

**Ubuntu**

Server

2025

**Índice**

VirtualBox …………………………………………………………………… 3

Servidor ……………………………………………………………………... 4

IP local del servidor ……………………………………………………….. 5

IP pública (acceso desde internet) ……………………………………… 6

IP dinámica (DNS) ………………………………………………………… 9

Configuración del Dominio en Nginx ……………………………………. 13

Certificado HTTPS con Certbot …………………………………………. 14

Revisiones para React Router …………………………………………... 15

Multiples servidores ……………………………………………………….. 16

Base de Datos ……………………………………………………………… 17

Backend …………………………………………………………………….. 19

Inicio automatizado ……………………………………………………….. 22

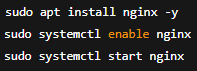
**Virtual Box**

**Servidor**

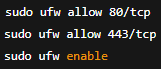
Teniendo la distro instalada, procedemos a actualizar



Teniendo el sistema actualizado, se procede a instalar **Nginx**  /engine x/



Comprobar que funciona

A continuación, configurar el firewall

O bien hacerlo manualmente

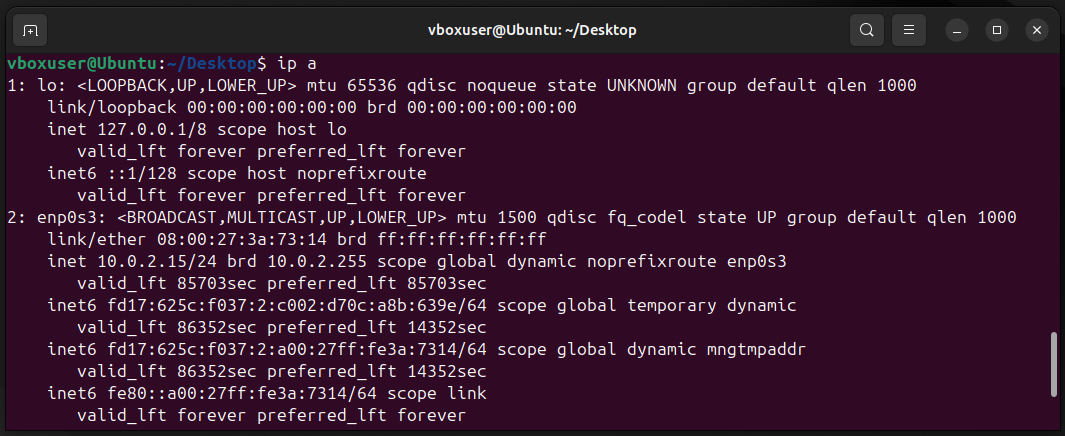
Ahora ya se pueden colocar los archivos de nuestra web para que se muestren en la página web en el directorio de Nginx

(por defecto, Nginx agrega un index.html de ejemplo)



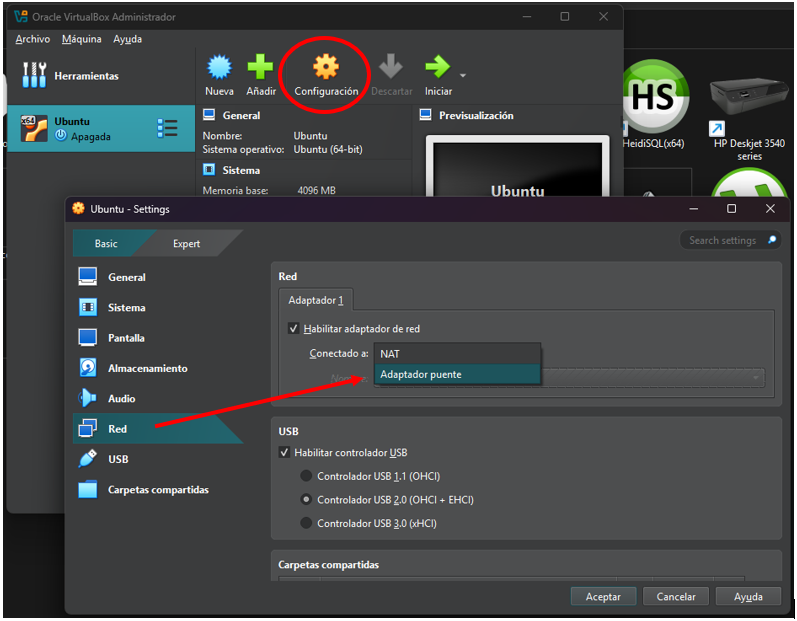
IP local del servidor

Para acceder al servidor ya iniciado (started), es necesario conocer nuestra IP



Esta es la IP local del VM

Es necesario saber que, si estamos dentro de una VM, la IP no se podrá encontrar dentro del LAN. Para ello es necesario cambiar el tipo de Red en la configuración de nuestra VM.



Hecho lo anterior, en la consola nos aparecerá nuestra IP local real

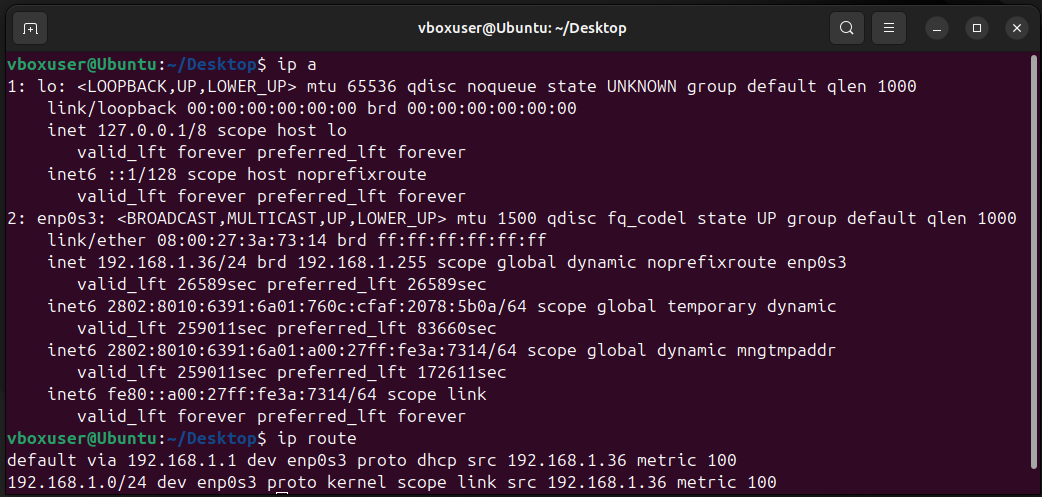
Así, ya podrás acceder al servidor con **http://192.168.1.36**

IP pública y acceso desde internet

* Fijar IP

Es necesario saber que, para que sea accesible desde internet, se requiere que la IP local del servidor se mantenga fija para que siempre sea la misma dirección de acceso.

1. Conocer los datos de nuestra red actual



Nombre de la interfaz (enp0s3 o eth0)

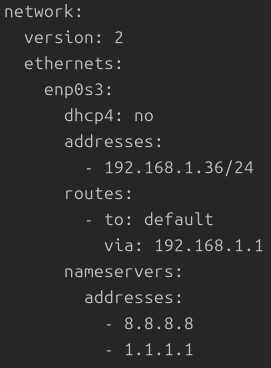
IP actual: 192.168.1.36 (en este caso)

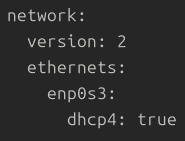
Gateway (suele terminar en .1): 192.168.1.1 (en este caso)

Servidores DNS: se pueden usar los del router o públicos (8.8.8.8, 1.1.1.1)

1. Editar la configuración de Netplan para cambiar de DHCP a IP fija

Abrí el archivo de configuración (puede llamarse 00-installer-config.yaml o parecido) en **/etc/netplan/**

 El archivo debe quedar de la siguiente manera:



Los datos se guardan con **Ctrl + O** , **Enter** y luego **Ctrl + X**

Para aplicar los cambios se ejecuta:

Es posible que, al aplicar los cambios, salga algún warning como este:

Permissions for /etc/netplan/01-network-manager-all.yaml are too open

Para solucionarlo, es necesario ejecutar el siguiente comando (para restringir el acceso al archivo y mejorar la seguridad).



* Configurar las reglas de puertos del router de internet

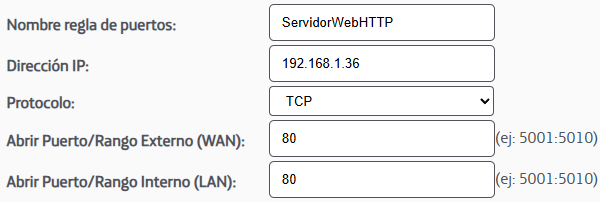
En general, se puede ingresar a dicha configuración con el siguiente link

<http://192.168.1.1> (en caso de no corresponder, revisar cuál es la dirección específica del router)

Crear reglas como

|  |  |  |
| --- | --- | --- |
| Puerto externo | Puerto interno | IP destino |
| 80 (TCP) | 80 (TCP) | 192.168.1.36 (en este caso) |
| 443 (TCP) | 443 (TCP) | 192.168.1.36 (en este caso) |

Ejemplo de Movistar:



* Configurar la IP pública

A continuación, instalar Curl

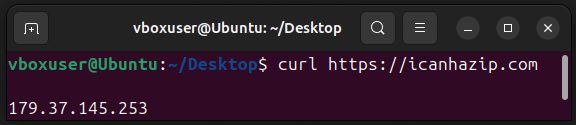
y utilizarlo para obtener la IP pública con alguno de los siguientes servicios



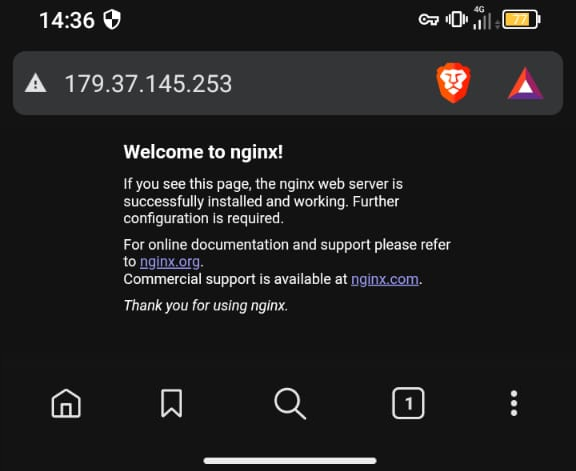




Encontrando el que funcione (u otro), obtendrás una IP



Teniendo esa IP y todo lo anterior configurado, se podrá acceder desde internet a través de esa dirección.



* Medidas de seguridad

Como el servidor ahora está expuesto a internet, es necesario tomar medidas de seguridad.

Es recomendable instalar fail2ban

IP dinámica con ***Domain Name System*** (DNS)

En general, los proveedores de internet asignan IPs dinámicas. La IP pública cambiará con el tiempo y, por lo tanto, ya no se podrá acceder desde esa misma dirección previamente obtenida con curl.

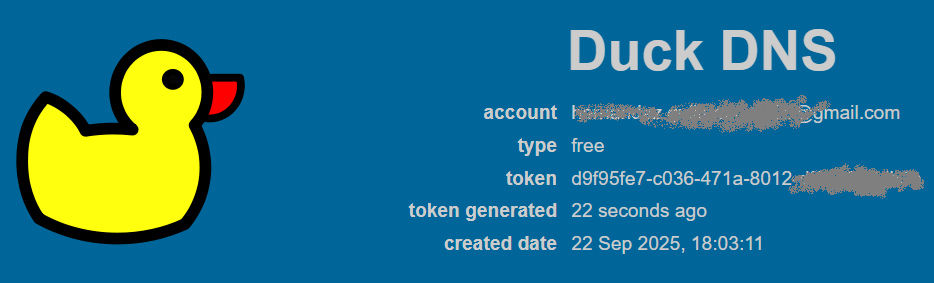
Para evitar esto, la mejor solución es tener un **Dominio** y crear un script dentro del VM (Linux) que actualice la dirección IP a la que apunta el Dominio cada cierto tiempo (ejemplo: cada 5 minutos).

* DuckDNS

La forma más práctica de mantener tu servidor accesible desde Internet es usando el servicio de DuckDNS (<https://www.duckdns.org/>) el cual permite crear subdominios gratuitos con sólo registrar tu cuenta de Gmail o Github.



Tras agregar el dominio (*en este caso guillenh*), la página te brindará un token.



Dicho token servirá para actualizar la IP pública a la cual apunta el dominio.

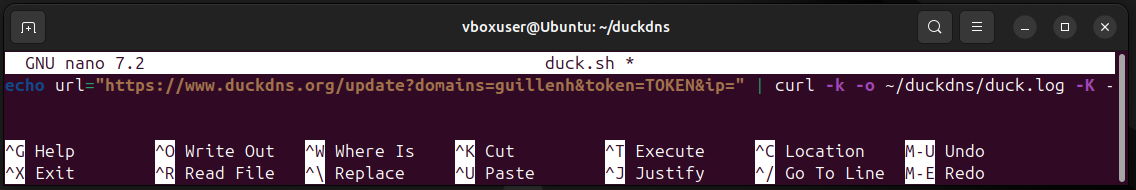
Para hacer el script de actualización automática, crearemos el script necesario.



Una vez creada la carpeta, creamos el script



Una vez dentro de dicho script, copiamos el siguiente código en un solo renglón.



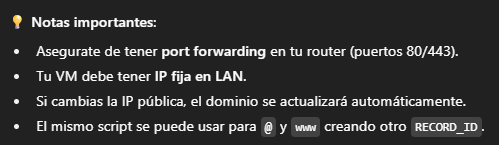
Guardar (Ctrl+O, Enter) y salir (Ctrl+X).

Aquí va el token

Teniendo el código del script, ahora haremos que se ejecute cada cierto tiempo con Crontab.

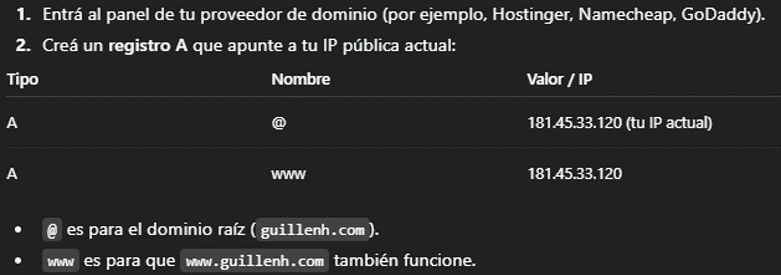


Agregar la siguiente línea (al final del archivo)

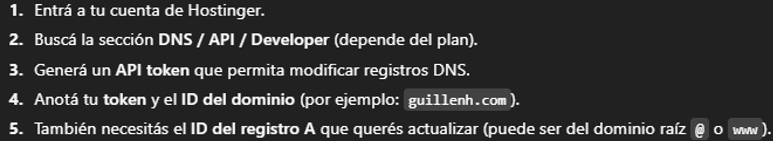


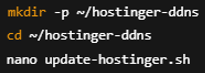
* Dominio propio

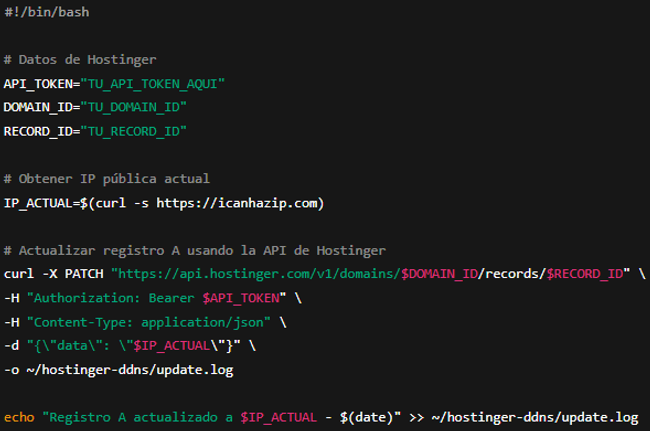
Teniendo un dominio propio (por ejemplo: Hostinger)



Obtener el token / API de Hostinger



A continuación, crear el script de actualización automática:

Y copiar el siguiente código:

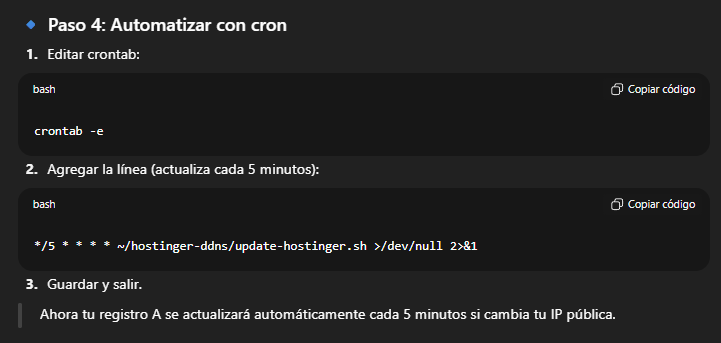
Ctrl + O (guardar)

Enter

Ctrl + X (cerrar)

Para hacer el ejecutable:

Es importante automatizar el proceso de actualizar automáticamente la IP pública a la que apunta el dominio cada vez que cambie.



Configuracion del dominio en Nginx

Es importante conocer los archivos de configuración de nuestra página web, que se encuentran ubicados en:

Debemos crear un archivo dentro de **sites-available**



Dentro del archivo, escribir la configuración

Puerto donde está ubicada la página

Dominio de la página

Ubicación de los archivos de la página

En caso de entrar en una uri desconocida, lo reenviará al index

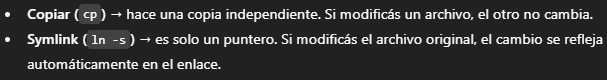
Configuración de una uri dedicada a un backend junto con el proxy necesario para poder conectar una Base de Datos, ubicada en el port 3000

Hecha la configuración, realizar un *symlink* para el **sites-enabled**



A continuación, probar que todo funcione correctamente

* Testear:
* Reinciar:



Certificado HTTPS

Se puede solicitar el certificado SSL mediante Certbot.



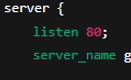
Una vez instalado, se hace la solicitud del certificado.

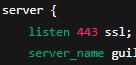


*(donde “midominio” en este caso es “guillenh”)*

Certbot hará preguntas sobre correo electrónico, aceptar términos y si se desea redirigir todo HTTP a HTTPS (recomendable que sí)

Certbot instala un cron job automáticamente para renovar el certificado cada 60 días. Aunque se puede hacer manual con el siguiente comando.

**Nota importante**

Nginx fue configurado en el puerto **80**

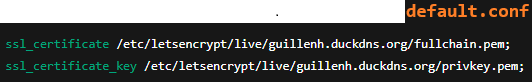
Sin embargo, Certbot configura automáticamente el puerto **443** para permitir el uso de HTTPS, y lo hace en un archivo diferente.

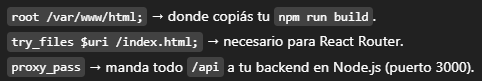
Es decir, el sitio web está configurado en ***/etc/nginx/sites-available/guillenh.conf***

El archivo de configuración hecho por Certbot puede estar ubicado en el mismo directorio, pero con un nombre diferente, por ejemplo: ***guillenh-le-ssl.conf***

O simplemente en un archivo ***default.conf***

Es importante saberlo ya que, si se necesita configurar un backend o corregir rutas para React Route (como se verá más adelante), se debe hacer aquí también.

Este es el código autogenerado por Certbot:

Revisiones para React Router

Cuando se trabaja con **React Router**, es necesario tener algunas consideraciones sobre Nginx para que funcione correctamente. Caso contrario, la página devolverá un **404 Not found** cuando se refresque una página distinta de **/**.

Para evitarlo, es necesario cambiar la configuración del servidor como se muestra a continuación:

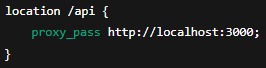


Es importante recordar que los archivos de la página web que se mostrarán, deben ser los creados con el comando **build** de React, es decir, los archivos que se encuentran dentro de la carpeta build (o dist).

Dichos archivos se deben copiar dentro del directorio de Nginx.

****

Si en la página de React se realiza algún **fetch** (petición a un backend con acceso a una base de datos), es recomendable que sean a una ruta como *api* y configurar un **reverse proxy** en Nginx a su respectivo puerto local.



**Multiples servidores**

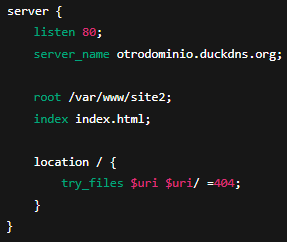
Si bien existe una sola IP pública, Nginx distingue a qué sitio web dirigir una petición usando el nombre de dominio.

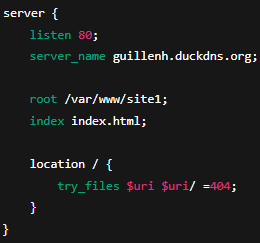
Para empezar, se deben crear distintas carpetas para cada sitio web con sus respectivos permisos.



En cada carpeta irán los respectivos contenidos de cada página web.

Luego, crear los archivos de configuración para cada sitio web.





Activar los sitios:



Finalmente activar el HTTPS con Certbot:

(no olvidar que esto genera nuevos bloques **server**   
en un archivo .***conf*** diferente que deben revisarse)

**Base de Datos**

Crearemos una Base de Datos dentro de la distro (Ubuntu) que sólo escuche en **localhost:3000** (importante para seguridad, nunca debe estar expuesta a internet).



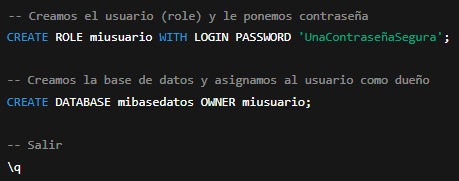
Verificar que funcione

o

Entrar a la cuenta administrativa de Postgres



desde allí crear un usuario y una base de datos



Probar la conexión desde la misma máquina (local)

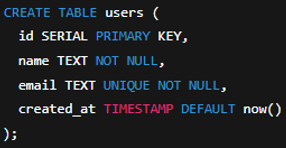


si se conecta, se mostrará **mibasedatos=>**

Se puede reiniciar PostgreSQL (sugerido después de realizar cambios) con el siguiente comando:

Además, cada cierto tiempo es recomendable hacer copias de seguridad.

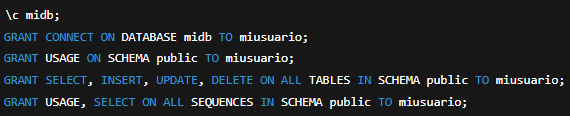
Ya se pueden crear tablas (desde dentro de **psql**)



y pueden agregar filas a tablas creadas con los siguientes comandos:

* Seleccionar la base de datos
* Insertar datos en la tabla “*users*”
* Visualizar los datos
* Salir

Puede ser necesario otorgar permisos al usuario (role) creado



**Backend**

Un “Backend” puede ser una API alojada dentro de la máquina, que será accesible desde un “Frontend” (sitio web visible para el usuario).

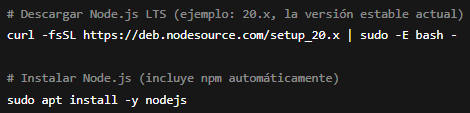
Se alojan en directorios diferentes ya que cada uno se ejecuta por sí mismo.

|  |  |
| --- | --- |
| **Frontend** | **Backend** |
| Servido desde Nginx (u otro)  Archivos estáticos (HTML, CSS y JS)  ***/var/www/html/*** | Servido desde Node.js (u otro)  Se ejecuta como aplicación  ***/var/www/backend/*** |

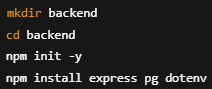
los directorios son sugeridos y pueden variar

Existen 2 opciones para un backend.

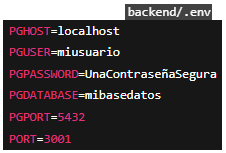
1. Alojar el backend de forma local para que sólo el frontend pueda accederlo
2. Alojar el backend públicamente para que se puedan realizar peticiones desde cualquier parte (por ejemplo: otras aplicaciones y otros sitios web)

**Node.js**

Lo primero será instalar Node.js y crear un proyecto en un directorio.



Se utilizará **express** y **pg** para hacer el puente con la Base de Datos.

También se utiliza **dotenv** para proteger los datos que se requieren para acceder a la base de datos y, para ello, se debe crear un archivo ***.env*** dentro del directorio del backend.



Teniendo el código ya hecho, el servidor puede ejecutarse:

(es necesario estar ubicado dentro del directorio backend)

Para este caso, usaremos un backend de ejemplo para obtener los datos de la base de datos creada anteriormente:

require("dotenv").config(); *// permite leer los datos del .env*

*const* express = require("express"); *// permite manejar las peticiones*

*const* { Pool } = require("pg"); *// conector con la base de datos*

*const* app = express();

app.use(express.json()); *// para manejar JSON en requests*

*// Crear un pool de conexiones*

*const* pool = new Pool({

  host: process.env.PGHOST,

  user: process.env.PGUSER,

  password: process.env.PGPASSWORD,

  database: process.env.PGDATABASE,

  port: process.env.PGPORT,

});

*// Endpoint de prueba*

app.get("/", (*req*, *res*) => {

  res.send("Servidor backend funcionando!");

});

*// Endpoint para obtener todos los usuarios*

app.get("/api/users", *async* (*req*, *res*) => {

  try {

*const* result = await pool.query("SELECT \* FROM users ORDER BY id");

    res.json(result.rows); *// devuelve un array JSON*

  } catch (err) {

    console.error(err);

    res.status(500).json({ error: "Error al consultar la base de datos" });

  }

});

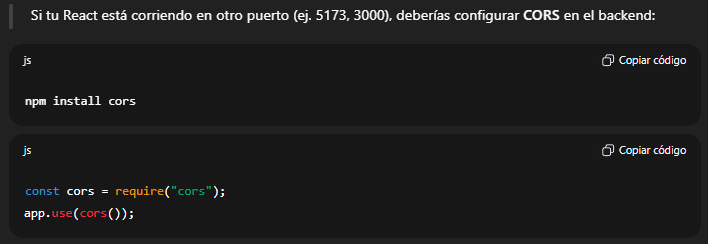
*// Iniciar servidor*

*const* PORT = process.env.PORT || 3001;

app.listen(PORT, () => {

  console.log(`Servidor escuchando en http://localhost:${PORT}`);

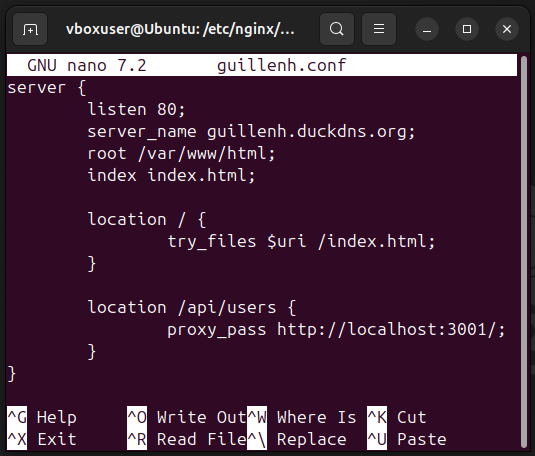
});

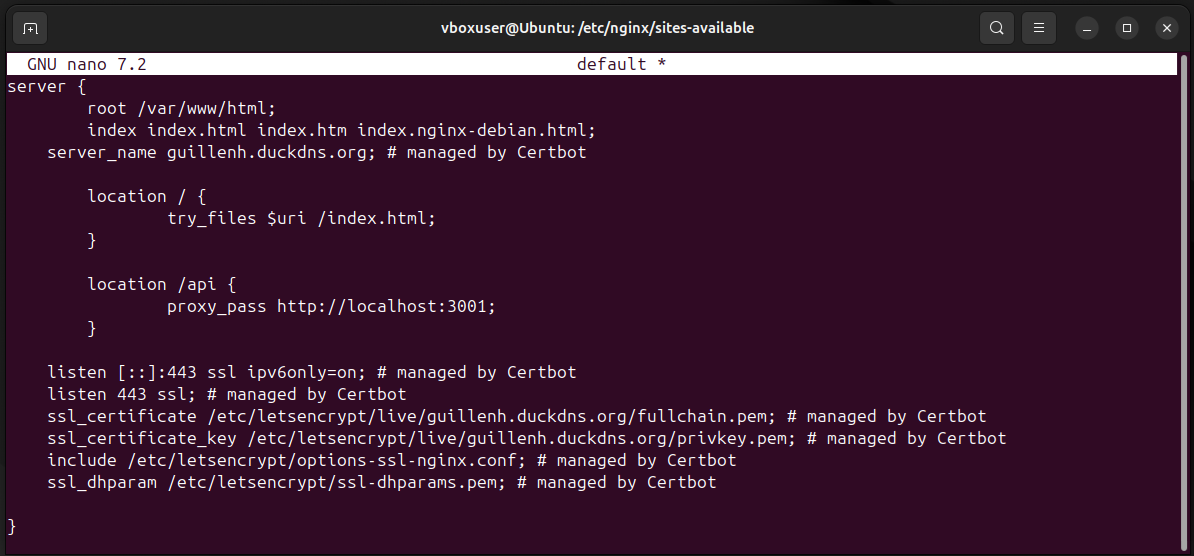


Junto con un pequeño ejemplo de index.html para nuestro frontend, podemos comprobar que todo está funcionando correctamente.



Con la configuración de Nginx de la siguiente manera:



Sin olvidar modificar el **default.conf** (o nombre similar) que nos deja Certbot al implementar HTTPS:

Hay más contenido en el archivo, pero modificamos únicamente el bloque de nuestro server

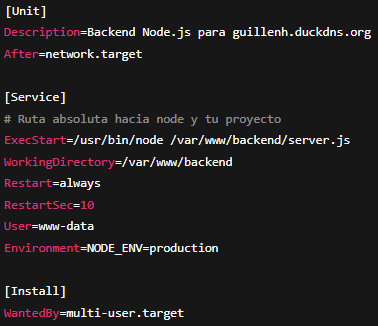
Inicio automatizado

En Linux es posible establecer el backend como un *servicio* con **systemd**, lo cual permite que quede siempre activo y no sea necesario ejecutarlo cada vez.

Para ello, es necesario crear un archivo .service



Cuyo contenido sea



ubicación del backend

Luego, recargar el systemd y habilitar el servicio



Verificar que esté funcionando



Active: **active** (running)

debe devolver

Se pueden comprobar logs mediante

Y se puede detener el servicio mediante