



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

```
sudo apt update && sudo apt upgrade -y # En Debian/Ubuntu
```

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

```
sudo apt install nginx -y
sudo systemctl enable nginx
sudo systemctl start nginx
```

Comprobar que funciona

```
http://<IP-de-tu-servidor>
```

A continuación, configurar el firewall

O bien hacerlo manualmente

```
sudo ufw allow 'Nginx Full'
```

```
sudo ufw allow 80
```

```
sudo ufw allow 443
```

```
sudo ufw allow 80/tcp
```

```
sudo ufw allow 443/tcp
```

```
sudo ufw enable
```

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)

```
/var/www/html
```





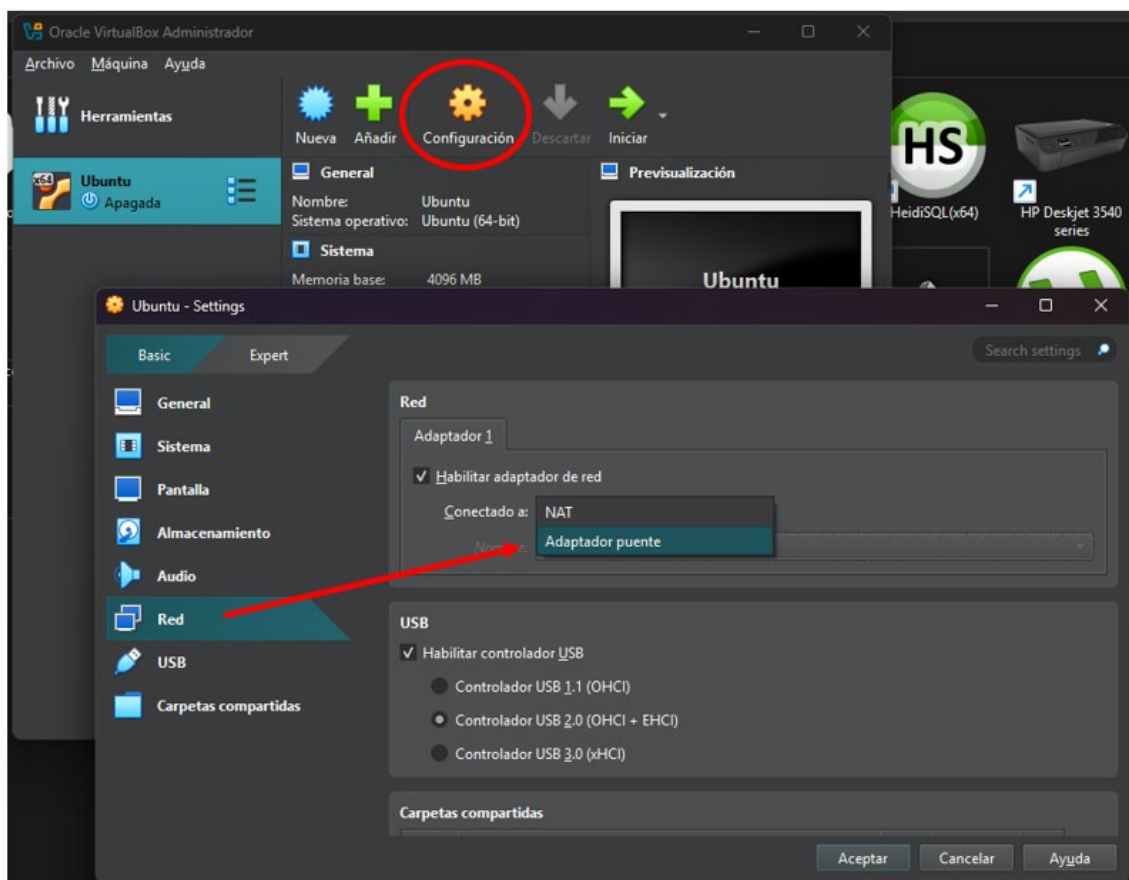
IP local del servidor

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

```
vboxuser@Ubuntu: ~/Desktop
vboxuser@Ubuntu:~/Desktop$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:3a:73:14 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
        valid_lft 85703sec preferred_lft 85703sec
    inet6 fd17:625c:f037:2:c002:d70c:a8b:639e/64 scope global temporary dynamic
        valid_lft 86352sec preferred_lft 14352sec
    inet6 fd17:625c:f037:2:a00:27ff:fe3a:7314/64 scope global dynamic mngtmpaddr
        valid_lft 86352sec preferred_lft 14352sec
    inet6 fe80::a00:27ff:fe3a:7314/64 scope link
        valid_lft forever preferred_lft forever
```

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**

```
link/ether 08:00:27:3a:73:14
inet 192.168.1.36/24
```



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

```
vboxuser@Ubuntu: ~/Desktop
vboxuser@Ubuntu:~/Desktop$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
   link/ether 08:00:27:3a:73:14 brd ff:ff:ff:ff:ff:ff
   inet 192.168.1.36/24 brd 192.168.1.255 scope global dynamic noprefixroute enp0s3
       valid_lft 26589sec preferred_lft 26589sec
   inet6 2802:8010:6391:6a01:760c:cfaf:2078:5b0a/64 scope global temporary dynamic
       valid_lft 259011sec preferred_lft 83660sec
   inet6 2802:8010:6391:6a01:a00:27ff:fe3a:7314/64 scope global dynamic mngtmpaddr
       valid_lft 259011sec preferred_lft 172611sec
   inet6 fe80::a00:27ff:fe3a:7314/64 scope link
       valid_lft forever preferred_lft forever
vboxuser@Ubuntu:~/Desktop$ ip route
default via 192.168.1.1 dev enp0s3 proto dhcp src 192.168.1.36 metric 100
192.168.1.0/24 dev enp0s3 proto kernel scope link src 192.168.1.36 metric 100
```

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)

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

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

El archivo debe quedar de la siguiente manera:

```
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: true
```

```
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: no
      addresses:
        - 192.168.1.36/24
      routes:
        - to: default
          via: 192.168.1.1
      nameservers:
        addresses:
          - 8.8.8.8
          - 1.1.1.1
```

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

Para aplicar los cambios se ejecuta:

```
sudo netplan apply
```



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).

```
sudo chmod 600 /etc/netplan/01-network-manager-all.yaml
```

- 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:

Nombre regla de puertos:	<input type="text" value="ServidorWebHTTP"/>
Dirección IP:	<input type="text" value="192.168.1.36"/>
Protocolo:	<input type="text" value="TCP"/>
Abrir Puerto/Rango Externo (WAN):	<input type="text" value="80"/> (ej: 5001:5010)
Abrir Puerto/Rango Interno (LAN):	<input type="text" value="80"/> (ej: 5001:5010)



- Configurar la IP pública

A continuación, instalar Curl `sudo apt install curl`

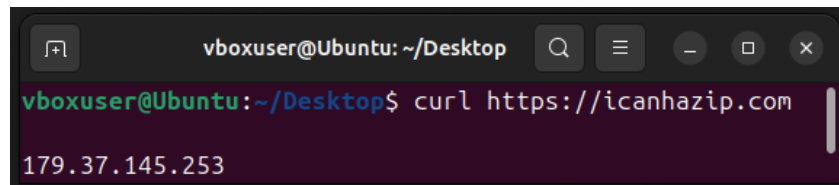
y utilizarlo para obtener la IP pública con alguno de los siguientes servicios

```
curl ifconfig.me
```

```
curl https://ipinfo.io/ip
```

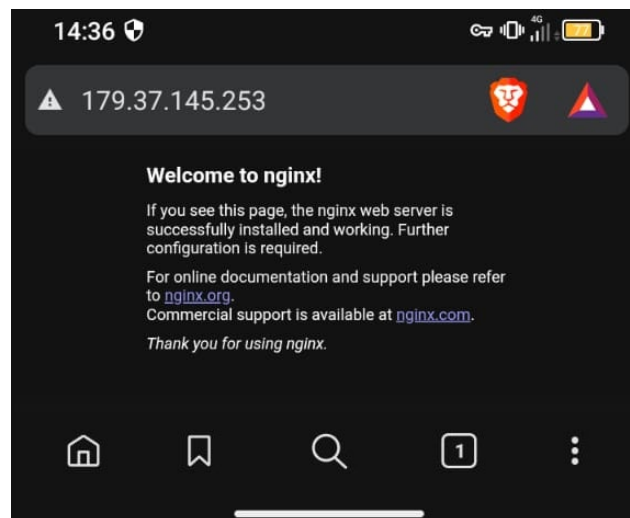
```
curl https://icanhazip.com
```

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



```
vboxuser@Ubuntu: ~/Desktop
vboxuser@Ubuntu:~/Desktop$ curl https://icanhazip.com
179.37.145.253
```

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`



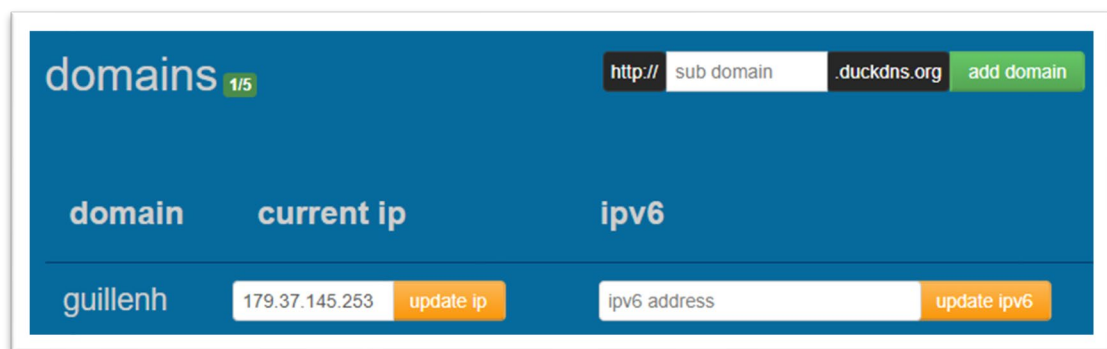
IP dinámica con **Domain Name System** (DNS)

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.

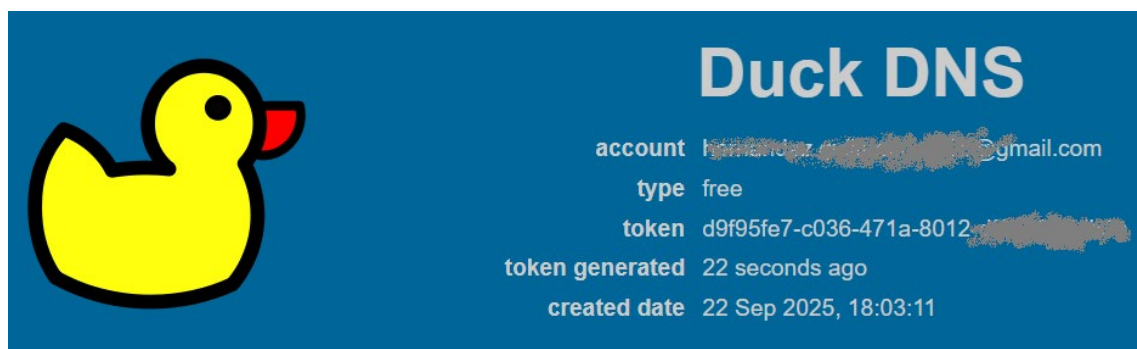
Para evitar esto, la mejor solución es tener un **Dominio** y crear un script dentro del VM (Linux) que actualice la dirección IP a la que apunta el Dominio cada cierto tiempo (ejemplo: cada 5 minutos).

- 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.

```
mkdir -p ~/duckdns
cd ~/duckdns
```

Una vez creada la carpeta, creamos el script

```
nano duck.sh
```

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

```
vboxuser@Ubuntu: ~/duckdns
GNU nano 7.2 duck.sh *
echo url="https://www.duckdns.org/update?domains=guillenh&token=TOKEN&ip=" | curl -k -o ~/duckdns/duck.log -K -
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line  M-E Redo
```

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.

```
crontab -e
```

Agregar la siguiente línea (al final del archivo)

```
*/5 * * * * ~/duckdns/duck.sh >/dev/null 2>&1
```

Notas importantes:

- Asegurate de tener **port forwarding** en tu router (puertos 80/443).
- Tu VM debe tener IP fija en LAN.
- Si cambias la IP pública, el dominio se actualizará automáticamente.
- El mismo script se puede usar para @ y www creando otro `RECORD_ID`.



- Dominio propio

Teniendo un dominio propio (por ejemplo: Hostinger)

1. Entrá al panel de tu proveedor de dominio (por ejemplo, Hostinger, Namecheap, GoDaddy).
2. Creá un **registro A** que apunte a tu IP pública actual:

Tipo	Nombre	Valor / IP
A	@	181.45.33.120 (tu IP actual)
A	www	181.45.33.120

- @ es para el dominio raíz (guillenh.com).
- www es para que www.guillenh.com también funcione.

Obtener el token / API de Hostinger

1. Entrá a tu cuenta de Hostinger.
2. Buscá la sección DNS / API / Developer (depende del plan).
3. Generá un API token que permita modificar registros DNS.
4. Anotá tu token y el ID del dominio (por ejemplo: guillenh.com).
5. También necesitás el ID del registro A que querés actualizar (puede ser del dominio raíz @ o www).

A continuación, crear el script de actualización automática:

Y copiar el siguiente código:

```
mkdir -p ~/hostinger-ddns
cd ~/hostinger-ddns
nano update-hostinger.sh
```

```
#!/bin/bash

# Datos de Hostinger
API_TOKEN="TU_API_TOKEN_AQUI"
DOMAIN_ID="TU_DOMAIN_ID"
RECORD_ID="TU_RECORD_ID"

# Obtener IP pública actual
IP_ACTUAL=$(curl -s https://icanhazip.com)

# Actualizar registro A usando la API de Hostinger
curl -X PATCH "https://api.hostinger.com/v1/domains/$DOMAIN_ID/records/$RECORD_ID" \
-H "Authorization: Bearer $API_TOKEN" \
-H "Content-Type: application/json" \
-d '{"data": {"IP_ACTUAL": "$IP_ACTUAL"}}' \
-o ~/hostinger-ddns/update.log

echo "Registro A actualizado a $IP_ACTUAL - $(date)" >> ~/hostinger-ddns/update.log
```

Ctrl + O (guardar)

Enter

Ctrl + X (cerrar)

Para hacer el ejecutable:

```
chmod +x update-hostinger.sh
```



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

◆ Paso 4: Automatizar con cron

1. Editar crontab:

```
bash
```

[Copiar código](#)

```
crontab -e
```

2. Agregar la línea (actualiza cada 5 minutos):

```
bash
```

[Copiar código](#)

```
*/5 * * * * ~/hostinger-ddns/update-hostinger.sh >/dev/null 2>&1
```

3. Guardar y salir.

Ahora tu registro A se actualizará automáticamente cada 5 minutos si cambia tu IP pública.



Configuración del dominio en Nginx

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

```
/etc/nginx/sites-available/  
/etc/nginx/sites-enabled/
```

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

```
sudo nano /etc/nginx/sites-available/guillenh.conf
```

Dentro del archivo, escribir la configuración

```
server {  
    listen 80;  
    server_name guillenh.duckdns.org;  
  
    root /var/www/html;  
    index index.html;  
  
    location / {  
        try_files $uri /index.html;  
    }  
  
    location /api {  
        proxy_pass http://localhost:3000;  
        proxy_http_version 1.1;  
        proxy_set_header Upgrade $http_upgrade;  
        proxy_set_header Connection 'upgrade';  
        proxy_set_header Host $host;  
        proxy_cache_bypass $http_upgrade;  
    }  
}
```

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**

```
sudo ln -s /etc/nginx/sites-available/guillenh.conf /etc/nginx/sites-enabled/
```

A continuación, probar que todo funcione correctamente

- Testear: `sudo nginx -t`
- Reiniciar: `sudo systemctl restart nginx`

- Copiar (`cp`) → hace una copia independiente. Si modificás un archivo, el otro no cambia.
- Symlink (`ln -s`) → es solo un puntero. Si modificás el archivo original, el cambio se refleja automáticamente en el enlace.



Certificado HTTPS

Se puede solicitar el certificado SSL mediante Certbot.

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

Una vez instalado, se hace la solicitud del certificado.

```
sudo certbot --nginx -d midominio.duckdns.org
```

(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.

```
sudo certbot renew --dry-run
```

Nota importante

Nginx fue configurado en el puerto **80**

```
server {
    listen 80;
    server_name g
```

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

```
server {
    listen 443 ssl;
    server_name guil
```

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:

default.conf

```
ssl_certificate /etc/letsencrypt/live/guillenh.duckdns.org/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/guillenh.duckdns.org/privkey.pem;
```



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:

```
server {  
    listen 80;  
    server_name guillenh.duckdns.org;  
  
    root /var/www/html;  
    index index.html;  
  
    location / {  
        try_files $uri /index.html;  
    }  
}
```



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.

```
sudo cp -r build/* /var/www/html/
```

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.

```
location /api {  
    proxy_pass http://localhost:3000;  
}
```

```
root /var/www/html; → donde copiás tu npm run build.  
try_files $uri /index.html; → necesario para React Router.  
proxy_pass → manda todo /api a tu backend en Node.js (puerto 3000).
```



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.

```
sudo mkdir -p /var/www/site1
sudo mkdir -p /var/www/site2
```

```
sudo chown -R $USER:$USER /var/www/site1
sudo chown -R $USER:$USER /var/www/site2
```

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

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

```
sudo nano /etc/nginx/sites-available/site1
```

```
server {
    listen 80;
    server_name guillenh.duckdns.org;

    root /var/www/site1;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

```
sudo nano /etc/nginx/sites-available/site2
```

```
server {
    listen 80;
    server_name otrodominio.duckdns.org;

    root /var/www/site2;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

Activar los sitios:

```
sudo ln -s /etc/nginx/sites-available/site1 /etc/nginx/sites-enabled/
sudo ln -s /etc/nginx/sites-available/site2 /etc/nginx/sites-enabled/
```

Finalmente activar el HTTPS con Certbot:

```
sudo certbot --nginx -d guillenh.duckdns.org
sudo certbot --nginx -d otrodominio.duckdns.org
```

(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).

```
sudo apt install postgresql postgresql-contrib -y
```

Verificar que funcione

```
sudo systemctl status postgresql
```

o

```
psql --version
```

Entrar a la cuenta administrativa de Postgres

```
sudo -u postgres psql
```

desde allí crear un usuario y una base de datos

```
-- Creamos el usuario (role) y le ponemos contraseña
CREATE ROLE miusuario WITH LOGIN PASSWORD 'UnaContraseñaSegura';

-- Creamos la base de datos y asignamos al usuario como dueño
CREATE DATABASE mibasedatos OWNER miusuario;

-- Salir
\q
```

Probar la conexión desde la misma máquina (local)

```
psql -h localhost -U miusuario -d mibasedatos -W
```

si se conecta, se mostrará mibasedatos=>

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

```
sudo systemctl restart postgresql
```

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

```
pg_dump -U miusuario -h localhost mibasedatos > backup.sql
```



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

```
sudo -u postgres psql
```

```
\c mibasedatos;
```

```
CREATE TABLE users (  
  id SERIAL PRIMARY KEY,  
  name TEXT NOT NULL,  
  email TEXT UNIQUE NOT NULL,  
  created_at TIMESTAMP DEFAULT now()  
);
```

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

- Seleccionar la base de datos

```
\c mibasedatos;
```

- Insertar datos en la tabla "users"

```
INSERT INTO users (name, email)  
VALUES ('Juan Pérez', 'juan@example.com');
```

- Visualizar los datos

```
SELECT * FROM users;
```

- Salir

```
\q
```

Puede ser necesario otorgar permisos al usuario (role) creado

```
\c midb;  
GRANT CONNECT ON DATABASE midb TO miusuario;  
GRANT USAGE ON SCHEMA public TO miusuario;  
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO miusuario;  
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO miusuario;
```



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.

```
# Descargar Node.js LTS (ejemplo: 20.x, la versión estable actual)
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -

# Instalar Node.js (incluye npm automáticamente)
sudo apt install -y nodejs
```

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.

```
mkdir backend
cd backend
npm init -y
npm install express pg dotenv
```

```
sudo nano .env
```

```
backend/.env
PGHOST=localhost
PGUSER=miusuario
PGPASSWORD=UnaContraseñaSegura
PGDATABASE=mibasedatos
PGPORT=5432
PORT=3001
```

Teniendo el código ya hecho, el servidor puede ejecutarse:
(es necesario estar ubicado dentro del directorio backend)

```
node server.js
```



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

```
/var/www/backend/
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}`);
});
```

Si tu React está corriendo en otro puerto (ej. 5173, 3000), deberías configurar **CORS** en el backend:

```
js Copiar código
```

```
npm install cors
```

```
js Copiar código
```

```
const cors = require("cors");
app.use(cors());
```



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

```
index.html
/var/www/html/

<!DOCTYPE html>
<html Lang="es">
  <head>
    <meta charset="UTF-8">
    <title>Frontend Simple</title>
  </head>
  <body>
    <h1>Frontend</h1>
    <button id="btn">Consultar Backend</button>
    <p id="p"></p>

    <script>
      document.getElementById("btn").addEventListener("click", async () => {
        try {
          const res = await fetch("/api/users"); // se conecta al backend
          const data = await res.json();
          console.log(data); // se mostrará la respuesta en la consola
          document.getElementById("p").textContent = "Consulta exitosa";
        } catch (error) { document.getElementById("p").textContent = "Error: " + error; }
      });
    </script>
  </body>
</html>
```

Con la configuración de Nginx de la siguiente manera:

```
guillenh.conf
/etc/nginx/sites-available/
/etc/nginx/sites-enabled/

server {
    listen 80;
    server_name guillenh.duckdns.org;
    root /var/www/html;
    index index.html;

    location / {
        try_files $uri /index.html;
    }

    location /api/users {
        proxy_pass http://localhost:3001/;
    }
}
```

Sin olvidar modificar el **default.conf** (o nombre similar) que nos deja Certbot al implementar HTTPS: `vboxuser@Ubuntu:/etc/nginx/sites-available$ sudo nano default`

```
default *
GNU nano 7.2

server {
    root /var/www/html;
    index index.html index.htm index.nginx-debian.html;
    server_name guillenh.duckdns.org; # managed by Certbot

    location / {
        try_files $uri /index.html;
    }

    location /api {
        proxy_pass http://localhost:3001;
    }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/guillenh.duckdns.org/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/guillenh.duckdns.org/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}
```

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`

```
sudo nano /etc/systemd/system/backend.service
```

Cuyo contenido sea

```
[Unit]
Description=Backend Node.js para guillenh.duckdns.org
After=network.target

[Service]
# Ruta absoluta hacia node y tu proyecto
ExecStart=/usr/bin/node /var/www/backend/server.js
WorkingDirectory=/var/www/backend
Restart=always
RestartSec=10
User=www-data
Environment=NODE_ENV=production

[Install]
WantedBy=multi-user.target
```

ubicación del backend

Luego, recargar el systemd y habilitar el servicio

```
sudo systemctl daemon-reload
sudo systemctl enable backend
sudo systemctl start backend
```

Verificar que esté funcionando

```
sudo systemctl status backend
```

debe devolver

Active: **active** (running)

Se pueden comprobar logs mediante

```
journalctl -u backend -f
```

Y se puede detener el servicio mediante

```
sudo systemctl stop backend
```