanda Ubuntu Pro base precios: 0.0134 USD por hora Bajo demanda Linux base precios: 0.0116 USD por hora

Bajo demanda SUSE base precios: 0.0116 USD por hora Bajo demanda Windows base precios: 0.0162 USD por hora

Bajo demanda RHEL base precios: 0.026 USD por hora

Todas las generaciones

[Comparar tipos de instancias](#)

Se aplican costos adicionales a las AMI con software preinstalado

Par De Claves:

Este paso es fundamental, ya que será el método con el que nosotros vamos a conectarnos a nuestro servidor a través de **SSH**. Si no tienes una clave ya creada, puedes generar una directamente desde el mismo asistente de lanzamiento:

Crear par de claves



Nombre del par de claves

Con los pares de claves es posible conectarse a la instancia de forma segura.

El nombre puede incluir hasta 255 caracteres ASCII. No puede incluir espacios al principio ni al final.

Tipo de par de claves

RSA
Par de claves pública y privada cifradas mediante RSA

ED25519
Par de claves privadas y públicas cifradas ED25519

Formato de archivo de clave privada

.pem
Para usar con OpenSSH

.ppk
Para usar con PuTTY

 Cuando se le solicite, almacene la clave privada en un lugar seguro y accesible del equipo. **Lo necesitará más adelante para conectarse a la instancia.** [Más información](#) 

Cancelar

Crear par de claves

🔗 ¿Cómo llenar este formulario?

- **Nombre del par de claves:** Puedes escribir algo como `llave-mi-servidor`
- **Tipo del par de claves:** RSA
- **Formato del archivo de clave privada:** `.pem` (para usarlo con OpenSSH en Linux o Mac; también compatible con Windows si usas WSL o Git Bash)

Configuraciones De Red:

Aquí es dónde nosotros vamos a estar definiendo los puertos que vamos a escuchar, para generar el acceso a internet a nuestra instancia en EC2

▼ Configuraciones de red Información

Editar

Red | [Información](#)

vpc-01308669b325c24f6

Subred | [Información](#)

Sin preferencias (subred predeterminada en cualquier zona de disponibilidad)

Asignar automáticamente la IP pública | [Información](#)

Habilitar

Se aplican cargos adicionales cuando no se cumplen los límites del nivel gratuito

Firewall (grupos de seguridad) | [Información](#)

Un grupo de seguridad es un conjunto de reglas de firewall que controlan el tráfico de la instancia. Agregue reglas para permitir que un tráfico específico llegue a la instancia.

Crear grupo de seguridad

Seleccionar un grupo de seguridad existente

Crearemos un nuevo grupo de seguridad denominado "launch-wizard-1" con las siguientes reglas:

Permitir el tráfico de SSH desde

Ayuda a establecer conexión con la instancia

Mi IP

187.237.125.90/32

Permitir el tráfico de HTTPS desde Internet

Para configurar un punto de enlace, por ejemplo, al crear un servidor web

Permitir el tráfico de HTTP desde Internet

Para configurar un punto de enlace, por ejemplo, al crear un servidor web

⚠ Las reglas con origen 0.0.0.0/0 permiten que todas las direcciones IP tengan acceso a la instancia. Le recomendamos que configure las reglas del grupo de seguridad para permitir el acceso únicamente desde direcciones IP conocidas. ✕

¿Qué es un grupo de seguridad?

Es cómo un firewall que decide el tipo de tráfico que puede entrar a nuestro servidor, por lo que debemos habilitar al menos:

- Tráfico **SSH (puerto 22)** para conectarnos al servidor
- Tráfico **HTTP (puerto 80)** o **HTTPS (puerto 443)** si vamos a mostrar un sitio web
- O el puerto que uses con Docker (por ejemplo, el 3000 o 8080)

Habilitar el puerto de nuestra aplicación

Como nuestro proyecto se ejecuta en el puerto 8000 (ya sea por configuración del Dockerfile o docker-compose), necesitamos agregar una regla personalizada:

1. Baja un poco y haz clic en "**Agregar regla de tráfico**".
2. Selecciona:
 - **Tipo:** Personalizado TCP
 - **Puerto:** 8000
 - **Origen:** Cualquier lugar (0.0.0.0/0)

Descripción - **obligatorio** | Información

launch-wizard-1 created 2025-07-11T18:11:20.288Z

Reglas de grupos de seguridad de entrada

▼ Regla del grupo de seguridad 1 (TCP, 22, 0.0.0.0/0) Eliminar

Tipo | Información: ssh

Protocolo | Información: TCP

Intervalo de puertos | Información: 22

Tipo de origen | Información: Cualquier lugar

Origen | Información:

Descripción - **opcional** | Información:

▼ Regla del grupo de seguridad 2 (TCP, 8000, 0.0.0.0/0) Eliminar

Tipo | Información: TCP personalizado

Protocolo | Información: TCP

Intervalo de puertos | Información: 8000

Tipo de origen | Información: Cualquier lugar

Origen | Información:

Descripción - **opcional** | Información:

⚠ Las reglas con origen 0.0.0.0/0 permiten que todas las direcciones IP tengan acceso a la instancia. Le recomendamos que configure las reglas del grupo de seguridad para permitir el acceso únicamente desde direcciones IP conocidas.

Agregar regla del grupo de seguridad

Obteniendo la IP pública

Una vez creada la instancia, podrás ver su IP pública desde el panel principal de EC2.

Configurando nuestro registro tipo A

Ahora podemos volver a nuestro manejador de DNS y terminar la configuración

<input type="checkbox"/>	Tipo ⓘ	Nombre ⓘ	Contenido ⓘ	Estado de proxy ⓘ	TTL ⓘ	Acciones
<input type="checkbox"/>	A	thinkguille.space	3.17.64.132	Solo DNS	Automático	Editar ▶

Subiendo La Imagen De Mi Proyecto

Una vez que hemos creado correctamente la imagen Docker de nuestro proyecto en local, el siguiente paso es subir esa imagen a un **registro remoto**, como por ejemplo:

- **Docker Hub** (<https://hub.docker.com/>)
- **Amazon Elastic Container Registry (ECR)**
- **GitHub Container Registry**
- **Google Container Registry**, entre otros.

En este ejemplo, vamos a utilizar **Docker Hub**, por ser uno de los más conocidos y de fácil acceso.

NOTA

En este ejemplo estamos suponiendo que ya contamos con una cuenta en docker hub, y que logramos acceder a nuestra cuenta.

```
Login Succeeded
PS C:\Users\x> |
```

Paso 1: Etiquetar la imagen

El primer paso será crear la imagen de nuestro proyecto, en el caso específico de esta documentación estamos trabajando con un proyecto en fastApi, por lo que hemos usado el comando:

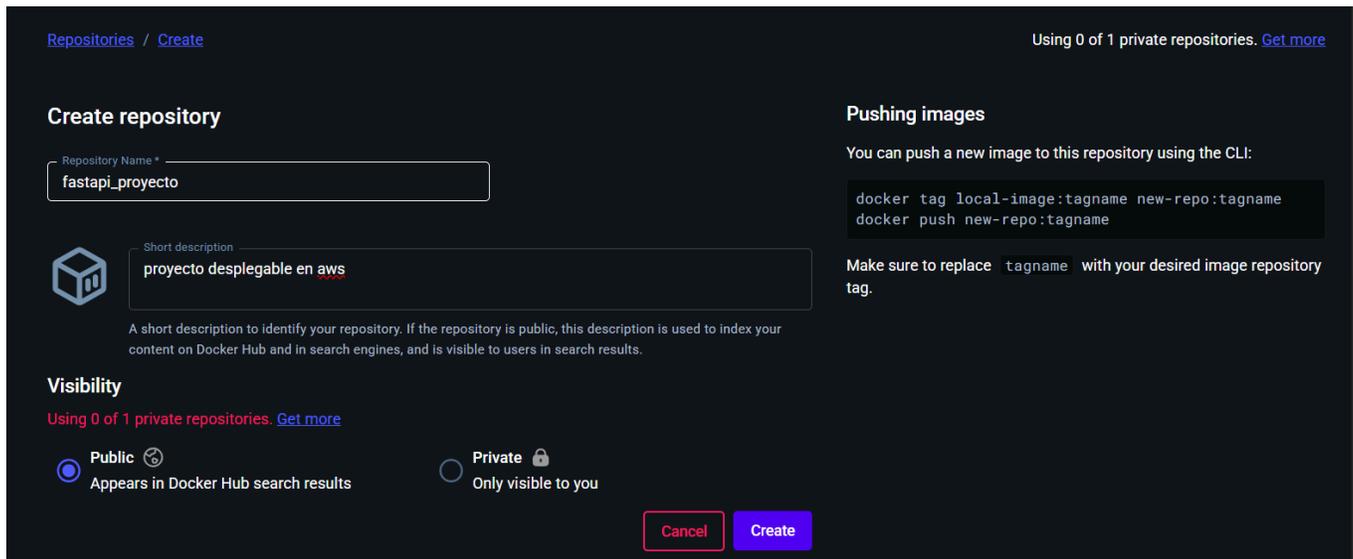
```
docker build -t fastapi_proyecto .
```

```
PS C:\Users\x\Desktop\FastApi_Proyecto> docker build -t fastapi_proyecto .
[+] Building 85.8s (5/11)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 432B
=> [internal] load metadata for docker.io/library/ubuntu:22.04
=> [auth] library/ubuntu:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/6] FROM docker.io/library/ubuntu:22.04@sha256:3c61d3759c2639d4b836d32a2d3c83fa0214e36f195a3421018dbaaf79cbe37f
=> => resolve docker.io/library/ubuntu:22.04@sha256:3c61d3759c2639d4b836d32a2d3c83fa0214e36f195a3421018dbaaf79cbe37f
=> => sha256:3c61d3759c2639d4b836d32a2d3c83fa0214e36f195a3421018dbaaf79cbe37f 6.69kB / 6.69kB
=> => sha256:08e2cd26ee66d0d46d6394df594f2877fc9b9381d9630a9ef5d86e27dfae9a95 424B / 424B
=> => sha256:1b668a2d748d748b0460ac95024ec80af7b663ede9416c235a9c102556bc1780 2.30kB / 2.30kB
=> => sha256:e735f3a6b70199ad991c5715d965a4d858540eca2be18be0d889698e5a0a3e8c 29.54MB / 29.54MB
=> => extracting sha256:e735f3a6b70199ad991c5715d965a4d858540eca2be18be0d889698e5a0a3e8c
=> [internal] load build context
=> => transferring context: 371.38MB
=> [2/6] RUN apt-get update && apt-get install -y python3 python3-pip python3-venv tesseract-ocr libgl1 && apt-get clean
=> => # Setting up libc-devtools (2.35-0ubuntu3.10) ...
=> => # Setting up python3-pkg-resources (59.6.0-1.2ubuntu0.22.04.3) ...
=> => # Setting up python3-distutils (3.10.8-1~22.04) ...
=> => # Setting up libgl1-amber-dri:amd64 (21.3.9-0ubuntu1~22.04.1) ...
=> => # Setting up python3.10-venv (3.10.12-1~22.04.10) ...
=> => # Setting up python3-setuptools (59.6.0-1.2ubuntu0.22.04.3) ...
```

Una vez que tengamos lista nuestra imagen creada, lo siguiente sería marcar esta imagen, para que nosotros podamos identificarla y llevar un control más estructurado de todas las imágenes de nuestro proyecto, sirviendo como control de versiones.

Repositorios

Debemos de tener creado un repositorio dentro de dockerhub para que podamos hacer el push de nuestra imagen a nuestra cuenta.



Repositories / Create Using 0 of 1 private repositories. [Get more](#)

Create repository

Repository Name *

Short description

A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.

Visibility

Using 0 of 1 private repositories. [Get more](#)

Public  Appears in Docker Hub search results

Private  Only visible to you

Pushing images

You can push a new image to this repository using the CLI:

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to replace `tagname` with your desired image repository tag.

Nombramos con:

```
docker tag fastapi_proyecto guillermo45/fastapi_proyecto
```

Y pusheamos nuestra imagen usando:

```
docker push guillermo45/fastapi_proyecto
```

Si todo sale bien deberíamos ver el siguiente resultado en la consola:

```
PS C:\Users\x\Desktop\FastApi_Proyecto> docker push guillermo45/fastapi_proyecto
Using default tag: latest
The push refers to repository [docker.io/guillermo45/fastapi_proyecto]
260e3aeb328a: Pushed
ae31f37d9ad9: Pushed
24fd72bad5a8: Pushed
728cf8671028: Pushed
d46d9cc34be2: Pushed
8d6b7eb76b62: Mounted from library/ubuntu
latest: digest: sha256:0247cb206d9a9a7507c3af8e8c1fd820a641e8df0402a86b6d5612933d7881ab size: 1585
```

Paso 2: Conectarnos a nuestro servidor con EC2

Debemos abrir la consola de nuestro proyecto para poder conectarnos a nuestra instancia de EC2 la cuál creamos en pasos anteriores, una vez que la hayamos desplegado, usaremos el comando:

```
ssh -i ssh -i C:\directory\acceso-servidor.pem ubuntu@mi-ip-publica
```

Si hemos seguido todos los pasos hasta este momento podremos conectarnos a nuestro servidor por ssh, y veremos algo cómo:

```
*** System restart required ***

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-40-83:~$
```

Paso 3: Descargar nuestra imagen alojada en docker hub

Con el paso anterior pudimos ver que nuestro servidor está corriendo de manera activa, ahora podremos bajar nuestra imagen con el comando:

```
docker pull usuario/repositorios
```

En caso de que nuestro entorno no reconozca docker, vamos a instalarlo haciendo uso de

```
sudo apt update
sudo apt install docker.io -y
```

Después tendremos que iniciar con nuestras credenciales usando:

```
docker login
```

Accesos

Esto nos va a llevar a autorizar nuestra entrada desde el navegador, tendremos que abrir el enlace dado en la terminal e insertar el código.

```
ubuntu@ip-172-31-40-83:~$ docker login

USING WEB-BASED LOGIN
To sign in with credentials on the command line, use 'docker login -u <username>'

Your one-time device confirmation code is: NCXT-VQPS
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate

Waiting for authentication in the browser...
█
```

Una vez que confirmemos en el navegador nuestro código, entonces veremos que en nuestra terminal aparece la confirmación de ingreso.

```
Waiting for authentication in the browser...
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

Login Succeeded
ubuntu@ip-172-31-40-83:~$ █
```

Con esto ya podremos ejecutar de manera correcta nuestro docker pull, si hasta ahora todo va bien nuestro comando nos traerá nuestra imagen creada previamente, lo que nos llevaría a usar el comando:

```
sudo docker run -d -p 8000:8000 usuario/repositorios
```

 **Dato**

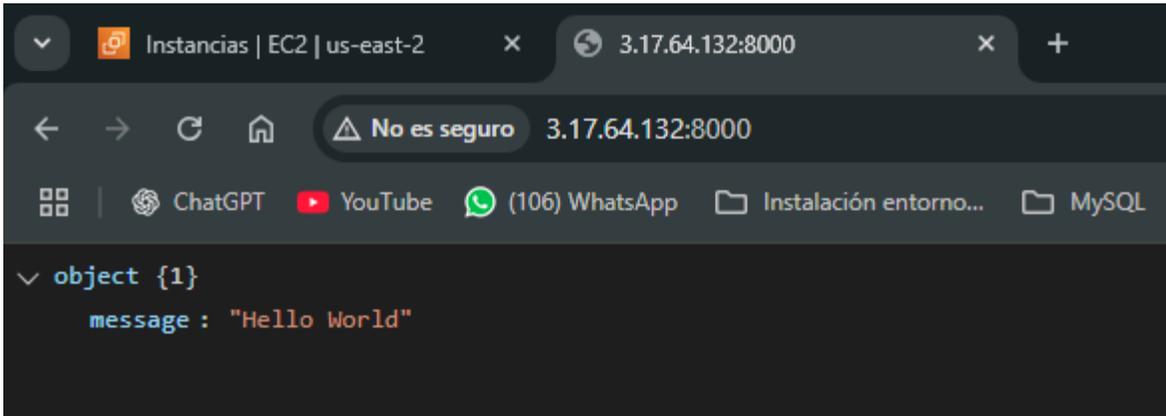
El primer número 8000 representa el puerto público que vamos a estar exponiendo en EC2 y el segundo será el puerto interno del contenedor, este debe coincidir con el que pusimos en Dockerfile

```
ubuntu@ip-172-31-40-83:~$ sudo docker run -d -p 8000:8000 guillermo45/fastapi_proyecto
Unable to find image 'guillermo45/fastapi_proyecto:latest' locally
latest: Pulling from guillermo45/fastapi_proyecto
e735f3a6b701: Extracting [=====>] 3.604MB/29.54MB
50d897b57346: Downloading [=====>] 178MB/256.4MB
da2db138d40b: Download complete
a3e532f9d48f: Downloading [=====>] 164.3MB/275.5MB
e5b4289e34eb: Download complete
1f3b55b86a2a: Downloading [=====>] 98.53MB/219.2MB
█
```

Al estar dentro de la terminal de Ubuntu deberemos pasar el sudo para dar los máximos permisos y ejecutar docker.

Configuraciones de red

Hasta este punto, podremos ver nuestra aplicación corriendo desde nuestra IP Pública y el puerto que especificamos



Sin embargo, aún nos hace falta 2 cosas importantes.

1. Agregar un certificado SSL
2. Conectar nuestro propio dominio

A continuación veremos una serie de pasos para lograr configurar de manera correcta nuestro proyecto.

Agregando un certificado SSL

El objetivo final es agregar nuestro dominio, algunos ya proporcionan de forma automática un certificado SSL, sin embargo vamos a ver cómo podemos manejar este de forma manual, en caso de que no tengamos este servicio.

¿Es la única opción?

Aunque en esta parte de la guía estaremos usando CERBOT, aws ya ofrece una opción para certificar nuestros proyectos, sin embargo esta documentación está trabajando sin balanceadores de carga, por lo que todas las configuraciones serán manuales.

Paso 1: Instalando Nginx

Haremos uso de Nginx para poder instalar y manejar Certbot, dentro de nuestra instancia EC2, los comandos que estaremos usando serán:

```
sudo apt install nginx -y
```

Posteriormente a la instalación, vamos a verificar que el servicio esté corriendo de manera adecuada:

```
sudo systemctl status nginx
```

```
ubuntu@ip-172-31-40-83:~$ sudo systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: enabled)
   Active: active (running) since Thu 2025-07-17 15:52:07 UTC; 17s ago
     Docs: man:nginx(8)
   Process: 29096 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
   Process: 29097 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
  Main PID: 29126 (nginx)
    Tasks: 2 (limit: 1124)
   Memory: 2.3M (peak: 4.2M)
      CPU: 18ms
   CGroup: /system.slice/nginx.service
           └─29126 "nginx: master process /usr/sbin/nginx -g daemon on; master_process on;"
             └─29128 "nginx: worker process"

Jul 17 15:52:07 ip-172-31-40-83 systemd[1]: Starting nginx.service - A high performance web server and a reverse proxy server...
Jul 17 15:52:07 ip-172-31-40-83 systemd[1]: Started nginx.service - A high performance web server and a reverse proxy server.
ubuntu@ip-172-31-40-83:~$
```

Si todo ha salido bien, veremos que ya tenemos el estatus `active (running)`.

Paso 2: Redireccionando a Docker

Ahora el siguiente paso será hacer una redirección desde nuestro Nginx, para que este mande todas las solicitudes HTTPS a nuestro contenedor (puerto 8000), por lo que vamos a entrar a la siguiente ruta: `/etc/nginx/sites-available/default`

Acceder a nuestro documento

Nos encontramos dentro del servidor, por lo que deberemos navegar entre carpetas para encontrar la ruta, deberemos retroceder dos espacios atrás y luego acceder.

Nos movemos dos espacios atrás con `cd ../../`

```
ubuntu@ip-172-31-40-83:~$ ls
ubuntu@ip-172-31-40-83:~$ cd ../../
ubuntu@ip-172-31-40-83:/$ ls
bin bin.usr-is-merged boot dev etc home lib lib.usr-is-merged lib64 lost-found media mnt opt proc root run sbin sbin.usr-is-merged snap srv sys tmp usr var
ubuntu@ip-172-31-40-83:/$ cd etc
```

Nos movemos a `etc/nginx/sites-available` con `cd etc/nginx/sites-available`

```

ubuntu@ip-172-31-40-83:/etc$ ls
ModemManager      ca-certificates  dbus-1           fwupd            hosts.deny        libblockdev      mdadm            nftables.conf
NetworkManager    ca-certificates.conf  debconf.conf    gai.conf         init              libibverbs.d     mime.types      nginx
PackageKit         chroot            debian_version   gnutls           init.d            libnl-3           mke2fs.conf     nsswitch.conf
X11                cloud             default          groff            initramfs-tools  locale.alias     modprobe.d      opt
acpi               cni              deluser.conf    group            inputrc           locale.conf      modules          os-release
adduser.conf       console-setup     depmod.d        group            iproute2         locale.gen       modules-load.d  overlayroot.conf
alternatives       credstore         dhcp            grub.d           iscsi            localtime        mtab            overlayroot.local.conf
apparmor           credstore.encrypted  dhcpcd.conf     gshadow          issue            logcheck         multipath        pam.conf
apparmor.d         cron.d            dnsmasq.d       gshadow          issue.net        login.defs       multipath.conf  pam.d
appot              cron.daily        docker          gss              kernel           logrotate.conf  nanorc          passwd
apt                cron.hourly       dpkg            hdparm.conf     landscape        logrotate.d      needrestart     passwd-
bash.bashrc        cron.monthly      e2scrub.conf    hibagent-config.cfg  ld.so.cache     lsb-release     netconfig       perl
bash_completion   cron.weekly       ec2_version     hibinit-config.cfg  ld.so.conf       lvm              netplan         pki
bash_completion.d cron.yearly       environment     host.conf        ld.so.conf.d     machine-id       network         plymouth
bindresvport.blacklist  crontab          ethertypes     hostname         ldap             magic            networkd-dispatcher  pm
binfmt.d           cryptsetup-initramfs  fstab          hosts            legal            magic.mime       networks        polkit-1
byobu              crypttab          fuse.conf       hosts.allow      libaudit.conf    manpath.config  newt            pollinate
ubuntu@ip-172-31-40-83:/etc$ cd nginx
ubuntu@ip-172-31-40-83:/etc/nginx$ ls
conf.d  fastcgi.conf  fastcgi_params  koi-utf  koi-win  mime.types  modules-available  modules-enabled  nginx.conf  proxy_params  scgi_params  sites-available

```

Modificamos el archivo default con `nano default`

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE  GITLENS
GNU nano 7.2
server {
    listen 80;
    server_name thinkguille.space www.thinkguille.space;

    location / {
        proxy_pass http://localhost:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}

```

Y pasamos una configuración similar a la mostrada en la ilustración, en dónde vamos a reemplazar el server name con nuestro dominio real.

```

server {
    listen 80;
    server_name thinkguille.space www.thinkguille.space;

    location / {
        proxy_pass http://localhost:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}

```

Con esto configurado podremos instalar ahora nuestro Cerbot.

¿Qué es lo que hicimos?

Gracias a la configuración anterior, Nginx sabe que cuando accedan a nuestro dominio, en este caso `thinkguille.space` (o tu propio dominio), redirigirá las solicitudes HTTP al puerto `8000` del servidor, que es donde se encuentra expuesto nuestro contenedor Docker.

Docker se encarga de mapear ese puerto al interior del contenedor donde corre nuestra aplicación, por lo que en términos simples el flujo sería:

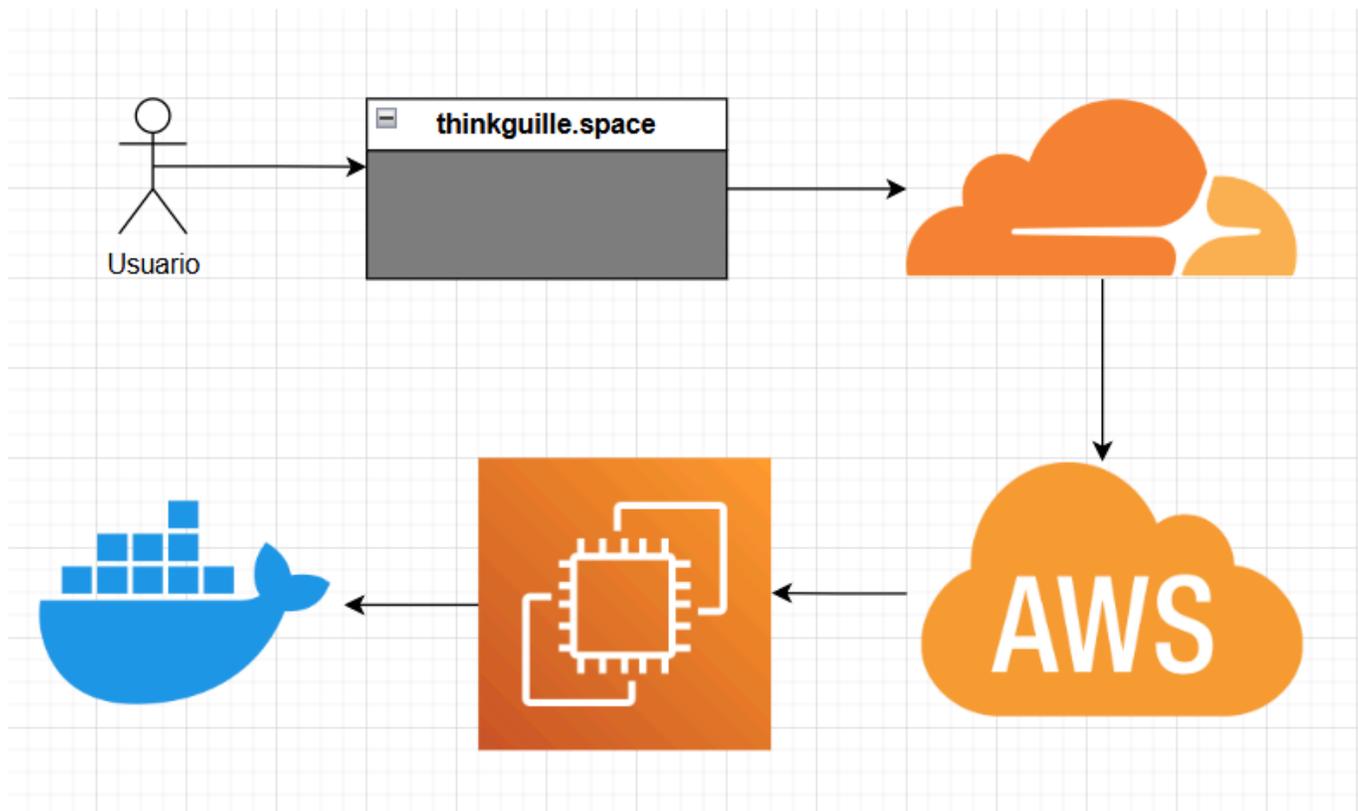
Usuario -> `thinkguille.space`

Cloudflare -> Redirecciona al servidor de AWS

AWS -> Instancia EC2

EC2 -> Puerto 8000 expuesto por Docker

Docker -> Contenedor corriendo el proyecto



Paso 3: Instalando Certbot

Una vez que nos ha quedado claro el proceso que sigue nuestra configuración, vamos a habilitar el certificado HTTPS, haciendo uso de Certbot (Let's Encrypt Service), configuraremos este ejecutando el siguiente comando:

```
sudo apt install certbot python3-certbot-nginx -y
```

```
ubuntu@ip-172-31-40-83:~$ sudo certbot --nginx
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Enter email address (used for urgent renewal and security notices)
(Enter 'c' to cancel): guillermo.jesus.garcia.canul@gmail.com
```

Consideraciones

Nos va a pedir el correo con el cuál nosotros registramos nuestra cuenta de aws

Posteriormente de su instalación, podremos ejecutarlo y disparar nuestro certificado SSL, por lo que haremos

```
sudo certbot --nginx
```

Configuraciones de red

En caso de que certbot falle, podemos forzar su conexión cambiando nuestro grupo de seguridad de la instancia y agregar una entrada para el puerto 80 y que este escuche todos los orígenes.



ID de la regla del grupo de seguridad	Tipo <small>Información</small>	Protocolo <small>Información</small>	Intervalo de puertos <small>Información</small>	Origen <small>Información</small>	Descripción: opcional <small>Información</small>	
sgr-0e8bb400d47ae6eee	TCP personalizado	TCP	8000	Persona...		Eliminar
sgr-0907b30ab9f4bdb06	SSH	TCP	22	Persona...		Eliminar
-	TCP personalizado	TCP	80	Anywhe...		Eliminar

[Agregar regla](#)

⚠ Las reglas cuyo origen es 0.0.0.0/0 o ::/0 permiten a todas las direcciones IP acceder a la instancia. Recomendamos configurar reglas de grupo de seguridad para permitir el acceso únicamente desde direcciones IP conocidas. [X](#)

[Cancelar](#) [Previsualizar los cambios](#) [Guardar reglas](#)

Si todo ha salido bien verás el siguiente mensaje en tu consola

```
ubuntu@ip-172-31-40-83:~$ sudo certbot --nginx
Saving debug log to /var/log/letsencrypt/letsencrypt.log

Which names would you like to activate HTTPS for?
We recommend selecting either all domains, or all domains in a VirtualHost/server block.
-----
1: thinkguille.space
2: www.thinkguille.space
-----
Select the appropriate numbers separated by commas and/or spaces, or leave input
blank to select all options shown (Enter 'c' to cancel):
Requesting a certificate for thinkguille.space and www.thinkguille.space

Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/thinkguille.space/fullchain.pem
Key is saved at: /etc/letsencrypt/live/thinkguille.space/privkey.pem
This certificate expires on 2025-10-15.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.

Deploying certificate
Successfully deployed certificate for thinkguille.space to /etc/nginx/sites-enabled/default
Successfully deployed certificate for www.thinkguille.space to /etc/nginx/sites-enabled/default
Congratulations! You have successfully enabled HTTPS on https://thinkguille.space and https://www.thinkguille.space

-----
If you like Certbot, please consider supporting our work by:
* Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate
* Donating to EFF: https://eff.org/donate-le
-----
```

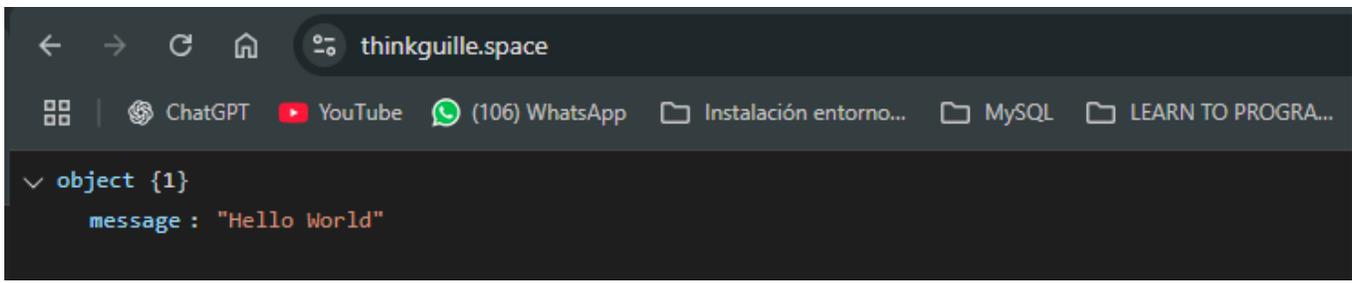
Por lo que ya puedes contar que tienes un certificado SSL.

¿Aún no puedes ver tu proyecto en la web?

En caso de que no puedas ver tu proyecto, debemos volver a revisar nuestra configuración en los grupos de seguridad, recordemos que al ser una solicitud HTTPS esta se encuentra en el puerto 443, por lo que deberá estar dentro de las reglas.

Reglas de entrada (4)	Administrar etiquetas	Editar reglas de entrada					
Nombre	ID de la regla del gr...	Versión de IP	Tipo	Protocolo	Intervalo de puertos	Origen	Descripción
-	sgr-0e8bb400d47ae6eee	IPv4	TCP personalizado	TCP	8000	0.0.0.0/0	-
-	sgr-01b569640f44db769	IPv4	HTTP	TCP	80	0.0.0.0/0	-
-	sgr-099d6f958f0d5fb11	IPv4	HTTPS	TCP	443	0.0.0.0/0	-
-	sgr-0907b50ab9f4bdb06	IPv4	SSH	TCP	22	0.0.0.0/0	-

Si estas están correctamente configuradas podrás ver tu app corriendo con certificación https en tu dominio personalizado.



¿Interesado en el siguiente paso?

Esta documentación contiene el paso a paso para desplegar en docker, haciendo uso de herramientas simples con un proyecto estático, sin embargo aún hay mucho margen de mejora, por lo que en el siguiente escrito, veremos cómo subir un proyecto real que contenga:

1. BackEnd
2. FrontEnd
3. Base de datos

MUCHAS GRACIAS POR LEER