**INFORME DE GUÍA PRÁCTICA**

1. **PORTADA**

Tema: Servidores (Procesos y Comunicación)

Unidad de Organización Curricular: PROFESIONAL

Nivel y Paralelo: Nivel - Paralelo

Alumnos participantes: Albán Chávez Melanie Elizabeth

Álvarez Coello Freddy Guillermo

Salas Acosta Alison Micaela

Soriano Proaño Rafael Andrés

Asignatura: Aplicaciones Distribuidas

Docente: Ing. José Caiza, Mg.

1. **INFORME DE GUÍA PRÁCTICA**
2. **PP**
3. **YY**
   1. **Objetivos**

**General:**

Conocer los modelos arquitectónicos de los sistemas distribuidos para una implementación exitosa manteniendo la integridad, la escalabilidad y la confiabilidad en un sistema distribuido.

**Específicos:**

* Diseñar e implementar una arquitectura distribuida que integre servicios de red como proxy inverso, servidor de API y base de datos utilizando contenedores Docker.
* Configurar un servidor proxy Squid para el control de acceso a sitios web mediante reglas dinámicas gestionadas desde una API.
* Automatizar la comunicación entre microservicios utilizando protocolos modernos como HTTP o gRPC para garantizar la interoperabilidad del sistema.
  1. **Modalidad**

Presencial

* 1. **Tiempo de duración**

**Presenciales:** 4

**No presenciales:** 0

* 1. **Instrucciones**

El trabajo se desarrollará en parejas.

Lea las indicaciones del archivo adjunto y desarrolle las actividades solicitadas.

La práctica se revisará en clase y el informe se debe subir al aula virtual de la materia en formato PDF.

* 1. **Listado de equipos, materiales y recursos**

Listado de equipos y materiales generales empleados en la guía práctica:

* Inteligencia artificial
* Internet
* Bases de datos disponibles en la biblioteca virtual de la Universidad.
* Bibliografía de la asignatura.
* Material disponible en el aula virtual de la asignatura.
* Virtual Box
* Linux

TAC (Tecnologías para el Aprendizaje y Conocimiento) empleados en la guía práctica:

Plataformas educativas

Simuladores y laboratorios virtuales

Aplicaciones educativas

Recursos audiovisuales

Gamificación

Inteligencia Artificial

Otros (Especifique): Docker

* 1. **Actividades por desarrollar**
* Bloqueo de URLs con patrones o palabras clave con Squid (con ACLs y expresiones regulares).
* API para controlar reglas dinámicamente con ASP.NET Core API.
* Servidor web separado con Ubuntu + Kestrel (o Nginx como proxy inverso).
* Servidor de base de datos separado con Ubuntu + SQL Server o MongoDB.
* Comunicación entre microservicios con gRPC o HTTP (según preferencia)
  1. **Resultados obtenidos**

La API fue desarrollada en Visual Studio utilizando el framework .NET 9 y el lenguaje C#, con el objetivo de gestionar reglas de bloqueo de URLs. Se implementó un controlador llamado UrlRulesController, que permite realizar operaciones CRUD sobre una base de datos MongoDB, almacenando patrones que serán utilizados por un servidor proxy Squid. Cada vez que se agrega o elimina una regla, el sistema actualiza automáticamente un archivo llamado blocked\_patterns.txt, el cual es monitoreado por un script que recarga la configuración de Squid sin necesidad de reiniciar el servicio. Para facilitar el despliegue, la API fue contenedorizada mediante Docker utilizando un Dockerfile basado en la imagen oficial de ASP.NET 9.0 (mcr.microsoft.com/dotnet/aspnet:9.0). En este archivo se define el directorio de trabajo (/app), se copian los archivos publicados desde bin/Release/net9.0/publish/, se expone el puerto 80 y se establece como punto de entrada la ejecución del archivo FuelApi.dll, lo que permite que la aplicación inicie automáticamente al levantar el contenedor.

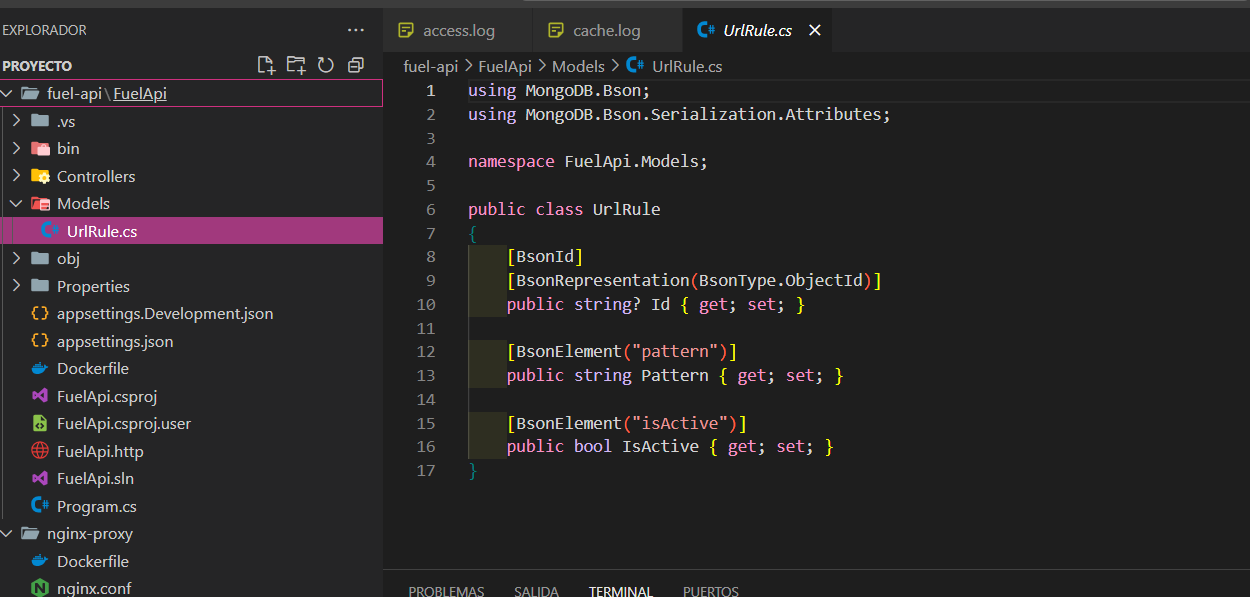


Ilustración 1. Modelo para creación de la API

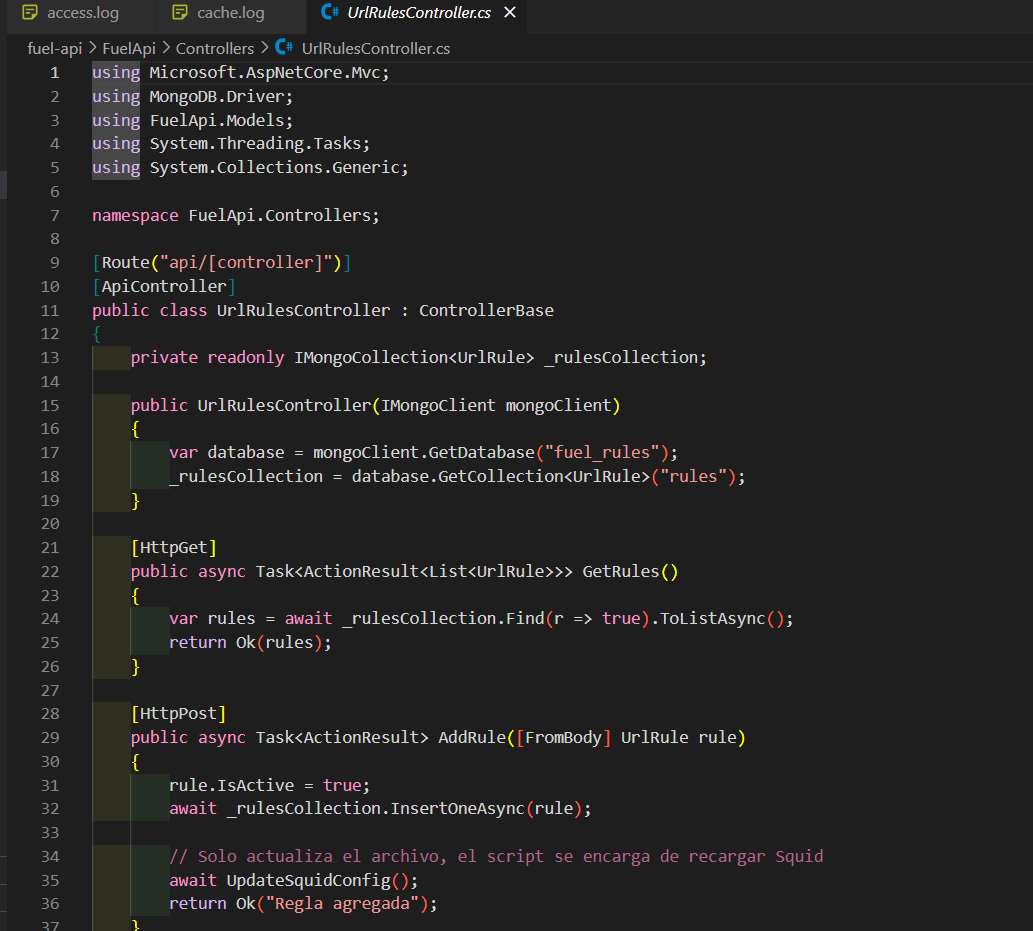


Ilustración 2. Métodos CRUD en el Controller

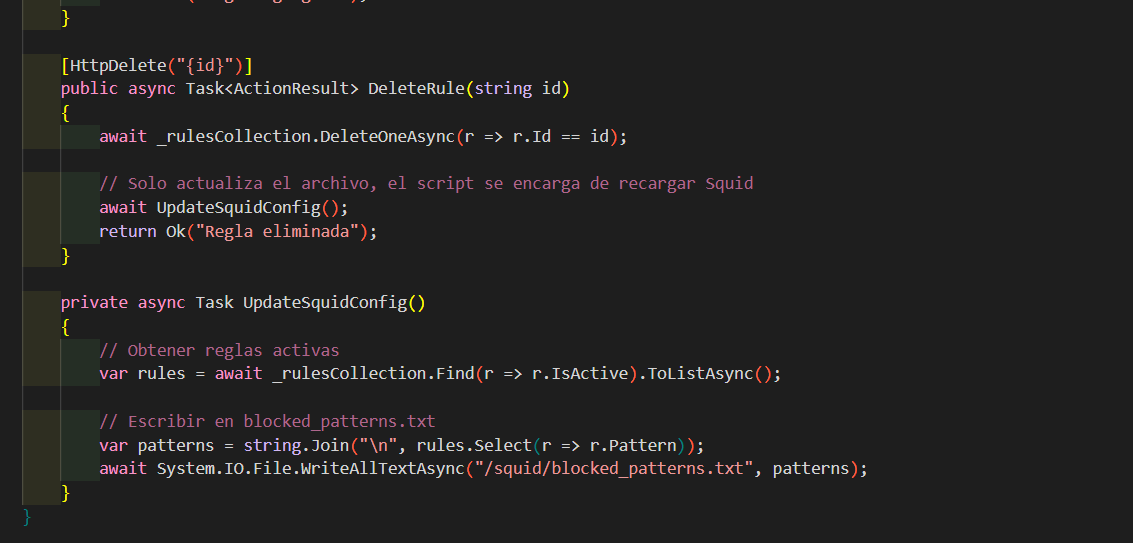


Ilustración 3. Métodos CRUD en el Controller

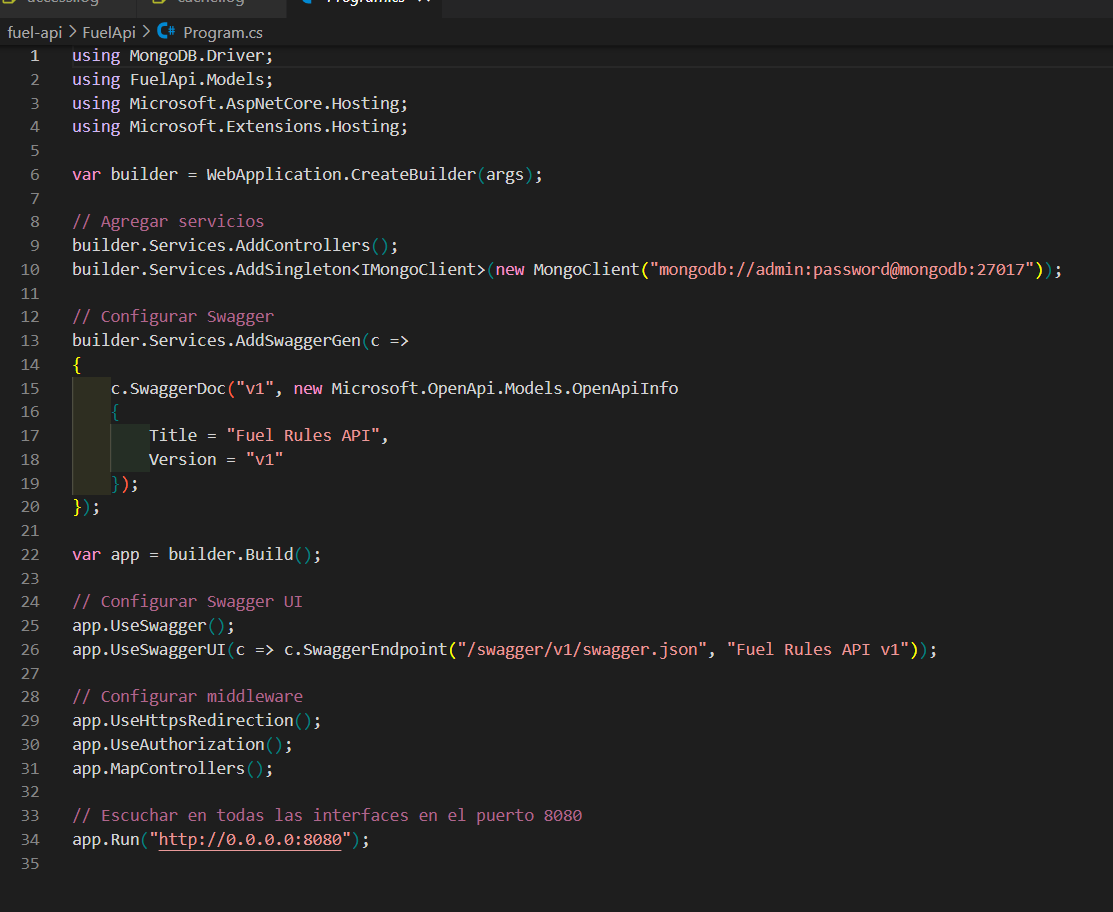


Ilustración 4. Configuración de Archivo Program.cs

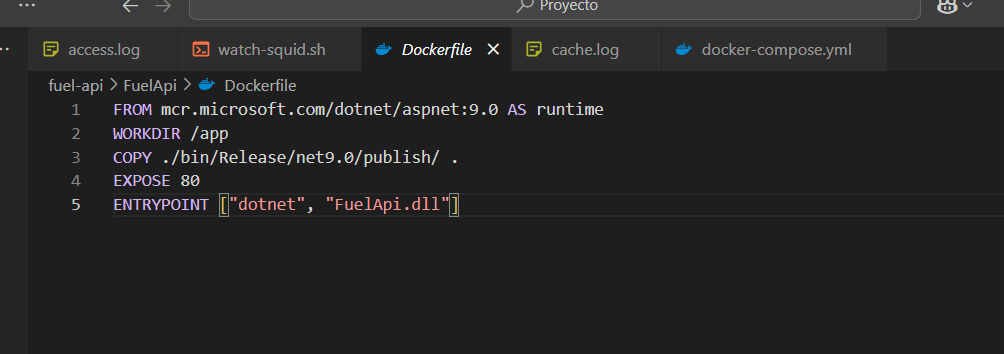


Ilustración 5. Configuración Dockerfile de la API

Para permitir el acceso externo a la API y centralizar el tráfico HTTP, se configuró un contenedor Nginx actuando como proxy inverso. El contenedor se construyó a partir de la imagen oficial nginx:latest, y se utilizó un Dockerfile sencillo que copia el archivo de configuración nginx.conf al directorio predeterminado de Nginx (/etc/nginx/conf.d/default.conf) y expone el puerto 80. En el archivo nginx.conf, se define un bloque server que escucha en el puerto 80 y redirige todas las solicitudes entrantes hacia el contenedor de la API (proyecto-fuel-api-1) utilizando la directiva proxy\_pass apuntando a http://proyecto-fuel-api-1:8080. Además, se configuran encabezados HTTP como Host, X-Real-IP, X-Forwarded-For y X-Forwarded-Proto para asegurar que la API reciba la información correcta sobre las solicitudes originales, lo que facilita la integración segura y eficiente entre el cliente y el backend.

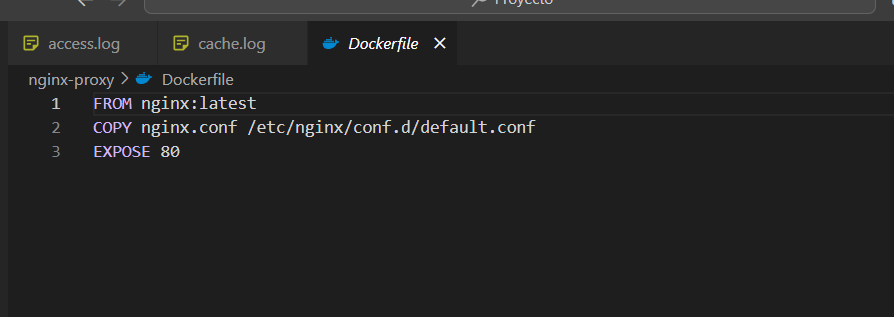


Ilustración 6. Contenedor nginx-proxy

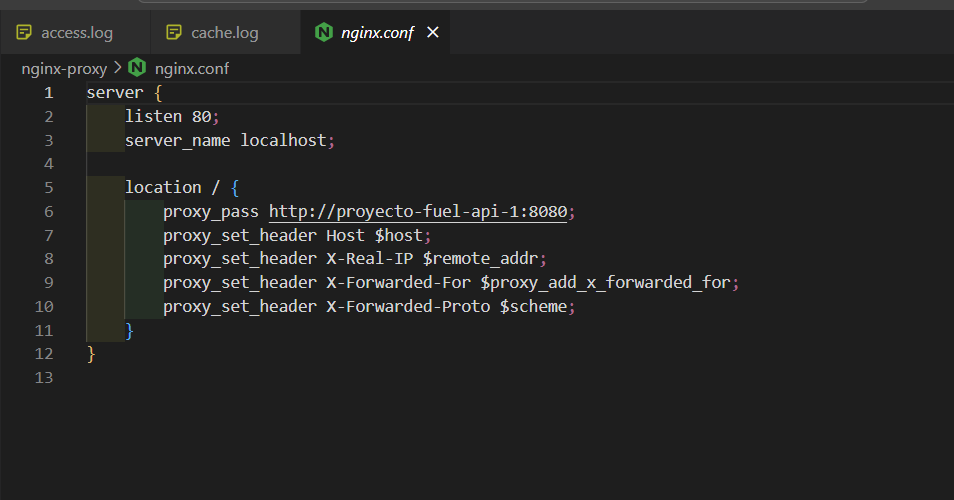


Ilustración 7. Configuración de nginx

Para controlar el acceso a sitios web y aplicar filtros de navegación, se implementó un servidor Squid Proxy dentro de un contenedor Docker utilizando una imagen basada en ubuntu:22.04. En su Dockerfile, se instalan Squid, inotify-tools (para monitoreo de archivos) y curl, además de copiarse un archivo personalizado de configuración (squid.conf) y el archivo blocked\_patterns.txt, donde se almacenan los patrones de URLs que deben ser bloqueadas. El contenedor expone el puerto 3128, correspondiente al puerto por defecto de Squid. La configuración de Squid utiliza una ACL (blocked\_urls) que carga dinámicamente los patrones desde el archivo blocked\_patterns.txt, negando el acceso a las URLs que coincidan con dichos patrones, mientras que permite el resto del tráfico. También se incluye un script (watch-squid.sh) encargado de vigilar cambios en el archivo de patrones bloqueados y recargar automáticamente la configuración de Squid. Esta integración asegura que cualquier actualización en las reglas de bloqueo definidas por la API sea aplicada de forma inmediata en el proxy sin intervención manual.



Ilustración 8. Archivo para bloqueo de rutas

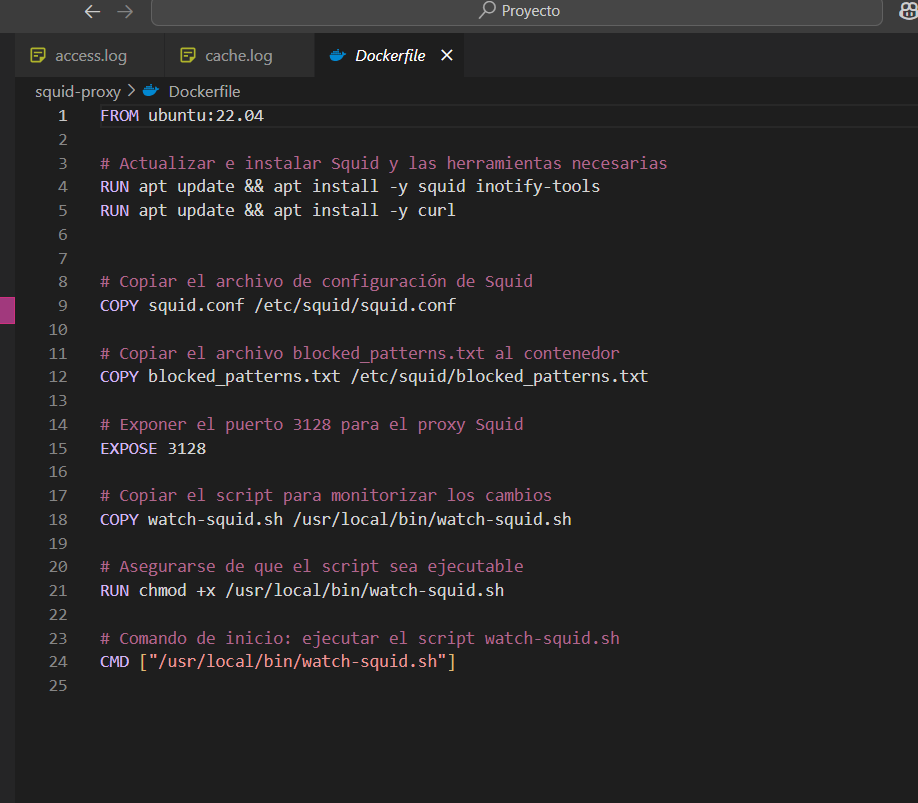


Ilustración 9. Configuración de contenedor Docker para squid-proxy

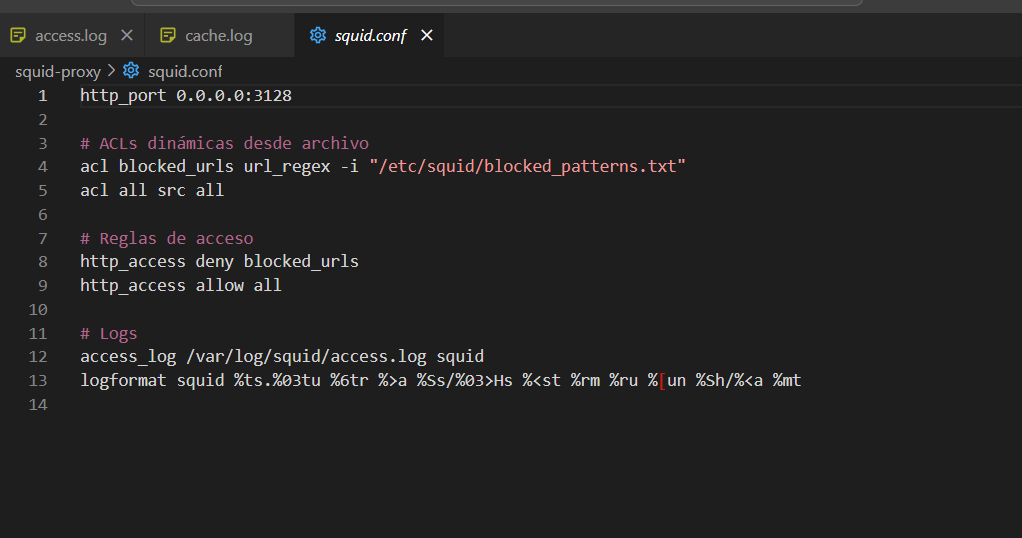


Ilustración 10. Configuración Squid

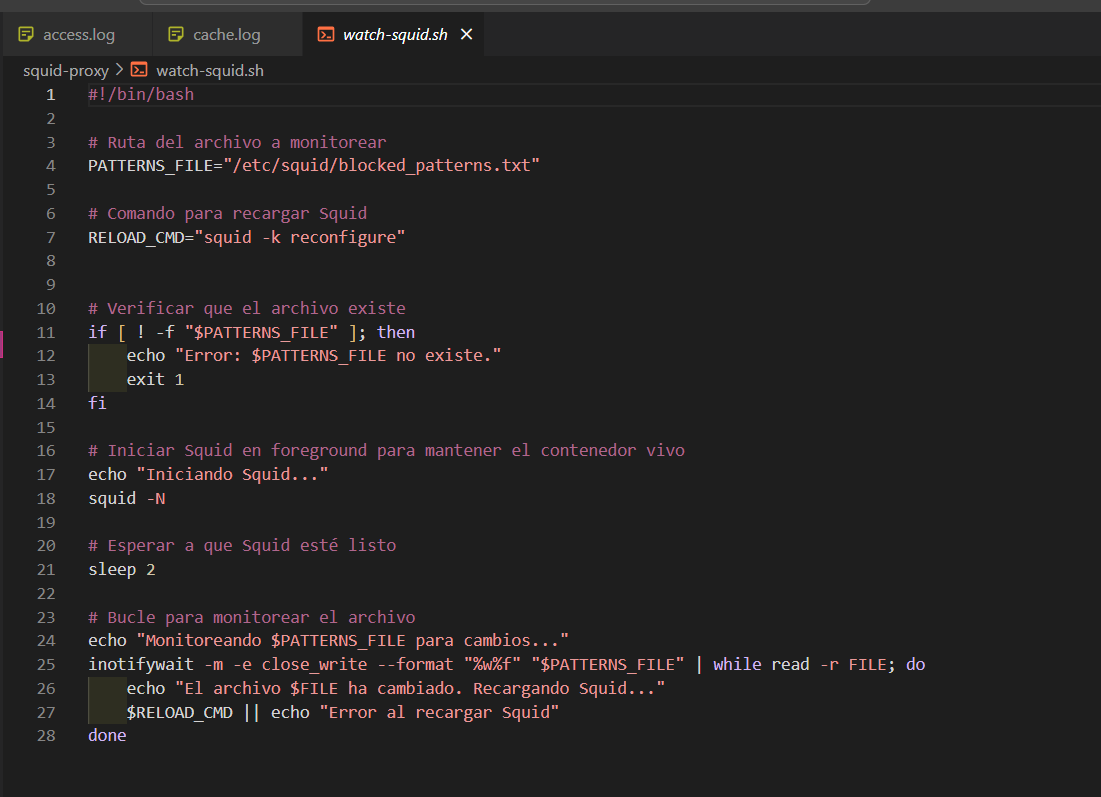


Ilustración 11. Script de automatización de cambios

El archivo docker-compose.yml define los servicios que conforman la infraestructura de la aplicación, incluyendo un Squid Proxy, una API de Fuel, un Nginx Proxy y una base de datos MongoDB. En el servicio squid-proxy, se configura el contenedor para construir la imagen desde el directorio ./squid-proxy, exponiendo el puerto 3128 y montando volúmenes para almacenar los logs de Squid y el archivo de patrones bloqueados blocked\_patterns.txt. El servicio fuel-api construye la API desde el directorio ./fuel-api/FuelApi y comparte el archivo de patrones bloqueados con Squid. El servicio nginx-proxy se encarga de redirigir las solicitudes HTTP al servicio de API en el puerto 8080, mientras que mongodb utiliza la imagen oficial de MongoDB para proporcionar la base de datos, exponiendo el puerto 27017. Todos los servicios se conectan a una red interna llamada fuel-network para facilitar la comunicación entre ellos, y se define un volumen persistente mongo-data para almacenar los datos de MongoDB. Esta configuración permite la integración entre los distintos componentes de la infraestructura de manera eficiente y aislada.

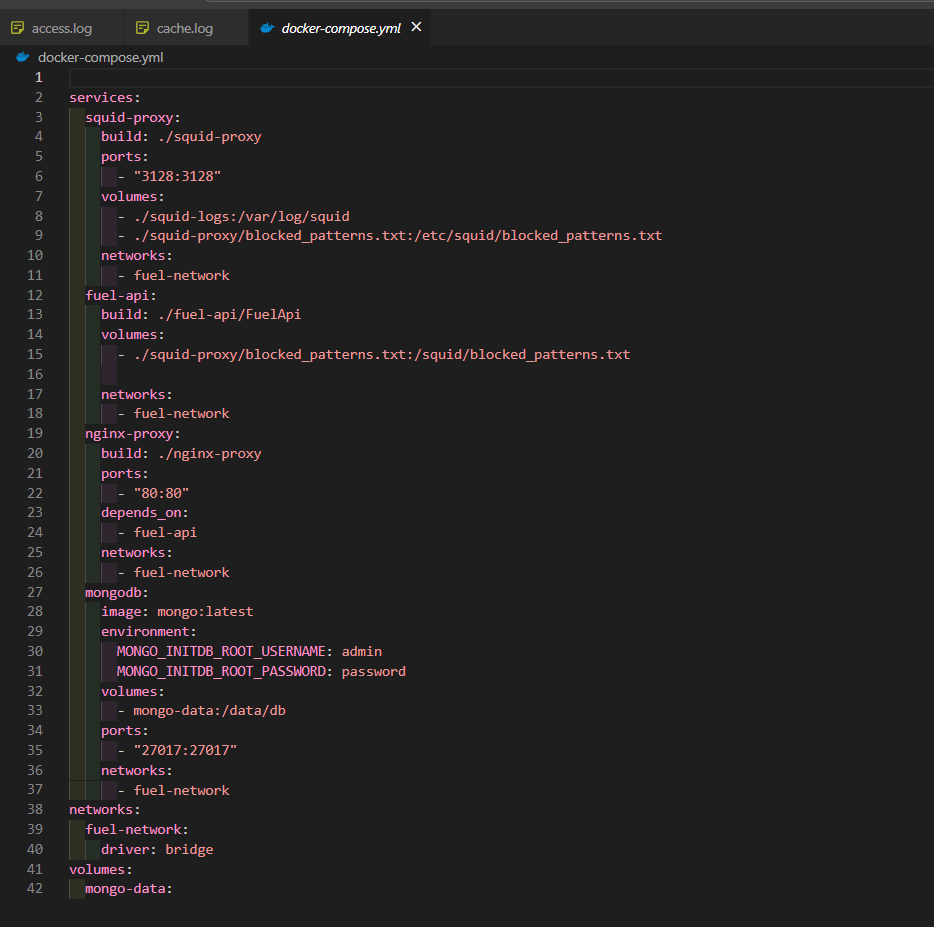


Ilustración 12. Configuración de docker-compose.yml

Se utilizó el comando docker compose up -d para iniciar los contenedores en segundo plano, creando y montando la infraestructura definida en el archivo docker-compose.yml. Al ejecutar este comando, los servicios como Squid Proxy, Fuel API, Nginx Proxy y MongoDB se desplegaron en Docker. Posteriormente, se pudo verificar la ejecución de los contenedores abriendo Docker que muestra los contenedores activos y sus respectivos puertos expuestos.

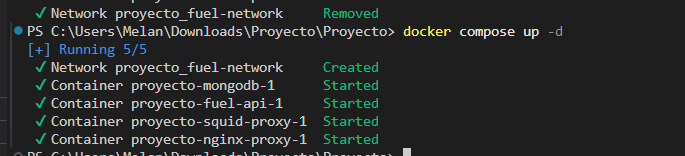


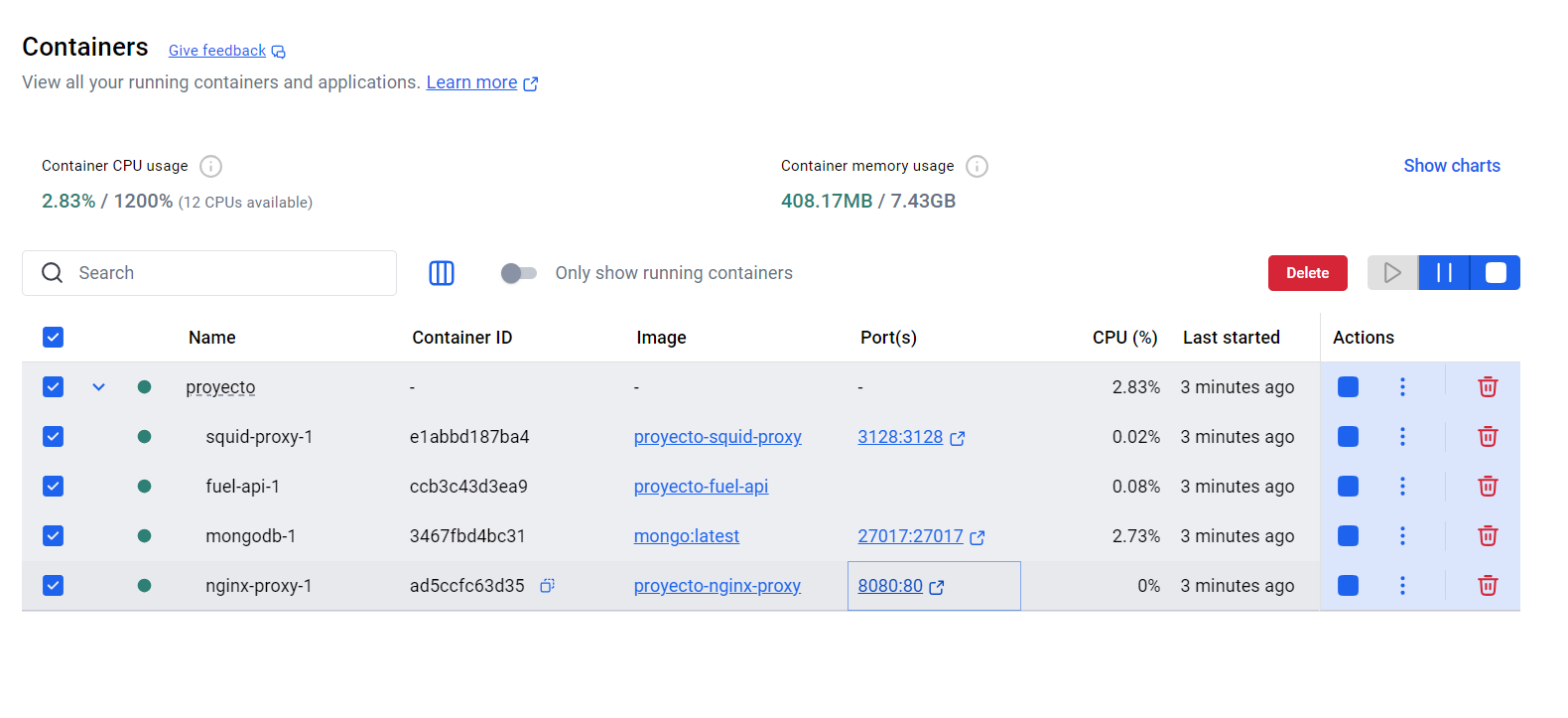
Ilustración 13. Inicialización de Docker

Ilustración 14. Contenedores en Docker

Después de levantar los contenedores utilizando docker compose up -d, pudimos acceder a la vista de Swagger para la API de Fuel en localhost:8080/swagger. Esta interfaz nos permitió interactuar de manera visual con los endpoints de la API, facilitando las pruebas y la verificación del correcto funcionamiento de los servicios expuestos.

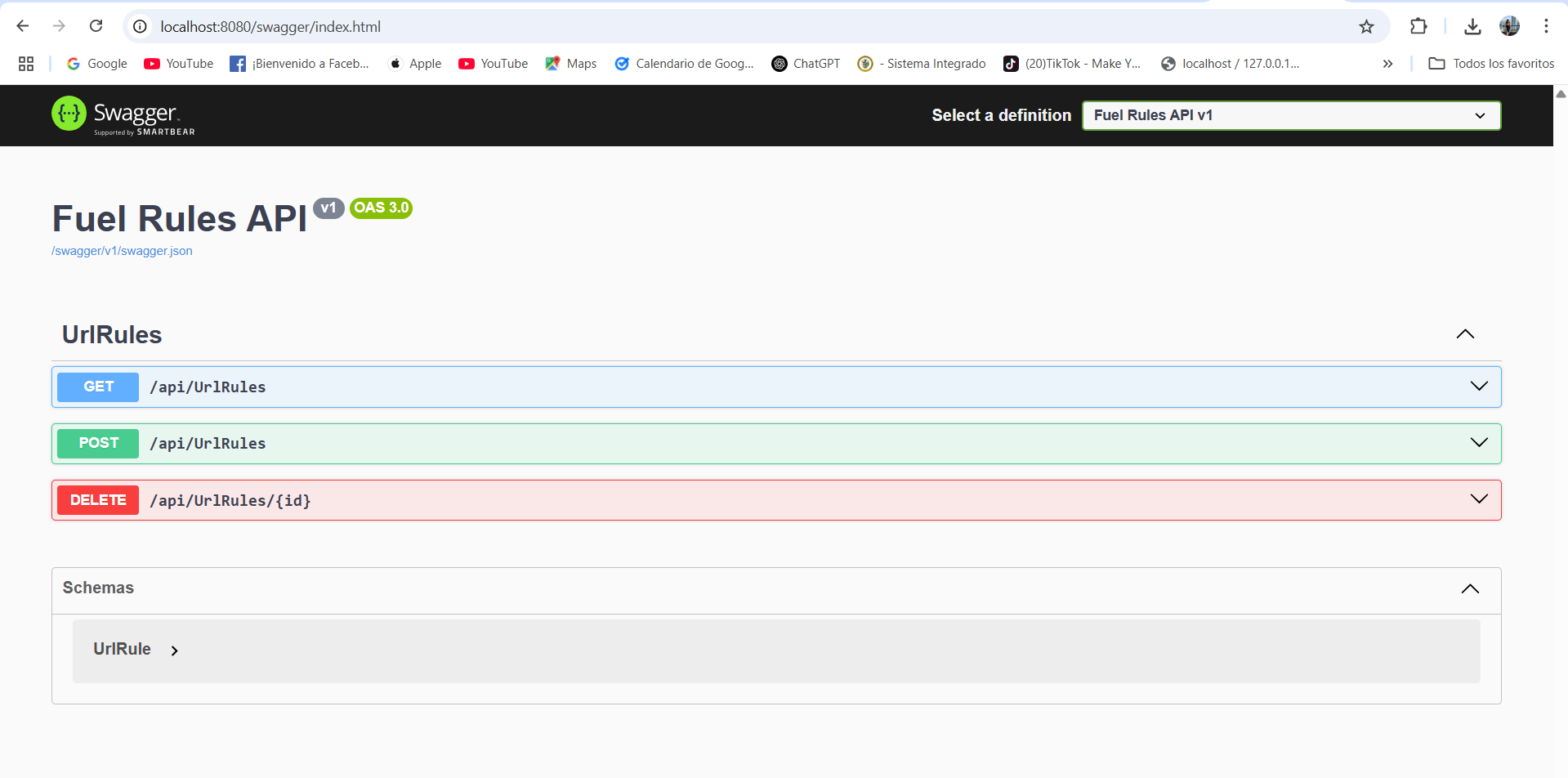


Ilustración 15. Interfaz de swagger para consumir la API

Una vez configurada la API, se ingresaron datos específicos a través de la API, como "facebook" en el patrón de bloqueo para que Squid los gestionara. Al realizar una solicitud GET a la API, se pudo visualizar correctamente la entrada con "facebook" como un patrón bloqueado, lo que permitió confirmar que el valor se había ingresado correctamente y que las reglas estaban funcionando según lo esperado para bloquear el acceso a URLs relacionadas con Facebook.

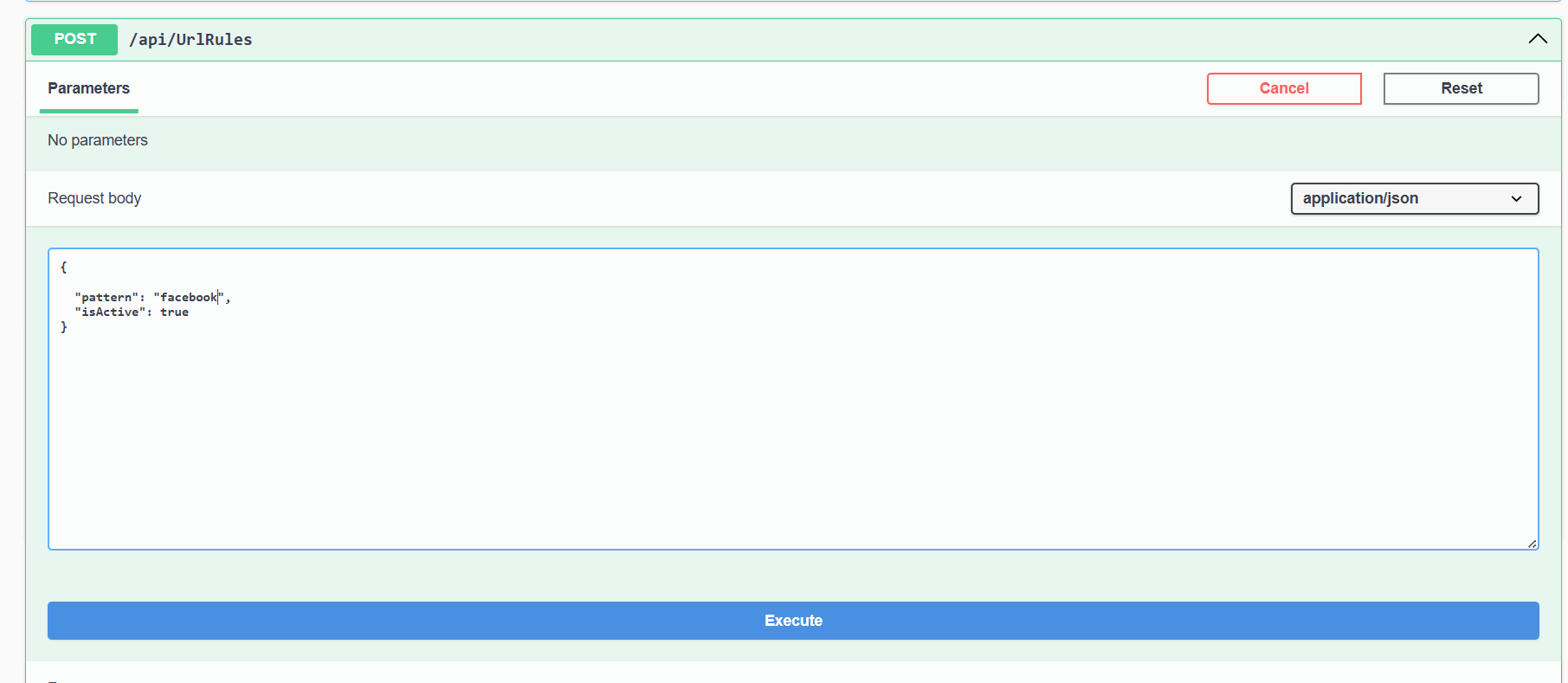


Ilustración 16. Creación de regla



Ilustración 17. Conexión correcta a la base de datos



Ilustración 18. Reglas de bloqueo de rutas ingresadas

Se verificó que el servidor proxy Squid estaba en funcionamiento accediendo a localhost:3128 desde el navegador. La respuesta del servicio confirmó su actividad.

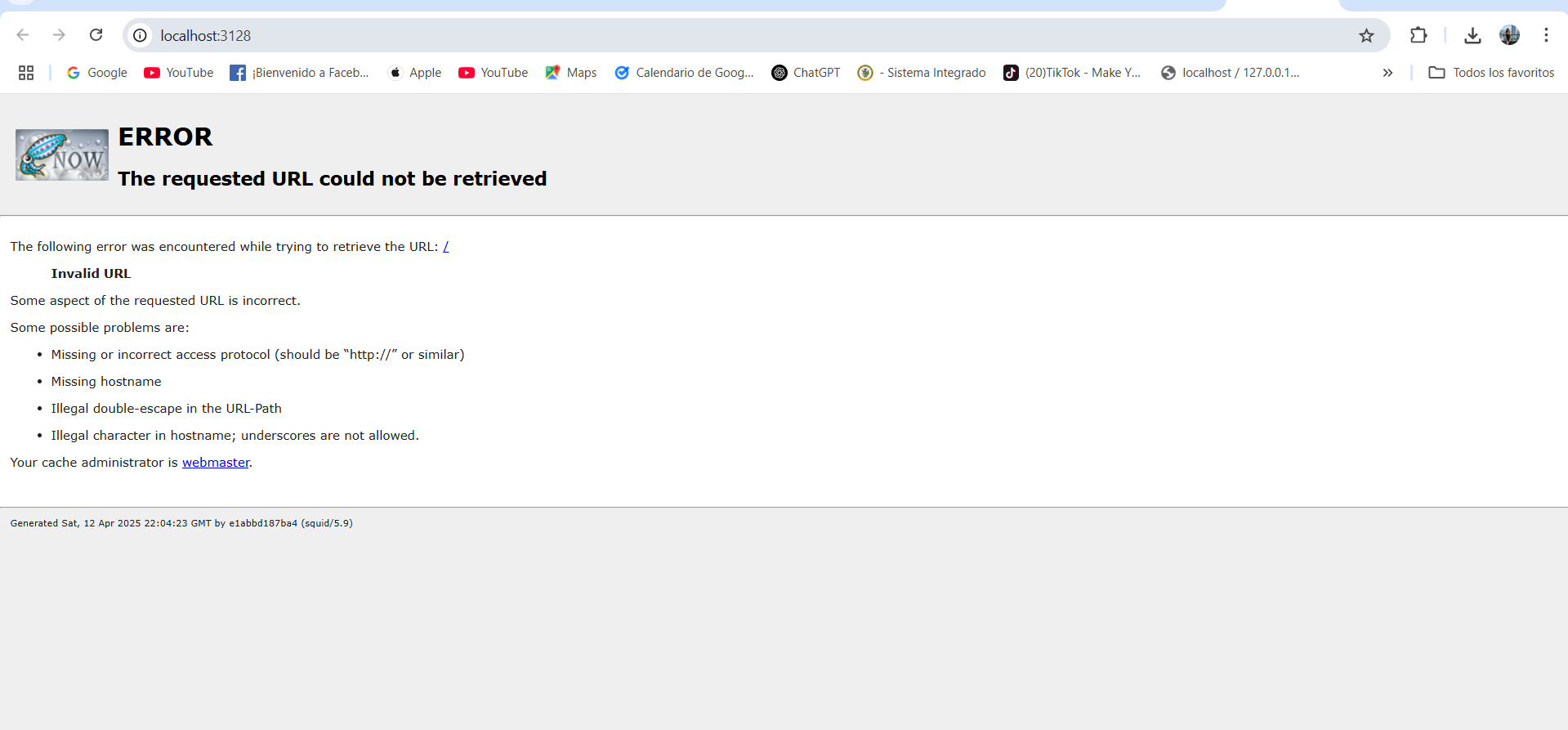


Ilustración 19. Servidor proxy funcionando

Se configuró el proxy manualmente en la máquina cliente el proxy para redirigir el tráfico web a través de localhost:3128. Esto permitió analizar las solicitudes HTTP/HTTPS registradas en los logs de Squid.

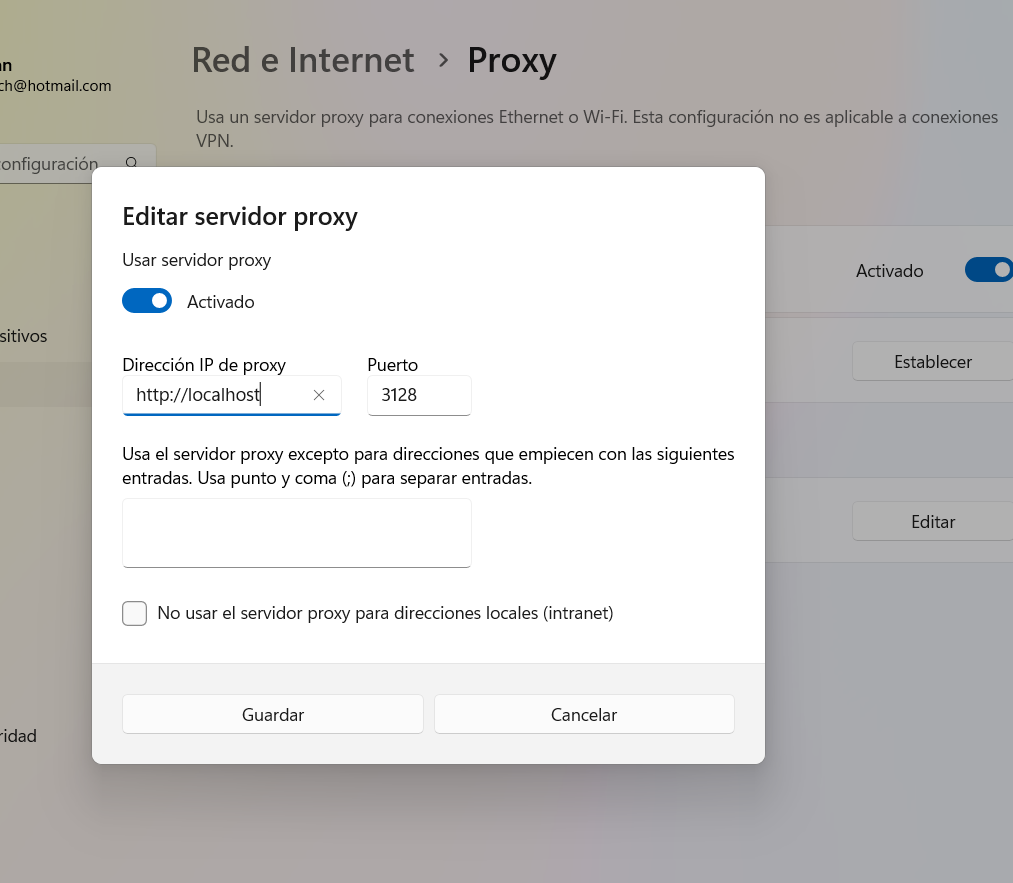


Ilustración 20. Configuración del Proxy en la maquina local

Los registros mostraron que algunas solicitudes fueron permitidas, como las de example.com (con respuestas TCP\_NISS/200 y TCP\_MEM\_HIT/200) y las de td.doubleclick.net y cdn.segment.com (con conexiones TCP\_TUNNEL/200, típicas de tráfico HTTPS); sin embargo, otras fueron bloqueadas, como las de [www.youtube.com](http://www.youtube.com/) y [www.facebook.com](http://www.facebook.com/), las cuales devolvieron un código TCP\_DENIED/403, lo que indica una denegación activa por parte del proxy conforme a sus políticas de restricción configuradas.

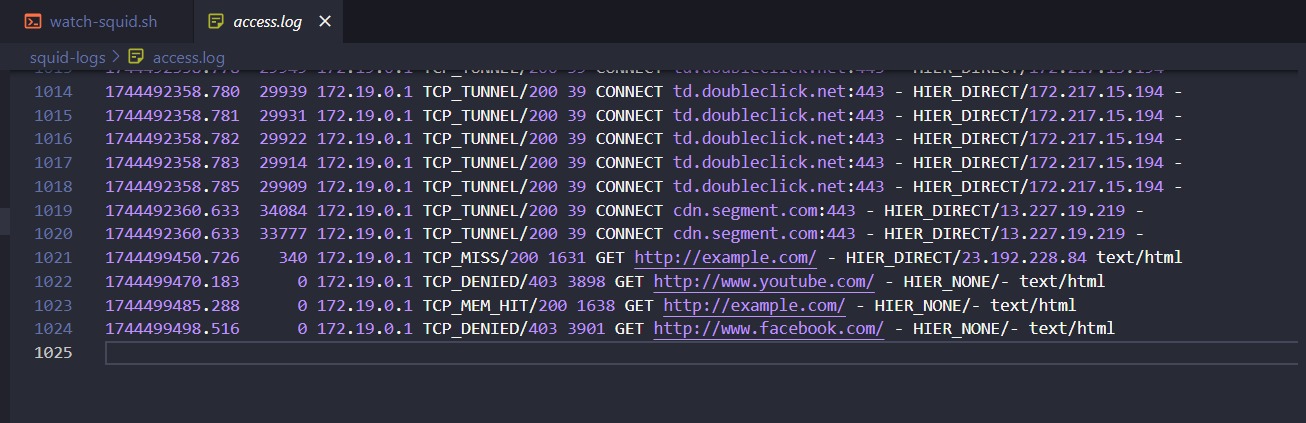


Ilustración 21. Registro de logs exitosos para bloqueos

El mensaje de error obtenido al intentar acceder a Facebook mediante curl confirma la restricción, mostrando un mensaje claro de "Access Denied" generado por Squid. Estos hallazgos evidencian una configuración intencional del proxy para bloquear el acceso a Facebook, posiblemente como parte de políticas de control de tráfico o seguridad.

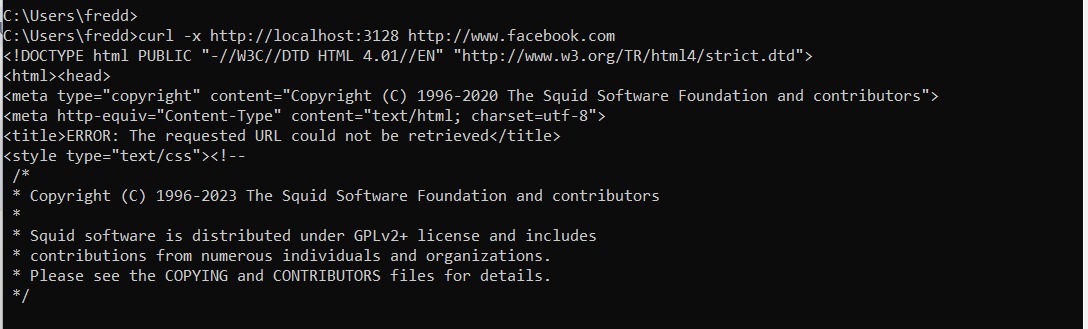


Ilustración 22. Proxy funcionando para el bloqueo de Facebook

* 1. **Habilidades blandas empleadas en la práctica**

Liderazgo

Trabajo en equipo

Comunicación asertiva

La empatía

Pensamiento crítico

Flexibilidad

La resolución de conflictos

Adaptabilidad

Responsabilidad

* 1. **Conclusiones**

La práctica permitió comprender e implementar los fundamentos de los sistemas distribuidos mediante la integración de herramientas como Docker, Squid Proxy, ASP.NET Core y MongoDB, logrando desarrollar un entorno funcional donde la API controla de manera dinámica las reglas de bloqueo de URLs. A través de esta arquitectura distribuida, se evidenció la efectividad del uso de microservicios, contenedores y redes internas para mantener una comunicación eficiente entre los componentes del sistema, cumpliendo con los objetivos planteados y demostrando una aplicación práctica de los conceptos teóricos de la asignatura.

* 1. **Recomendaciones**

Se recomienda continuar profundizando en la gestión de microservicios y contenedores, especialmente en el uso avanzado de Docker, explorando aspectos como la orquestación con Docker Compose, la optimización de imágenes y el manejo de volúmenes y redes personalizadas. Esto permitirá fortalecer la infraestructura desarrollada y mejorar la escalabilidad y mantenibilidad de los servicios. Además, aplicar este tipo de prácticas en proyectos reales o académicos más exigentes contribuirá significativamente al desarrollo de competencias técnicas y a la comprensión integral de los sistemas distribuidos.

* 1. **Referencias bibliográficas**

[**https://github.com/melanieAlban/aplicaciones-distribuidas.git**](https://github.com/melanieAlban/aplicaciones-distribuidas.git)