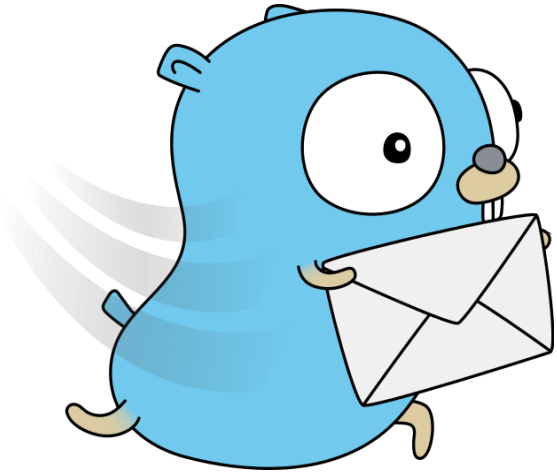
LABORATORIO DE REDES Y SISTEMAS OPERATIVOS

Trabajo Final
Instalación, compilación y configuración de un servidor de notificaciones Gotify



Profesor:
José Luis Di Biase

Integrantes:
Frank Matias Flores Rea
Scheffelaar Klotz German

Universidad Nacional de Quilmes

Índice

1. Descripción del proyecto

El objetivo de este trabajo consistió en instalar, compilar y configurar un servidor de notificaciones autoalojado utilizando Gotify, una aplicación de software libre desarrollada principalmente en el lenguaje Go. Gotify permite enviar, almacenar y recibir notificaciones mediante una API REST, una interfaz web y diferentes clientes compatibles.

El servidor fue instalado en una máquina virtual con Ubuntu, ejecutada sobre Oracle VirtualBox. Como parte del proyecto, se descargaron y compilaron desde sus respectivos repositorios oficiales el servidor Gotify, el cliente de línea de comandos Gotify CLI y el servidor web Caddy.

Posteriormente, Caddy se configuró como proxy inverso, recibiendo las conexiones HTTP a través del puerto 80 y redirigiéndolas al servidor Gotify, que se ejecuta internamente sobre el puerto 8080. Tanto Gotify como Caddy fueron configurados como servicios administrados mediante systemd, permitiendo su inicio automático junto con el sistema operativo.

Asimismo, se implementó un mecanismo para optimizar automáticamente las imágenes utilizadas como logotipos de las aplicaciones creadas en Gotify. Para ello se emplearon las herramientas ImageMagick y OptiPNG, junto con un script desarrollado en Bash, cuya ejecución fue automatizada mediante una tarea programada con Cron.

Finalmente, el correcto funcionamiento del sistema fue verificado mediante distintos mecanismos de prueba: la interfaz web de Gotify, solicitudes a la API REST utilizando curl, el cliente de línea de comandos Gotify CLI y la aplicación oficial para Android ejecutada en un emulador de Android Studio.

2. Relevamiento del entorno utilizado

2.1. Características del equipo anfitrión

El desarrollo del proyecto se realizó sobre una computadora portátil Dell Latitude 7320, utilizando Windows 11 Enterprise de 64 bits como sistema operativo anfitrión. Sobre este equipo se instaló Oracle VirtualBox, herramienta utilizada para crear y administrar la máquina virtual donde se llevó a cabo la instalación, compilación y configuración del servidor Gotify.

Las principales características del equipo anfitrión se presentan en la siguiente tabla.

.

Recurso	Característica
Modelo del equipo	Dell Latitude 7320
Procesador	Intel® Core™ i5-1145G7 (11.ª generación) @ 2.60 GHz
Memoria RAM instalada	16 GB
Sistema operativo anfitrión	Windows 11 Enterprise de 64 bits
Plataforma de virtualización	Oracle VirtualBox
Tipo de conexión de red	Wi-Fi

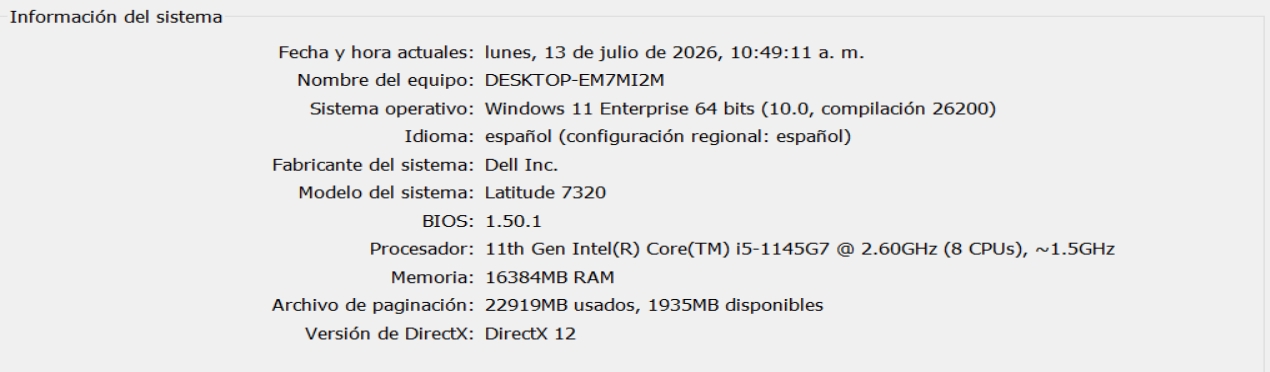


Figura 1. Información general del equipo anfitrión utilizada para el desarrollo del proyecto

2.2. Características de la máquina virtual

El servidor Gotify se instaló y configuró dentro de una máquina virtual creada con Oracle VirtualBox, utilizando como sistema operativo invitado Ubuntu de 64 bits. La máquina virtual fue configurada con los recursos necesarios para compilar el software desde su código fuente y ejecutar los distintos servicios utilizados durante el proyecto.

Las características asignadas fueron las siguientes:

Recurso	Valor
Sistema operativo invitado	Ubuntu (64 bits)
Memoria RAM asignada	10.816 MB
Procesadores virtuales	4 vCPU
Adaptador de red	Adaptador puente (Bridged Adapter)
Interfaz física utilizada	Intel® Wi-Fi 6 AX201 160 MHz
Dirección IP utilizada durante las pruebas	192.168.1.40

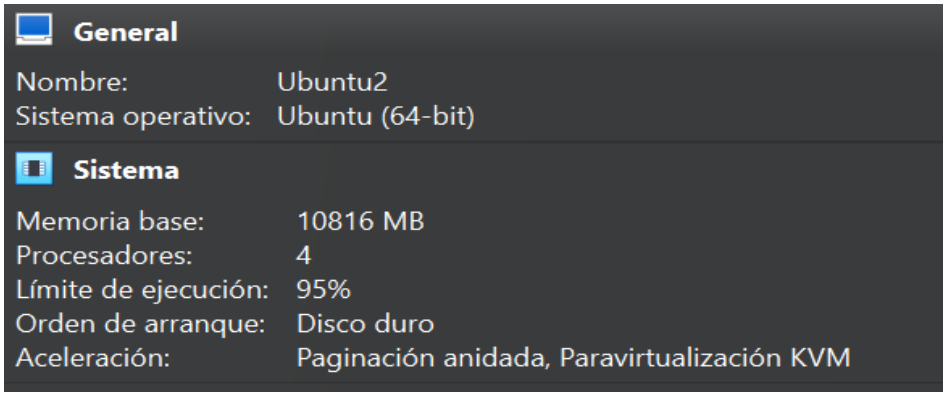


Figura 2. Configuración general de la máquina virtual en Oracle VirtualBox

3. Software y herramientas utilizadas

Para el desarrollo del proyecto fue necesario utilizar diferentes herramientas de desarrollo, compilación y administración del sistema. Estas herramientas permitieron obtener el código fuente de los distintos componentes, compilarlos, configurar el entorno de ejecución y verificar el correcto funcionamiento del servidor Gotify.

3.1. Herramientas de desarrollo

Las principales herramientas utilizadas durante el desarrollo del proyecto se presentan en la siguiente tabla:

Software	Función
Git	Clonado de los repositorios oficiales de Gotify, Gotify CLI y Caddy.
Go	Lenguaje de programación y herramienta de compilación utilizada para construir Gotify Server, Gotify CLI y Caddy desde su código fuente.
Node.js	Entorno de ejecución utilizado para compilar la interfaz web de Gotify.
NVM (Node Version Manager)	Instalación y administración de la versión de Node.js utilizada durante el proyecto.
npm	Instalación de las dependencias utilizadas por la interfaz web de Gotify.
Yarn	Gestión de dependencias y compilación de la interfaz web de Gotify.
GCC	Herramienta de compilación utilizada por distintos componentes y dependencias del proyecto.
Make	Automatización del proceso de compilación mediante archivos *Makefile*.
curl	Realización de pruebas de conectividad y envío de solicitudes a la API REST.
Nano	Edición de archivos de configuración y scripts desde la terminal.

3.2. Componentes instalados y configurados

Los principales componentes instalados y configurados durante el desarrollo del proyecto se presentan en la siguiente tabla:

Software	Descripción
Gotify Server	Servidor autoalojado encargado de recibir, almacenar y distribuir las notificaciones.
Gotify CLI	Cliente oficial de línea de comandos utilizado para enviar notificaciones al servidor.
Caddy	Servidor web configurado como proxy inverso para publicar Gotify a través del puerto 80.
systemd	Administrador de servicios utilizado para gestionar el inicio y la ejecución de Gotify y Caddy.
UFW	Firewall utilizado para administrar el acceso a los puertos del servidor.
ImageMagick	Herramienta utilizada para redimensionar automáticamente las imágenes cargadas en Gotify.

Software	Descripción
OptiPNG	Herramienta utilizada para optimizar imágenes PNG sin pérdida de calidad.
Cron	Servicio utilizado para automatizar la ejecución del script de optimización de imágenes.
Android Studio	Plataforma utilizada para crear y ejecutar un dispositivo Android virtual durante las pruebas.
Gotify para Android	Cliente oficial utilizado para verificar la recepción de notificaciones en el dispositivo Android virtual.

3.3. Versiones del software utilizado

Con el objetivo de documentar el entorno de trabajo empleado durante la compilación, configuración y pruebas del servidor Gotify, se verificaron las versiones de las principales herramientas utilizadas. Los resultados se presentan en la siguiente tabla.

Componente	Versión
Go	1.26.4
Git	2.51.0
Node.js	22.23.1
npm	10.9.8
Yarn	1.22.22
GCC	15.2.0
Make	4.4.1
curl	8.14.1
ImageMagick	7.1.2-3
OptiPNG	7.9.1

4. Guía de instalación, compilación y configuración

En esta sección se presenta el procedimiento seguido para instalar, compilar y configurar el servidor de notificaciones Gotify sobre Ubuntu. Los pasos descriptos permiten reproducir el entorno de trabajo implementado durante el desarrollo del proyecto.

4.1. Preparación del entorno

Como paso previo a la instalación de Gotify, se actualizó el sistema operativo y se instalaron las herramientas básicas necesarias para compilar el software.

Actualizar el índice de paquetes

```
sudo apt update
```

Actualizar los paquetes instalados

```
sudo apt upgrade -y
```

Instalar Git

```
sudo apt install -y git
```

Instalar GCC

```
sudo apt install -y gcc
```

Instalar Make

```
sudo apt install -y make
```

Instalar Build Essential

```
sudo apt install -y build-essential
```

Instalar curl

```
sudo apt install -y curl
```

Una vez completados estos pasos, el sistema quedó preparado para instalar el entorno de desarrollo y continuar con la compilación de los componentes del proyecto.

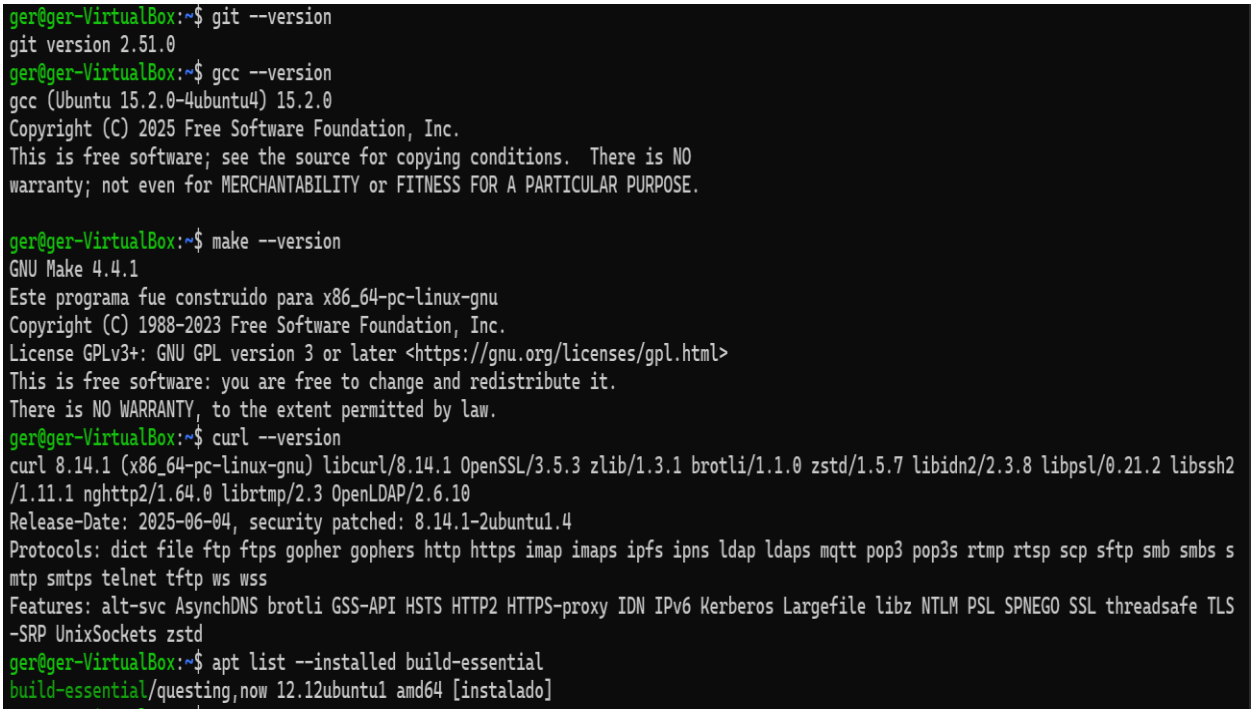


Figura 3. Verificación de la correcta instalación de las herramientas básicas de desarrollo utilizadas durante el proyecto.

4.2. Instalación del lenguaje Go

Como paso previo a la compilación de los componentes del proyecto, se instaló el lenguaje Go utilizando la distribución oficial para Linux de 64 bits, disponible en el sitio web oficial del proyecto.

Descargar la distribución oficial

Descargar el paquete oficial correspondiente a la arquitectura de 64 bits:

```
wget https://go.dev/dl/go1.26.4.linux-amd64.tar.gz
```

Eliminar una instalación previa

En caso de existir una versión previamente instalada, eliminarla antes de continuar con la instalación.

```
sudo rm -rf /usr/local/go
```

Instalar Go

Descomprimir el paquete descargado en el directorio `/usr/local`.

```
sudo tar -C /usr/local -xzf go1.26.4.linux-amd64.tar.gz
```

Configurar la variable de entorno PATH

Agregar el directorio de ejecutables de Go a la variable de entorno `PATH`, para poder utilizar el compilador desde cualquier ubicación del sistema.

```
echo 'export PATH=/usr/local/go/bin:$PATH' >> ~/.bashrc
source ~/.bashrc
```

Verificar la instalación

Comprobar que la instalación se realizó correctamente.

```
go version
```

Una vez finalizada esta etapa, el sistema quedó preparado para compilar Gotify Server, el cliente Gotify CLI y el servidor web Caddy.

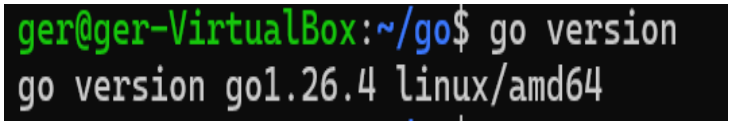


Figura 4. Verificación de la correcta instalación del lenguaje Go.

4.3. Instalación de Node.js y Yarn

De acuerdo con la documentación oficial de Gotify, la compilación de la interfaz web requiere disponer de Node.js y Yarn para instalar las dependencias del proyecto y generar los archivos estáticos que posteriormente serán utilizados por el servidor. Por este motivo, Node.js se instaló mediante NVM (Node Version Manager), siguiendo el procedimiento recomendado por el proyecto.

Instalar NVM (Node Version Manager)

Descargar e instalar NVM utilizando el script oficial del proyecto.

```
curl -o- https://raw.githubusercontent.com/nvm-
sh/nvm/v0.40.3/install.sh | bash
```

Una vez finalizada la instalación, cerrar la terminal y volver a abrirla para que los cambios realizados en el perfil del usuario tengan efecto.

Verificar la instalación.

```
nvm --version
```

Instalar Node.js

Instalar la versión 22 de Node.js mediante NVM.

```
nvm install 22
```

Establecer la versión instalada como predeterminada.

```
nvm alias default 22
```

Verificar la instalación.

```
node -v
npm -v
```

Instalar Yarn

Instalar Yarn utilizando npm.

```
sudo npm install -g yarn
```

Verificar la instalación.

```
yarn -v
```

Una vez completados estos pasos, el sistema quedó preparado para compilar la interfaz web de Gotify.

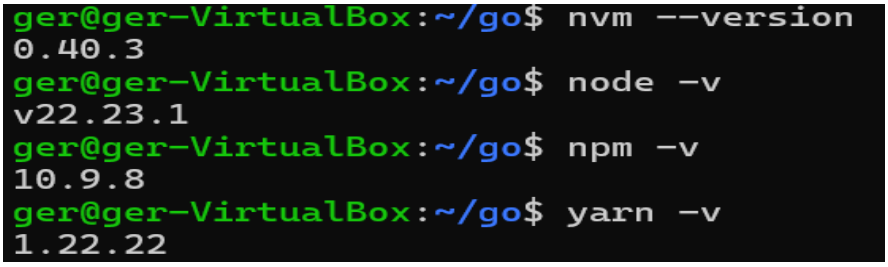


Figura 5. Verificación de la correcta instalación de NVM, Node.js, npm y Yarn.

4.4. Descarga y preparación del código fuente

Una vez instalado el entorno de desarrollo, se procedió a descargar el código fuente del servidor Gotify desde su repositorio oficial y a preparar las dependencias necesarias para su compilación.

Crear el directorio de trabajo

Crear un directorio donde se almacenarán los repositorios del proyecto.

```
mkdir -p ~/proyectos
```

Ingresar al directorio creado.

```
cd ~/proyectos
```

Clonar el repositorio oficial de Gotify

Descargar el código fuente desde el repositorio oficial de GitHub.

```
git clone https://github.com/gotify/server.git
```

Ingresar al directorio del proyecto.

```
cd server
```

Preparar las herramientas de compilación

De acuerdo con la documentación oficial de Gotify, ejecutar el siguiente comando para descargar las herramientas y dependencias necesarias para el proceso de compilación.

```
make download-tools
```

Preparar la interfaz web

Ingresar al directorio correspondiente a la interfaz web.

```
cd ui
```

Instalar las dependencias de la interfaz web utilizando Yarn.

```
yarn
```

Finalizada la instalación, regresar al directorio principal del proyecto.

```
cd ..
```

Una vez completados estos pasos, el código fuente del servidor y sus dependencias quedaron preparados para iniciar el proceso de compilación.

4.5. Compilación del servidor Gotify

Una vez preparado el código fuente y sus dependencias, se procedió a compilar el servidor Gotify. De acuerdo con la documentación oficial, la interfaz web debe compilarse previamente para generar los archivos estáticos que serán incorporados al ejecutable del servidor.

Debido a que la consigna del trabajo requería realizar la compilación sin utilizar Docker, el ejecutable fue generado directamente mediante el comando `go build`.

Compilar la interfaz web

Desde el directorio principal del proyecto ejecutar:

```
(cd ui && yarn build)
```

Compilar el servidor

Generar el ejecutable utilizando el comando `go build`.

```
go build -o gotify-app
```

Verificar la compilación

Comprobar que el ejecutable fue generado correctamente.

```
ls -lh gotify-app
```

Verificar el tipo de archivo obtenido.

```
file gotify-app
```

Una vez finalizada esta etapa, el ejecutable del servidor Gotify quedó listo para su instalación y configuración.

Figura 6. Verificación del ejecutable generado durante la compilación del servidor Gotify.

4.6. Instalación de Gotify

Una vez compilado el servidor, se procedió a instalar los archivos necesarios en su ubicación definitiva dentro del sistema. Para ello se creó un usuario específico para la ejecución del servicio, un directorio de instalación y se copiaron el ejecutable, el archivo de configuración y el directorio de datos.

Crear el usuario del servicio

Crear un usuario de sistema que será utilizado exclusivamente para ejecutar Gotify.

```
sudo useradd \
  --system \
  --home /opt/gotify \
  --shell /usr/sbin/nologin \
  gotify
```

Crear el directorio de instalación

Crear el directorio donde se instalarán los archivos del servidor.

```
sudo mkdir -p /opt/gotify
```

Copiar el ejecutable del servidor

Copiar el ejecutable generado durante la compilación al directorio de instalación.

```
sudo cp gotify-app /opt/gotify/
```

Copiar el archivo de configuración

Copiar el archivo de configuración utilizado por el servidor.

```
sudo cp gotify-server.env /opt/gotify/
```

Copiar el directorio de datos

Copiar el directorio donde Gotify almacenará su base de datos, imágenes y demás archivos persistentes.

```
sudo cp -r data /opt/gotify/
```

Asignar permisos

Asignar la propiedad del directorio al usuario del servicio para que Gotify pueda acceder correctamente a todos sus archivos.

```
sudo chown -R gotify:gotify /opt/gotify
```

Verificar la instalación

Comprobar que los archivos fueron copiados correctamente.

```
ls -lah /opt/gotify
```

Al finalizar la instalación, el directorio quedó organizado de la siguiente manera:

```
/opt/gotify
├── gotify-app
├── gotify-server.env
└── data/
```

En esta etapa el servidor ya se encuentra instalado en el sistema de archivos, aunque todavía no se ejecuta automáticamente como un servicio. La configuración e integración con systemd se realizará en el apartado siguiente.

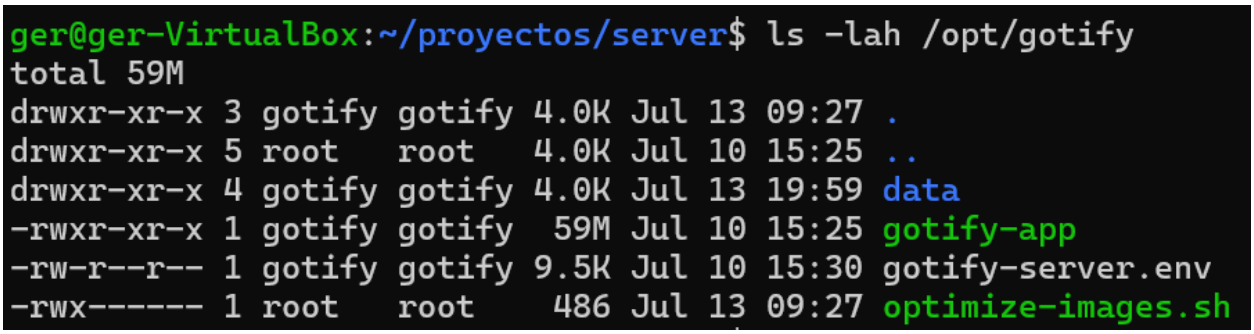


Figura 7. Contenido del directorio de instalación /opt/gotify luego de copiar el ejecutable, el archivo de configuración y el directorio de datos.

4.7. Configuración del servicio Gotify mediante systemd

Una vez instalado Gotify, el siguiente paso consistió en integrarlo con systemd, el administrador de servicios utilizado por Ubuntu. Esto permite iniciar el servidor automáticamente durante el arranque del sistema y administrarlo mediante los comandos estándar de systemctl.

Crear el archivo del servicio

Crear el archivo `gotify.service` dentro del directorio de servicios de systemd.

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

Incorporar la siguiente configuración:

```
[Unit]
Description=Gotify Server
```

```
After=network.target

[Service]
Type=simple
User=gotify
Group=gotify
WorkingDirectory=/opt/gotify
ExecStart=/opt/gotify/gotify-app
Restart=always
RestartSec=5

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

Guardar los cambios y cerrar el editor de texto.

Recargar la configuración de systemd

```
sudo systemctl daemon-reload
```

Habilitar el inicio automático

```
sudo systemctl enable gotify
```

Verificar que el servicio haya quedado habilitado.

```
systemctl is-enabled gotify
```

Si la configuración fue realizada correctamente, el sistema responderá: `enabled`

Iniciar el servicio

```
sudo systemctl start gotify
```

Verificar el funcionamiento

```
sudo systemctl status gotify --no-pager
```

Si la instalación fue realizada correctamente, la salida deberá indicar que el servicio se encuentra en estado: `Active: active (running)`

Con estos pasos, Gotify quedó configurado como un servicio del sistema operativo, iniciándose automáticamente en cada arranque y permitiendo administrar su ejecución mediante los comandos de systemctl.

```
ger@ger-VirtualBox:~/proyectos/server$ systemctl is-enabled gotify
enabled
ger@ger-VirtualBox:~/proyectos/server$ sudo systemctl status gotify --no-pager
● gotify.service - Gotify Server
     Loaded: loaded (/etc/systemd/system/gotify.service; enabled; preset: enabled)
     Active: active (running) since Mon 2026-07-13 17:54:28 -03; 2h 18min ago
 Invocation: 233a2d5271ca483b8b1631a17c149f70
   Main PID: 1711 (gotify-app)
      Tasks: 10 (limit: 11940)
     Memory: 51.6M (peak: 52.1M)
        CPU: 4.686s
     CGroup: /system.slice/gotify.service
             └─1711 /opt/gotify/gotify-app
```

Figura 8. Verificación del servicio Gotify mediante systemctl, mostrando su habilitación para el inicio automático y su estado de ejecución.

4.8. Compilación e instalación de Caddy

Una vez compilado e instalado el servidor Gotify, se procedió a obtener y compilar Caddy, un servidor web que posteriormente será utilizado como proxy inverso para publicar el servicio dentro de la red.
La documentación oficial de Caddy ofrece distintos métodos de instalación y compilación. En este proyecto se optó por compilar el software directamente desde su código fuente utilizando Git y Go, sin emplear binarios precompilados ni imágenes Docker.

Obtener el código fuente

Ingresar al directorio de trabajo.

```
cd ~/proyectos
```

Clonar el repositorio oficial de Caddy indicando la versión utilizada durante el desarrollo del proyecto.

```
git clone --depth 1 --branch v2.7.6
https://github.com/caddyserver/caddy.git caddy-src
```

Ingresar al directorio donde se encuentra el programa principal.

```
cd ~/caddy-src/cmd/caddy
```

Compilar Caddy

Compilar el ejecutable utilizando el comando go build.

```
go build -o caddy
```

Al finalizar el proceso se generará el ejecutable caddy dentro del directorio actual.

Instalar el ejecutable

Copiar el binario compilado a una ubicación accesible para todo el sistema.

```
sudo cp ~/caddy-src/cmd/caddy/caddy /usr/local/bin/
```

Verificar la instalación

Comprobar que Caddy fue instalado correctamente.

```
which caddy
```

Verificar la versión del ejecutable instalado.

```
caddy version
```

Si la instalación fue realizada correctamente, el primer comando mostrará la ubicación del ejecutable y el segundo la versión de Caddy instalada**.**
Una vez completados estos pasos, Caddy quedó instalado y preparado para ser configurado como proxy inverso del servidor Gotify.

```
ger@ger-VirtualBox:~/caddy-src/cmd/caddy$ which caddy
/usr/local/bin/caddy
ger@ger-VirtualBox:~/caddy-src/cmd/caddy$ caddy version
6d9a83376b5e19b3c0368541ee46044ab284038b (07 Dec 23 20:26 UTC)
ger@ger-VirtualBox:~/caddy-src/cmd/caddy$ ls -lh /usr/local/bin/caddy
-rwxr-xr-x 1 root root 41M Jul 12 12:22 /usr/local/bin/caddy
ger@ger-VirtualBox:~/caddy-src/cmd/caddy$
```

Figura 9. Verificación de la correcta instalación de Caddy mediante la consulta de su ubicación, versión y ejecutable instalado

4.9. Configuración del proxy inverso

Una vez instalado Caddy, se configuró como proxy inverso para publicar el servidor Gotify a través del puerto 80 de la máquina virtual. De esta manera, las solicitudes HTTP recibidas por Caddy son reenviadas automáticamente al servidor Gotify, que se ejecuta localmente sobre el puerto 8080.

Crear los directorios de trabajo

Crear los directorios utilizados por Caddy para almacenar su configuración y archivos de trabajo.

```
sudo mkdir -p /etc/caddy
sudo mkdir -p /var/lib/caddy
sudo mkdir -p /var/log/caddy
```

Crear el usuario del servicio

Crear un usuario específico para ejecutar Caddy.

```
sudo useradd \
  --system \
  --home /var/lib/caddy \
  --shell /usr/sbin/nologin \
  caddy
```

Asignar los permisos correspondientes.

```
sudo chown -R caddy:caddy /var/lib/caddy
sudo chown -R caddy:caddy /var/log/caddy
```

Crear el archivo de configuración

Crear el archivo principal de configuración de Caddy.

```
sudo nano /etc/caddy/Caddyfile
```

Incorporar la siguiente configuración:

```
:80 {
    reverse_proxy localhost:8080
}
```

Esta configuración indica que Caddy debe escuchar las conexiones HTTP sobre el puerto 80 y reenviar todas las solicitudes al servidor Gotify, que se ejecuta localmente sobre el puerto 8080. La directiva reverse_proxy permite reenviar las solicitudes recibidas por Caddy hacia el servidor Gotify sin exponer directamente el puerto utilizado por la aplicación.

Validar la configuración

Antes de iniciar el servidor, verificar que el archivo de configuración no contenga errores de sintaxis.

```
caddy validate --config /etc/caddy/Caddyfile
```

Si la configuración es correcta, el comando mostrará el siguiente mensaje:

```
Valid configuration
```

Probar el funcionamiento

Ejecutar Caddy manualmente para comprobar que el proxy inverso funciona correctamente.

```
sudo caddy run --config /etc/caddy/Caddyfile
```

Desde otra terminal verificar la respuesta del servidor.

```
curl http://localhost
```

Si la configuración fue realizada correctamente, se obtendrá el código HTML correspondiente a la interfaz web de Gotify, confirmando que Caddy está reenviando correctamente las solicitudes hacia el servidor.

Una vez completados estos pasos, el proxy inverso quedó correctamente configurado y preparado para ser ejecutado como un servicio administrado por systemd, cuya configuración se realizará en el apartado siguiente.

```
ger@ger-VirtualBox:~/caddy-src/cmd/caddy$ caddy validate --config /etc/caddy/Caddyfile
2026/07/14 12:35:05.281 INFO    using provided configuration    {"config_file": "/etc/caddy/Caddyfile", "config_adapter": ""}
2026/07/14 12:35:05.283 WARN    Caddyfile input is not formatted; run 'caddy fmt --overwrite' to fix inconsistencies    {"adapter": "
caddyfile", "file": "/etc/caddy/Caddyfile", "line": 2}
2026/07/14 12:35:05.283 WARN    http.auto_https server is listening only on the HTTP port, so no automatic HTTPS will be applied to t
his server      {"server_name": "srv0", "http_port": 80}
2026/07/14 12:35:05.283 INFO    tls.cache.maintenance   started background certificate maintenance      {"cache": "0x891748dad00"}
2026/07/14 12:35:05.284 INFO    tls.cache.maintenance   stopped background certificate maintenance      {"cache": "0x891748dad00"}
Valid configuration
```

Figura 10. Validación del archivo de configuración de Caddy mediante el comando `caddy validate`.

```
ger@ger-VirtualBox:~/caddy-src/cmd/caddy$ curl http://localhost
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#3f51b5">
    <link rel="manifest" href="manifest.json">
    <title>Gotify</title>
```

Figura 11. Verificación del funcionamiento del proxy inverso mediante una solicitud HTTP realizada con `curl`, obteniendo como respuesta el código HTML de la interfaz web de Gotify.

4.10. Configuración del servicio Caddy mediante systemd

Una vez configurado el proxy inverso, se integró Caddy con systemd para que el servicio se inicie automáticamente durante el arranque del sistema operativo y pueda administrarse mediante los comandos de systemctl.

Crear el archivo del servicio

Crear el archivo `caddy.service` dentro del directorio de servicios de `systemd`.

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

Incorporar la siguiente configuración:

```
[Unit]
Description=Caddy Web Server
After=network.target

[Service]
User=root
Group=root
ExecStart=/usr/local/bin/caddy run --config /etc/caddy/Caddyfile
ExecReload=/usr/local/bin/caddy reload --config /etc/caddy/Caddyfile
Restart=on-failure
RestartSec=5

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

Guardar los cambios y cerrar el editor.

Recargar la configuración de systemd

```
sudo systemctl daemon-reload
```

Habilitar el inicio automático

```
sudo systemctl enable caddy
```

Verificar que el servicio haya quedado habilitado.

```
systemctl is-enabled caddy
```

Si la configuración fue realizada correctamente, el sistema responderá:

```
enabled
```

Iniciar el servicio

```
sudo systemctl start caddy
```

Verificar el funcionamiento

```
sudo systemctl status caddy --no-pager
```

Si la configuración fue realizada correctamente, la salida deberá indicar que el servicio se encuentra en estado:

```
Active: active (running)
```

Con estos pasos, Caddy quedó configurado como un servicio del sistema operativo, iniciándose automáticamente en cada arranque y permitiendo administrar su ejecución mediante los comandos de systemctl.

```
ger@ger-VirtualBox:~$ systemctl is-enabled caddy
enabled
ger@ger-VirtualBox:~$ sudo systemctl status caddy --no-pager
● caddy.service - Caddy Web Server
     Loaded: loaded (/etc/systemd/system/caddy.service; enabled; preset: enabled)
     Active: active (running) since Tue 2026-07-14 09:07:12 -03; 52min ago
 Invocation: 6d3165b79abb4f879097457a8412e781
   Main PID: 1759 (caddy)
      Tasks: 10 (limit: 11940)
     Memory: 49.7M (peak: 50.9M)
        CPU: 1.512s
     CGroup: /system.slice/caddy.service
             └─1759 /usr/local/bin/caddy run --config /etc/caddy/Caddyfile
```

Figura 12. Verificación del servicio Caddy mediante systemctl, mostrando su habilitación para el inicio automático y su estado de ejecución.

4.11. Configuración del firewall

Para permitir el acceso a los servicios publicados durante el proyecto, se configuró el firewall UFW (Uncomplicated Firewall), habilitando únicamente los puertos necesarios para el funcionamiento del servidor Gotify y del proxy inverso Caddy.

Habilitar el acceso al proxy inverso

Permitir conexiones HTTP hacia Caddy.

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

Habilitar el acceso al servidor Gotify

Permitir el acceso al puerto utilizado por Gotify durante las tareas de configuración y prueba.

```
sudo ufw allow 8080/tcp
```

Verificar la configuración

Comprobar las reglas configuradas en el firewall.

```
sudo ufw status numbered
```

Si la configuración fue realizada correctamente, deberán visualizarse reglas similares a las que se indican en la siguiente tabla.

Puerto	Servicio	Función
80/TCP	Caddy	Publicación del servidor Gotify mediante el proxy inverso.
8080/TCP	Gotify	Acceso directo al servidor durante las tareas de configuración y prueba.

Con esta configuración, el firewall permitió el acceso únicamente a los servicios necesarios para el funcionamiento y las pruebas realizadas durante el proyecto.

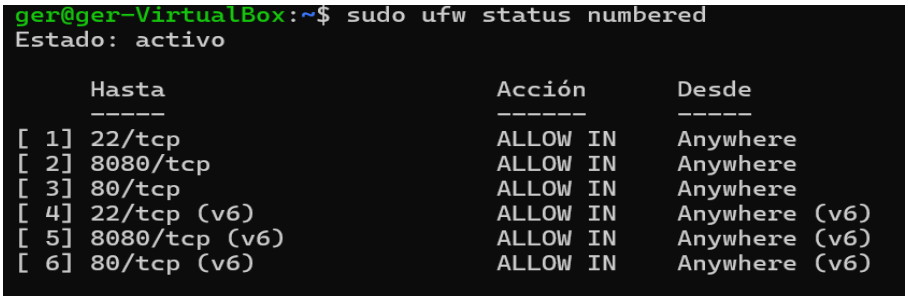


Figura 13. Reglas configuradas en UFW mostrando los puertos habilitados para el acceso al servidor Gotify y al proxy inverso Caddy.

4.12. Optimización de imágenes

Como parte de los requisitos del proyecto, se implementó un mecanismo para optimizar las imágenes utilizadas como logotipos de las aplicaciones creadas en Gotify. El objetivo fue reducir el espacio ocupado por estos archivos sin afectar significativamente su calidad visual. Para ello se utilizaron ImageMagick, encargado del redimensionamiento de las imágenes, y OptiPNG, utilizado para optimizar los archivos PNG sin pérdida de calidad.

Instalar ImageMagick

Instalar ImageMagick desde los repositorios oficiales de Ubuntu.

```
sudo apt install imagemagick
```

Verificar la instalación.

```
magick -version
```

Instalar OptiPNG

Instalar OptiPNG desde los repositorios oficiales de Ubuntu.

```
sudo apt install optipng
```

Verificar la instalación.

```
optipng -v
```

Crear el script de optimización

Crear el archivo que realizará el procesamiento de las imágenes.

```
sudo nano /opt/gotify/optimize-images.sh
```

Agregar el siguiente contenido:

```
#!/usr/bin/env bash
set -e

DATA=/opt/gotify/data

for FILE in "$DATA"/images/*; do
    if [ "$FILE" -nt "$DATA"/images-optimized ]; then
        EXT=$(echo "${FILE##*.}" | tr '[:upper:]' '[:lower:]')

        if [ "$EXT" = "png" ] || [ "$EXT" = "jpg" ] || [ "$EXT" =
"jpeg" ] || [ "$EXT" = "gif" ]; then
            magick "$FILE" -resize "512>" "$FILE"
        fi

        if [ "$EXT" = "png" ]; then
            optipng "$FILE"
        fi
    fi
done

touch "$DATA"/images-optimized
```

Este script recorre las imágenes almacenadas en el directorio `/opt/gotify/data/images`. Los archivos con formato PNG, JPG, JPEG y GIF son redimensionados utilizando ImageMagick, limitando la dimensión más grande de la imagen a 512 píxeles.. Posteriormente, las imágenes en formato PNG son optimizadas mediante OptiPNG. Finalmente, el script registra la fecha de la última optimización para que, en las siguientes ejecuciones, solo se procesen las imágenes nuevas o aquellas que hayan sido modificadas.

Asignar permisos de ejecución

Otorgar permisos para ejecutar el script.

```
sudo chmod u+x /opt/gotify/optimize-images.sh
```

Ejecutar una prueba

Ejecutar manualmente el proceso de optimización.

```
sudo /opt/gotify/optimize-images.sh
```

Verificar el resultado

Comprobar las dimensiones de la imagen optimizada.

```
sudo identify /opt/gotify/data/images/*
```

Verificar el tamaño final del archivo.

```
sudo ls -lh /opt/gotify/data/images
```

Durante las pruebas realizadas, una imagen de aproximadamente 3000 × 2000 píxeles y un tamaño de 1,3 MB fue reducida automáticamente a 512 × 341 píxeles, disminuyendo su tamaño hasta aproximadamente 47 KB.

Con este procedimiento, se reduce considerablemente el espacio ocupado por las imágenes almacenadas en Gotify y el volumen de datos transferidos a los clientes, sin afectar significativamente su calidad visual.

```
ger@ger-VirtualBox:~$ magick -version
Version: ImageMagick 7.1.2-3 Q16 x86_64 23340 https://imagemagick.org
Copyright: (C) 1999 ImageMagick Studio LLC
License: https://imagemagick.org/script/license.php
Features: Cipher DPC Modules OpenMP(4.5)
Delegates (built-in): bzlib djvu fftw fontconfig freetype heic jbig jng jp2 jpeg lcms lqr ltdl lzma openexr pangocairo png raw tiff w
ebp wmf x xml zlib zstd
Compiler: gcc (15.2)
ger@ger-VirtualBox:~$ optipng -v
OptiPNG version 7.9.1
Copyright (C) 2001-2025 Cosmin Truta and the Contributing Authors.

This program is open-source software. See LICENSE for more details.

Portions of this software are based in part on the work of:
  Jean-loup Gailly and Mark Adler (zlib)
  Glenn Randers-Pehrson and the PNG Development Group (libpng)
  Miyasaka Masaru (BMP support)
  David Koblas (GIF support)

Using libpng version 1.6.50 and zlib version 1.3.1
```

Figura 14. Verificación de la instalación de ImageMagick y OptiPNG, herramientas utilizadas para el procesamiento y optimización de imágenes.

```
ger@ger-VirtualBox:~$ sudo identify /opt/gotify/data/images/*
/opt/gotify/data/images/prueba.jpg JPEG 3000x2000 3000x2000+0+0 8-bit sRGB 1.23834MiB 0.000u 0:00.000
ger@ger-VirtualBox:~$ sudo ls -lh /opt/gotify/data/images
total 1.3M
-rw-r--r-- 1 root root 1.3M Jul 14 10:42 prueba.jpg
```

Figura 15(a). Estado de la imagen antes de ejecutar el script de optimización, mostrando sus dimensiones originales (3000 × 2000 píxeles) y un tamaño aproximado de 1,3 MB.

```
ger@ger-VirtualBox:~$ sudo /opt/gotify/optimize-images.sh
ger@ger-VirtualBox:~$ sudo identify /opt/gotify/data/images/*
/opt/gotify/data/images/prueba.jpg JPEG 512x341 512x341+0+0 8-bit sRGB 48090B 0.010u 0:00.001
ger@ger-VirtualBox:~$ sudo ls -lh /opt/gotify/data/images
total 48K
-rw-r--r-- 1 root root 47K Jul 14 10:45 prueba.jpg
```

Figura 15(b). Resultado obtenido luego de ejecutar el script de optimización, mostrando la reducción de la imagen a 512 × 341 píxeles y un tamaño aproximado de 47 KB.

4.13. Automatización mediante Cron

Una vez verificado el funcionamiento del script de optimización de imágenes, se automatizó su ejecución utilizando Cron, el planificador de tareas incluido en Ubuntu. De esta manera, el proceso de optimización se ejecuta automáticamente todos los días sin requerir intervención del usuario.

Crear la tarea programada

Crear el archivo de configuración dentro del directorio de tareas programadas de Cron.
sudo nano /`etc/cron.d/gotify`
Agregar la siguiente línea:

```
12 12 * * * root /opt/gotify/optimize-images.sh
```

Esta configuración indica que el script optimize-images.sh será ejecutado diariamente a las 12:12 horas utilizando el usuario root.

Verificar el archivo de configuración

Comprobar que la tarea programada fue creada correctamente.

```
cat /etc/cron.d/gotify
```

Verificar el servicio Cron

Comprobar que el servicio Cron se encuentre en ejecución.

```
sudo systemctl status cron --no-pager
```

Si la configuración fue realizada correctamente, la salida deberá indicar que el servicio se encuentra en estado:

```
Active: active (running)
```

Con esta configuración, la optimización de imágenes quedó automatizada, manteniendo optimizados los archivos almacenados por Gotify sin necesidad de intervención manual.

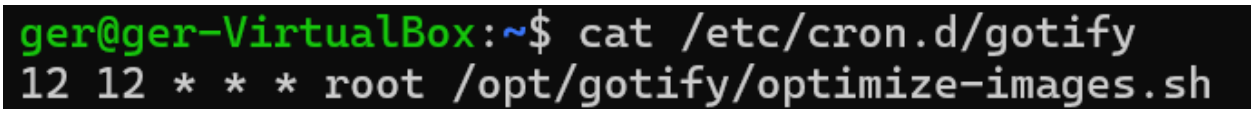


Figura 16. Verificación de la tarea programada para la ejecución automática del script de optimización mediante Cron.

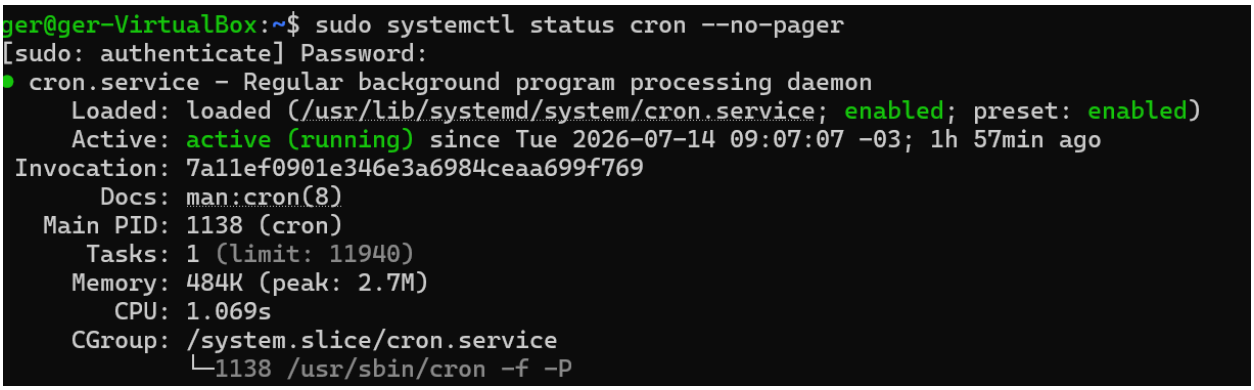


Figura 17. Verificación del servicio Cron en ejecución mediante systemctl.

4.14. Compilación e instalación de Gotify CLI

Con el objetivo de disponer de una herramienta de línea de comandos para enviar notificaciones al servidor, se compiló e instaló Gotify CLI a partir de su código fuente. Aunque la documentación oficial prioriza la distribución mediante binarios precompilados, el proyecto incluye un Makefile que muestra el procedimiento de compilación utilizando Go. En este trabajo se compiló únicamente el ejecutable correspondiente al sistema Linux de 64 bits.

Obtener el código fuente

Ingresar al directorio de trabajo.

```
cd ~/proyectos
```

Clonar el repositorio oficial de Gotify CLI.

```
git clone https://github.com/gotify/cli.git gotify-cli-src
```

Ingresar al directorio del proyecto.

```
cd ~/proyectos/gotify-cli-src
```

Compilar Gotify CLI

Generar el ejecutable utilizando el comando go build.

```
go build -o gotify-cli cli.go
```

Instalar el ejecutable

Copiar el ejecutable compilado a una ubicación accesible para todo el sistema.

```
sudo cp gotify-cli /usr/local/bin/
```

Verificar la instalación

Comprobar que el ejecutable fue instalado correctamente.

```
which gotify-cli
```

Verificar que el ejecutable fue generado correctamente.

```
ls -lh gotify-cli
```

Verificar el tipo de archivo obtenido.

```
file gotify-cli
```

Una vez completados estos pasos, Gotify CLI quedó instalado y preparado para enviar notificaciones al servidor desde la línea de comandos. Su funcionamiento será verificado en la sección de pruebas.

```
ger@ger-VirtualBox:~/proyectos/gotify-cli-src$ which gotify-cli
/usr/local/bin/gotify-cli
ger@ger-VirtualBox:~/proyectos/gotify-cli-src$ ls -lh gotify-cli
-rwxrwxr-x 1 ger ger 16M Jul 14 12:22 gotify-cli
ger@ger-VirtualBox:~/proyectos/gotify-cli-src$ file gotify-cli
gotify-cli: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, Go Buil
dID=rrcdGWhaJFE6iWzxiD5d/_BkC4EIjoSR2n22FxTTM/lVVGSyPLzA2jD2b_KXdt/oitESCWkK01H0TcX2odL, BuildID[sha1]=cc1de6c286e5a80d4335c6a38fb28d
11aa21b16f, with debug_info, not stripped
```

Figura 18. Verificación de la compilación e instalación de Gotify CLI, mostrando la ubicación del ejecutable instalado en /usr/local/bin y las características del binario generado.

5. Pruebas de funcionamiento

Con el objetivo de comprobar el correcto funcionamiento del sistema implementado, se realizaron distintas pruebas utilizando los diferentes mecanismos de acceso y envío de notificaciones disponibles en Gotify.

Las verificaciones incluyeron el acceso mediante la interfaz web, el envío de notificaciones utilizando la API REST y Gotify CLI, la recepción de mensajes en el cliente oficial para Android y la comprobación del funcionamiento de los servicios configurados.

5.1. Acceso mediante la interfaz web

Una vez finalizada la instalación y configuración del servidor Gotify y del proxy inverso Caddy, se verificó el acceso al sistema utilizando la interfaz web proporcionada por la aplicación.

Acceder al servidor

Abrir un navegador web e ingresar la dirección del servidor.

```
http://192.168.1.40
```

Reemplazar la dirección IP por la correspondiente al servidor utilizado.

Iniciar sesión

Ingresar las credenciales configuradas durante la instalación de Gotify y acceder al panel principal de la aplicación.

Verificar el funcionamiento

Si la configuración fue realizada correctamente, se visualizará la interfaz principal de Gotify, desde la cual es posible administrar aplicaciones, usuarios, tokens y visualizar las notificaciones recibidas.

Con esta prueba se comprobó que el servidor Gotify, el proxy inverso Caddy y la interfaz web funcionan correctamente, permitiendo acceder al sistema desde un navegador web y administrar las aplicaciones, usuarios y notificaciones.

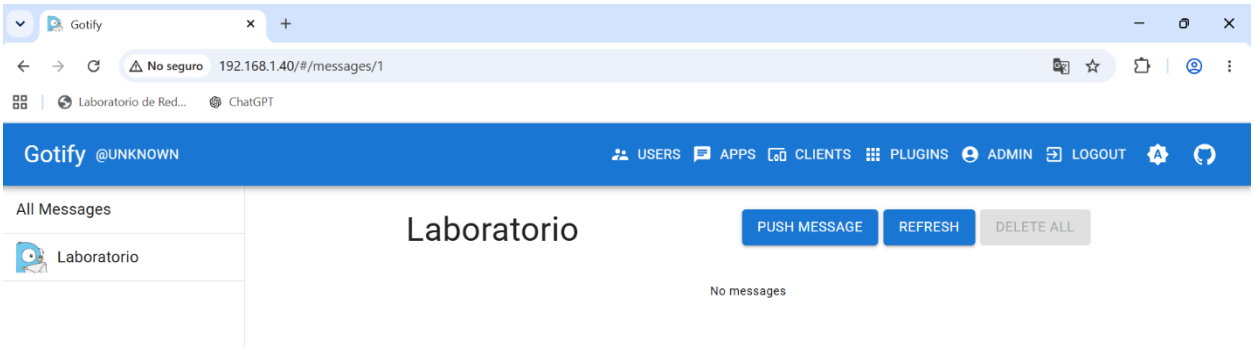


Figura 19. Interfaz principal de Gotify accesible desde un navegador web luego de completar la instalación y configuración del servidor, lista para recibir notificaciones

5.2. Envío de notificaciones mediante la API REST

Una vez verificado el acceso a la interfaz web, se comprobó el funcionamiento de la API REST de Gotify enviando una notificación mediante el comando curl.

Enviar una notificación

Ejecutar el siguiente comando desde la terminal:

```
curl -X POST "http://localhost/message?token=TOKEN_DE_LA_APLICACION" \
-F "title=Prueba API REST" \
-F "message=Notificación enviada mediante la API REST utilizando
curl." \
-F "priority=5"
```

Reemplazar `TOKEN_DE_LA_APLICACION` por el token generado para la aplicación configurada en Gotify.

Si el envío se realizó correctamente, el servidor responderá con un mensaje en formato JSON que contiene la información de la notificación creada.

Verificar el resultado

Actualizar la interfaz web de Gotify y comprobar que la notificación enviada aparezca en la lista de mensajes recibidos.

Con esta prueba se verificó el correcto funcionamiento de la API REST y la recepción inmediata de las notificaciones enviadas al servidor. Asimismo, se comprobó el correcto funcionamiento del proxy inverso Caddy, ya que la solicitud fue enviada al puerto 80 y reenviada automáticamente al servidor Gotify.

```
ger@ger-VirtualBox:~$ curl -X POST "http://localhost/message?token=gtfya.nhGSv5enBnGWUHKxXDM-LSIit_sfKYzELC74zld6hnE" \
-F "title=Prueba API REST" \
-F "message=Notificación enviada mediante la API REST utilizando curl." \
-F "priority=5"
{"id":6,"appid":1,"message":"Notificación enviada mediante la API REST utilizando curl.","title":"Prueba API REST","priority":5,"date
":"2026-07-14T14:52:45.493418108-03:00"}ger@ger-VirtualBox:~$
```

Figura 20. Envío de una notificación al servidor Gotify mediante la API REST utilizando el comando curl.

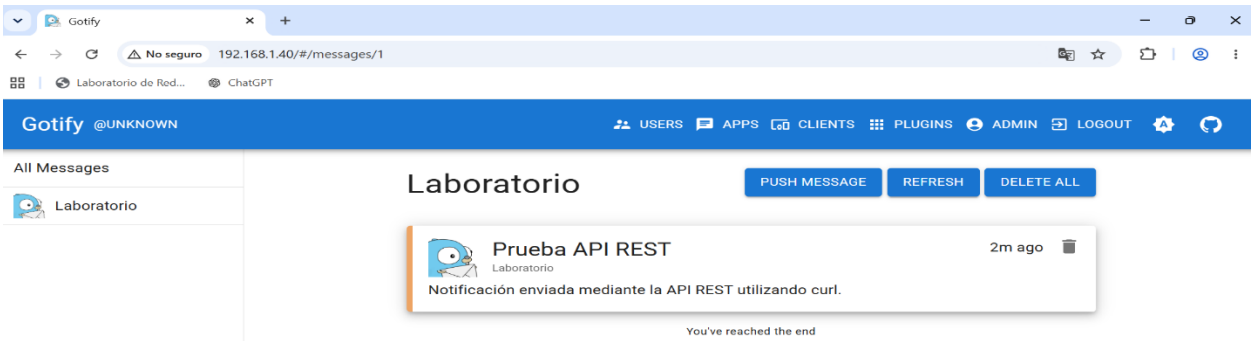


Figura 21. Visualización en la interfaz web de Gotify de la notificación enviada mediante la API REST.

5.3. Envío de notificaciones mediante Gotify CLI

Una vez comprobado el funcionamiento de la API REST, se verificó el cliente oficial de línea de comandos **Gotify CLI**, compilado previamente desde el código fuente. Para ello, se configuró el cliente mediante el asistente interactivo y posteriormente se envió una notificación al servidor.

Configurar Gotify CLI
Inicializar el cliente ejecutando el siguiente comando:

```
gotify-cli init
```

Durante la configuración se deberán indicar los siguientes parámetros:

- URL del servidor Gotify.
- Credenciales del usuario administrador.
- Nombre de la aplicación.
- Descripción de la aplicación (opcional).
- Prioridad predeterminada.
- Ubicación donde se almacenará el archivo de configuración (cli.json).

Una vez finalizado el asistente, el archivo de configuración (cli.json) deberá contener el token correspondiente a la aplicación utilizada durante el proyecto.

Nota: Durante las pruebas se observó que el asistente de configuración genera automáticamente una nueva aplicación y su correspondiente token de acceso. Para reutilizar la aplicación creada previamente en Gotify, fue necesario reemplazar el token almacenado en el archivo cli.json por el token de la aplicación original utilizada durante el proyecto.

```
ger@ger-VirtualBox:~$ gotify-cli init
Gotify URL: http://localhost
Connecting -> Success!
Gotify vunknown@unknown

Configure an application token
1. Enter an application-token
2. Create an application token (with user/pass)
Enter 1 or 2: 2

Enter Credentials (only used for creating the token not saved afterwards)
Username: admin
Password:
Authenticating -> Success!

Application name: Laboratorio
Application description (can be empty): Cliente de línea de comandos utilizado para las pruebas del TP.
Creating -> Success!

Default Priority [0-10]: 5


Where to put the config file?
1. ./cli.json
2. /home/ger/.config/gotify/cli.json
3. /home/ger/.gotify/cli.json
4. /etc/gotify/cli.json
Enter a number: 2

Written config to: /home/ger/.config/gotify/cli.json
ger@ger-VirtualBox:~$
```

Figura 22. Configuración inicial del cliente Gotify CLI mediante el asistente interactivo, indicando la dirección del servidor, las credenciales del administrador y los parámetros de configuración de la aplicacion

Enviar una notificación

Ejecutar el siguiente comando:

```
gotify-cli push \
-t "Prueba Gotify CLI" \
-p 5 \
"Notificación enviada mediante el cliente oficial de línea de comandos."
```

Si el envío se realizó correctamente, el cliente responderá con el siguiente mensaje:

```
message created
```

```
ger@ger-VirtualBox:~$ gotify-cli push \
-t "Prueba Gotify CLI" \
-p 5 \
"Notificación enviada mediante el cliente oficial de línea de comandos."
message created
```

Figura 23. Envío de una notificación al servidor Gotify utilizando el cliente oficial de línea de comandos (Gotify CLI).

Verificar el resultado

Actualizar la interfaz web de Gotify y comprobar que la notificación enviada aparezca en la lista de mensajes recibidos.

Con esta prueba se verificó el correcto funcionamiento del cliente oficial Gotify CLI, confirmando que es posible enviar notificaciones al servidor desde la línea de comandos y que estas son recibidas inmediatamente por la interfaz web de Gotify.

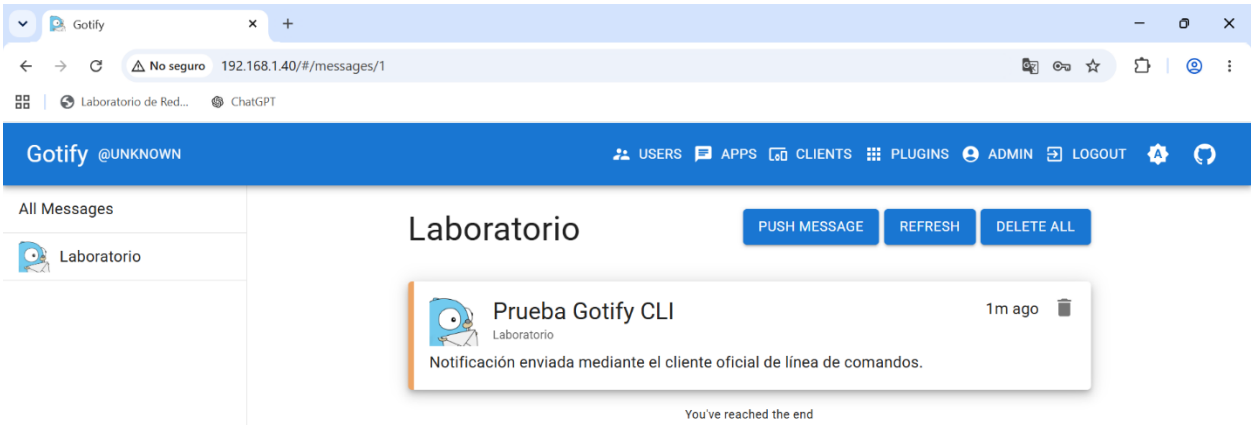


Figura 24. Notificación recibida en la interfaz web de Gotify luego de ser enviada mediante el cliente oficial Gotify CLI.

5.4. Recepción de notificaciones en el cliente Android

Con el objetivo de comprobar el funcionamiento del cliente móvil de Gotify, se realizaron pruebas utilizando un dispositivo Android emulado mediante Android Studio. Para ello, se instaló la aplicación oficial de Gotify utilizando su archivo APK, disponible en la sección "Releases" del repositorio oficial del proyecto.

Instalar y configurar el cliente Android

Iniciar Android Studio y ejecutar un dispositivo Android virtual (Android Virtual Device - AVD). Una vez iniciado el emulador, instalar la aplicación oficial de Gotify mediante el archivo APK descargado.

Abrir la aplicación e ingresar la dirección del servidor Gotify configurado durante el proyecto. Luego de verificar la conexión con el servidor, introducir las credenciales del usuario administrador para autenticarse. Una vez iniciada la sesión, crear un nuevo cliente Android indicando el nombre que identificará al dispositivo. Finalmente, asignar un nombre al cliente y completar la configuración. Si el procedimiento se realizó correctamente, la aplicación quedará conectada al servidor y preparada para recibir notificaciones.

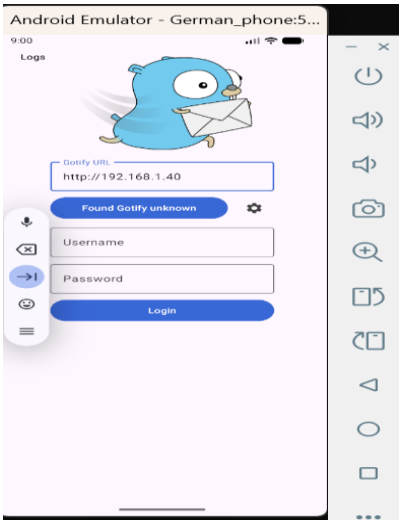


Figura 25. Configuración del cliente oficial de Gotify en Android indicando la dirección del servidor y las credenciales de acceso.

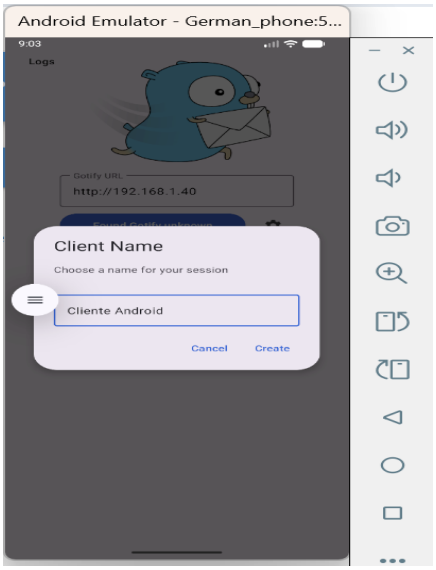


Figura 26. Creación del cliente Android que será utilizado para recibir las notificaciones enviadas por el servidor.

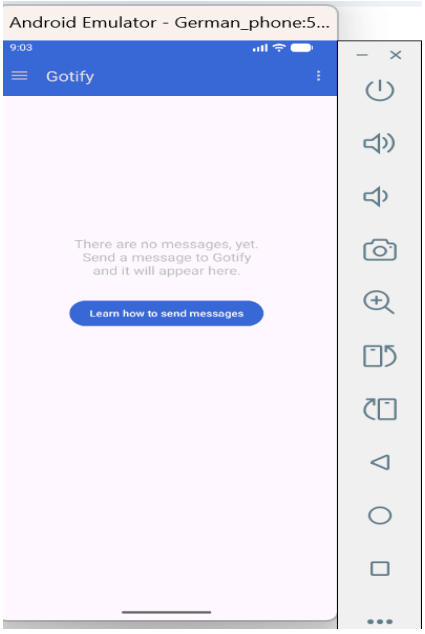


Figura 27. Cliente Android conectado correctamente al servidor Gotify y preparado para recibir notificaciones.

Enviar una notificación de prueba

Desde el servidor, enviar una notificación utilizando el cliente oficial Gotify CLI mediante el siguiente comando:

```
gotify-cli push \
-t "Prueba Android" \
-p 5 \
"Notificación recibida correctamente en el cliente Android."
```

Si el envío se realizó correctamente, la terminal mostrará el siguiente mensaje:
`message created`

```
ger@ger-VirtualBox:~$ gotify-cli push \
-t "Prueba Android" \
-p 5 \
"Notificación recibida correctamente en el cliente Android."
message created
ger@ger-VirtualBox:~$
```

Figura 28. Envío de una notificación desde el servidor utilizando el cliente oficial Gotify CLI.

Verificar el resultado

Actualizar el cliente Android y comprobar que la notificación enviada aparezca correctamente en la lista de mensajes recibidos.

Con esta prueba se verificó el correcto funcionamiento del cliente oficial de Gotify para Android, confirmando que las notificaciones enviadas desde el servidor son recibidas correctamente en el dispositivo emulado.

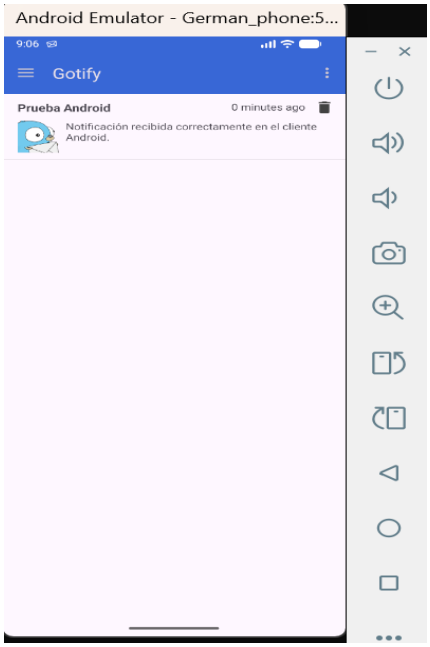


Figura 29. Recepción de la notificación enviada desde el servidor en el cliente oficial de Gotify para Android.

6. Estado final de la infraestructura

Al finalizar el proyecto se obtuvo un servidor Gotify completamente funcional ejecutándose sobre una máquina virtual con Ubuntu. La infraestructura implementada permite administrar y distribuir notificaciones desde una única plataforma, utilizando diferentes mecanismos de acceso y clientes.

El servidor Gotify fue compilado desde el código fuente y configurado para funcionar junto con el proxy inverso Caddy, permitiendo el acceso a la interfaz web mediante el puerto 80.

Asimismo, se compiló el cliente oficial Gotify CLI para el envío de notificaciones desde la línea de comandos y se configuró el cliente oficial para Android utilizando un dispositivo virtual ejecutado mediante Android Studio.

Como parte de la infraestructura también se implementó un sistema de optimización automática de imágenes utilizando ImageMagick y OptiPNG, cuya ejecución fue automatizada mediante Cron para procesar periódicamente los archivos almacenados en el servidor.
Durante las pruebas de funcionamiento se verificó correctamente el acceso a la interfaz web, el envío de notificaciones mediante la API REST y Gotify CLI, así como la recepción de los mensajes en el cliente Android. Todas las verificaciones realizadas demostraron el correcto funcionamiento de los componentes instalados y la adecuada integración entre ellos.
De esta manera, se obtuvo una infraestructura estable y completamente operativa, capaz de administrar el envío y la recepción de notificaciones utilizando diferentes clientes y mecanismos de comunicación, cumpliendo con los objetivos planteados para el desarrollo del proyecto.

7. Problemas encontrados y soluciones

Durante el desarrollo del proyecto surgieron diversos inconvenientes técnicos relacionados con la instalación, compilación y configuración de los distintos componentes del sistema. A continuación, se describen los principales problemas encontrados y las soluciones implementadas para resolverlos.

7.1. Problemas de resolución de nombres (DNS)

Al comenzar la instalación del entorno de desarrollo se presentaron inconvenientes para acceder a algunos repositorios oficiales debido a problemas de resolución de nombres (DNS) en la máquina virtual. Como consecuencia, no era posible descargar correctamente algunas dependencias necesarias para la compilación del software.
La situación se resolvió configurando los servidores DNS públicos de Cloudflare (1.1.1.1) y Google (8.8.8.8). Una vez aplicada esta configuración, la resolución de nombres volvió a funcionar correctamente y fue posible continuar con la instalación del entorno de desarrollo.

7.2. Renovación de la dirección IP mediante DHCP

Durante algunas etapas del proyecto, la máquina virtual perdió la conectividad con la red local debido a que la dirección IP asignada dinámicamente dejó de ser válida o no fue renovada correctamente por el servidor DHCP.
La situación se resolvió renovando la dirección IP asignada a la máquina virtual y verificando posteriormente la conectividad de red. Una vez obtenida una nueva dirección IP, fue posible continuar con el desarrollo y las pruebas del proyecto.

7.3. Problemas de conectividad entre IPv4 e IPv6

Durante la descarga de algunas dependencias se observó que determinadas conexiones intentaban establecerse mediante IPv6, mientras que la red utilizada durante el proyecto operaba correctamente sobre IPv4. Esta situación ocasionaba demoras y fallos intermitentes durante las descargas.
El problema se resolvió realizando las pruebas de conectividad y las descargas utilizando IPv4, verificando además que todos los servicios fueran accesibles mediante la dirección IPv4 asignada a la máquina virtual.

7.4. Descarga lenta de dependencias de Go

Durante la compilación de Go y posteriormente del servidor Gotify, la descarga de las dependencias resultó considerablemente más lenta de lo esperado.

Para solucionar este inconveniente se configuró el proxy oficial de módulos de Go (GOPROXY), mejorando significativamente la velocidad de descarga y permitiendo completar la compilación sin inconvenientes.

7.5. Conflicto de puertos con Apache

Durante la configuración del proxy inverso se detectó que el puerto 80 ya estaba siendo utilizado por el servicio Apache instalado en el sistema, impidiendo el inicio de Caddy.
La solución consistió en detener y deshabilitar el servicio Apache, liberando el puerto para que pudiera ser utilizado por Caddy.

7.6. Configuración del firewall

En las primeras pruebas no era posible acceder al servidor desde otros equipos de la red debido a las reglas configuradas en el firewall.
El inconveniente se resolvió agregando las excepciones correspondientes para los puertos utilizados por el proyecto (80 y 8080), permitiendo el acceso tanto al servidor Gotify como al proxy inverso.

7.7. Configuración de los servicios del sistema

La integración de Gotify y Caddy con systemd requirió varios ajustes hasta definir correctamente los archivos de servicio y asegurar su inicio automático junto con el sistema operativo.
Una vez completada la configuración, ambos servicios fueron verificados mediante systemctl, comprobando que permanecieran en estado active (running) y se iniciaran automáticamente después de reiniciar la máquina virtual.

7.8. Incompatibilidad del token en Gotify CLI

Durante la configuración de Gotify CLI se detectó que el cliente rechazaba el token generado por el servidor, debido a que el código fuente validaba que este tuviera exactamente 15 caracteres de longitud, mientras que las versiones actuales de Gotify Server generan tokens más largos, identificados mediante los prefijos gtfya. (aplicación) y gtfyc. (cliente). Al no cumplirse esta validación, la inicialización del cliente mediante el comando gotify-cli init fallaba al intentar utilizar el token de la aplicación creada en el servidor. Para resolver este inconveniente, se modificó el código fuente del archivo command/initialize.go, reemplazando la condición de validación de longitud exacta (!= 15) por una validación de longitud mínima (< 15), permitiendo así aceptar tokens de mayor longitud sin comprometer la verificación básica del formato. Una vez aplicado el cambio mediante el comando sed, el ejecutable de Gotify CLI fue recompilado con go build, de esta manera, el cliente pudo inicializarse correctamente utilizando el token real de la aplicación configurada en el servidor

8. Uso de Inteligencia Artificial

Durante el desarrollo del proyecto se utilizó ChatGPT (OpenAI, modelo GPT-5.5) como herramienta de apoyo para colaborar en la resolución de problemas técnicos específicos y en la revisión de la documentación elaborada.
La inteligencia artificial fue utilizada principalmente para:

- Analizar mensajes de error durante la compilación de los distintos componentes del proyecto.

- Colaborar en la resolución de inconvenientes relacionados con la configuración de la red, servicios y dependencias.
- Asistir en la interpretación de errores asociados a Go, Node.js y otras herramientas utilizadas durante la instalación.
- Brindar apoyo en la configuración de componentes como Caddy, systemd y la automatización mediante Cron.

Ejemplos de prompts utilizados

Algunos de los prompts utilizados durante el desarrollo del trabajo fueron:

- "Analizar el error generado durante la compilación de Gotify y proponer posibles causas."
- "Verificar la configuración de Caddy como proxy inverso para Gotify."
- "Revisar la configuración de los servicios systemd y validar su funcionamiento."
- "No puedo descargar los módulos de Go, está muy lento, ¿cómo configuro un proxy para que sea más rápido?"
- “Como puedo solucionar la incompatibilidad de los tokens que se generan al crear una app con Gotify cli”

9. Conclusiones

El desarrollo de este proyecto permitió instalar, compilar y configurar exitosamente un servidor de notificaciones autoalojado utilizando Gotify, empleando exclusivamente herramientas de software libre y siguiendo la documentación oficial.

Durante el trabajo fue posible profundizar conocimientos relacionados con la compilación de aplicaciones desde su código fuente, la administración de servicios mediante systemd, la configuración de un proxy inverso utilizando Caddy, la gestión del firewall UFW y la automatización de tareas mediante Cron. Asimismo, se incorporaron herramientas complementarias como ImageMagick y OptiPNG para optimizar automáticamente las imágenes enviadas al servidor.

Las distintas pruebas realizadas permitieron verificar el correcto funcionamiento de toda la infraestructura implementada, comprobando el acceso mediante la interfaz web, el envío de notificaciones utilizando la API REST y Gotify CLI, así como la recepción de mensajes en el cliente oficial para Android.

Además de cumplir con los objetivos planteados para el trabajo final, el proyecto permitió aplicar y profundizar numerosos conceptos abordados durante la cursada, fortaleciendo los conocimientos adquiridos sobre administración de sistemas Linux, servicios de red y compilación de software.

En conclusión, se cumplieron satisfactoriamente los objetivos propuestos, obteniendo una infraestructura completamente funcional, reproducible y adecuadamente documentada, que puede servir como guía para futuras instalaciones y configuraciones de Gotify en entornos similares.

10. Referencias

Go Project. Installing Go from source. https://go.dev/doc/install/source
Go Project. Documentación oficial. https://go.dev/doc/
Gotify. Documentación oficial. https://gotify.net/docs/
Gotify. Repositorio oficial de Gotify Server. https://github.com/gotify/server
Gotify. Repositorio oficial de Gotify CLI. https://github.com/gotify/cli
Caddy. Build from source. https://caddyserver.com/docs/build
Caddy. Documentación oficial. https://caddyserver.com/docs/
ImageMagick Studio LLC. ImageMagick Documentation. https://imagemagick.org/
OptiPNG Project. OptiPNG Documentation. https://optipng.sourceforge.net/
Ubuntu Project. UFW–Uncomplicated Firewall. https://help.ubuntu.com/community/UFW
freedesktop.org. systemd Documentation. https://systemd.io/
Android Developers. Android Studio. https://developer.android.com/studio