Servicios en Contenedores

Arturo Córdoba-Villalobos
arturocv16@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Resumen*—En este documento se presenta el diseño e implementación de un programa cliente-servidor que clasifica imágenes según su color dominante: rojo, verde o azul. El servidor se ejecuta en un contenedor de Docker mientras que el cliente se encuentra en una computadora local, por lo que se explora el concepto de contenedores y se explotan las características que ofrece Docker para la creación y modificación de imágenes.**

***Palabras clave*—Docker, contenedores, virtualización, sistemas operativos, sockets, C.**

I. Introducción

En este documento se presentará el ambiente de desarrollo, diseño e instrucciones de uso de un programa cliente-servidor cuyo objetivo es enviar imágenes al servidor para que este las clasifique según su color dominante: rojo, verde o azul. Según su clasificación la imagen será almacenada en una carpeta exclusica para su color: R (rojo), G (verde), B (azul). El servidor contará con una lista de direcciones ip confiables y no confiables, las imágenes enviadas por una fuente no confiable no recibirán ningún tipo de procesamiento, únicamente serán almacenadas en una carpeta exclusiva para este tipo de clientes ('Not trusted'). Si un cliente no se encuentra en esta lista de direcciones conocidas su conexión con el servidor no será aprobada. El servidor está diseñado para que sea ejecutado en un contenedor de Docker. Los directorios donde se almacenan las imágenes son exclusivos de cada contenedor, por lo que al crear uno se crea una carpeta exclusiva para él en caso de que conviva con más contenedores.

Docker es un proyecto open source que permite la creación de contenedores que aíslan una aplicación en su propio ambiente controlado. Docker utiliza el kernel de Linux para aislar procesos, de manera que estos puedan ejecutarse de manera independiente. Está basado en la construcción de imágenes, las cuales permiten compartir aplicaciones o servicios con todas sus dependencias, facilitando la creación e implementación en diferentes entornos. La imagen es el conjunto de instrucciones necesario para llevar a la vida a un contenedor, en el cual se ejecutan las aplicaciones [1].

II. Ambiente de Desarrollo

En el desarrollo del proyecto se utilizaron diferentes herramientas para llevar a cabo el diseño e implementación de la solución al problema planteado. Como sistema operativo se utilizó Xubuntu, debido a que Docker funciona de manera nativa en sistemas operativos basados en Linux. Se utilizó el lenguaje de programación C, el cual fue compilado con la ayuda del GNU Compiler Collection (GCC), cuyas instrucciones fueron automatizadas con un makefile para simplificar este proceso. Se utilizó Visual Studio Code para la escritura y prueba de los archivos fuente desarrollados. Docker fue la pieza clave de la implementación de este proyecto, ya que esta herramienta permite crear y gestionar contenedores de una manera simplificada y sencilla, permitiendo un despliegue de aplicaciones de software que se encuentren aisladas dentro de su propio ambiente, exponiendo solo aquellas características que sean necesarias. Docker permite crear imágenes personalizadas a partir de un archivo denominado Dockerfile, en el cual se listan todas las instrucciones que deben ser ejecutadas para configurar el ambiente que necesita una aplicación para funcionar de la manera esperada, por lo que se utilizó esta característica para el desarrollo de la solución, utilizando como imagen base una imagen de Ubuntu 20.04. Para el algoritmo de color solicitado se utilizó la biblioteca libpng, la cual permite el procesamiento de imágenes cuyo formato es exclusivamente PNG. Para la conexión entre el cliente y el servidor se utilizó la bibioteca nativa de C para el desarrollo de sockets. Se utilizó Git para llevar un control de versiones del código desarrollado, en conjunto con GitHub y GitKraken, este último permite el manejo de Git a través de una interfaz gráfica.

III. Atributos

Durante el desarrollo de este proyecto se reforzaron principalmente dos atributos. En primer lugar, el aprendizaje continuo. Se adquirió el conocimiento acerca de qué es Docker, cómo funciona y qué lo caracteriza. Además, se aprendieron los pasos necesarios para realizar su configuración y despliegue, cómo crear imágenes, cómo crear contenedores, y cómo éstos se gestionan. Se profundizó el conocimiento sobre la sintaxis, manejo de memoria, creación de sockets y gestión de archivos en el lenguaje de programación C. En segundo lugar, se reforzó el atributo de herramientas de ingeniería. Se aplicó el conocimiento sobre sockets para la comunicación entre un cliente y un servidor, donde éste último se ejecutó dentro de un contenedor. Se aprendió a configurar el editor de código Visual Studio Code para gestionar las imágenes y contenedores de Docker, permitiendo una gestión más accesible.

IV. Diseño

En esta sección se detallará el diseño del cliente y servidor. Se utilizarán diagramas de flujo para el servidor y el cliente, y se explicará cada una de las secciones que los componen, luego, se mostrará la interacción entre ambos a través de un diagrama de secuencia, donde se explica el proceso de conexión y envío de la imagen, y cuál es la interacción del usuario en el proceso.

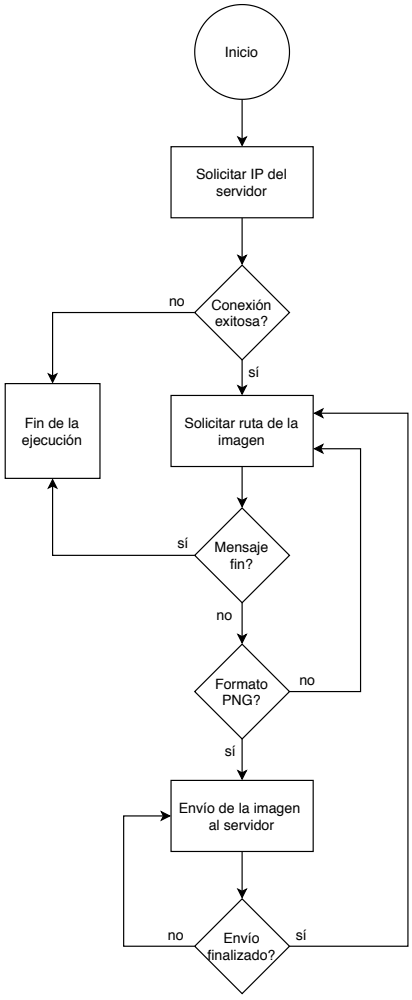

Figura 1: Diagrama de flujo para el cliente

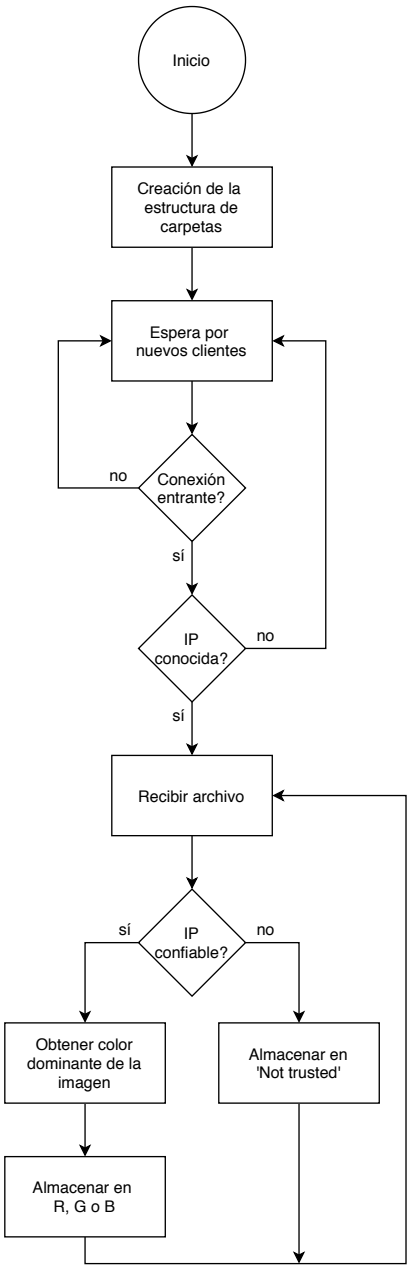

Figura 2: Diagrama de flujo para el servidor

En la figura 1 se muestra un diagrama de flujo que representa los pasos que realiza el algoritmo del cliente. Cuando se inicia el programa se le solicita al usuario que ingrese la dirección ip del servidor y se intenta realizar la conexión, si esta es exitosa se procede a solicitar la ruta de la imagen que se desea enviar, en caso contrario el programa finaliza. Una vez ingresada la ruta de la imagen, se verifica si el usuario escribió la palabra *fin*, ya que esta es una palabra reservada para detener el programa. Posteriormente, se verifica si el archivo que se desea enviar tiene una extensión *.png*, si es así se procede con el envío, si no, se vuelve a solicitar la ruta de la imagen. El envío de la imagen se realiza por paquetes, por lo tanto, el algoritmo verifica si el envío ha sido completado antes de volver a solicitar una nueva imagen.

En la figura 2 se muestra un diagrama de flujo que representa los pasos ejecutados por el algoritmo del servidor. Cuando se ejecuta el programa, se crea la estructura de carpetas del programa, la cual se explicará más adelante. Después de la creación de las carpetas, el servidor espera la conexión de un nuevo cliente, una vez recibida, se verifica si la dirección ip es confiable o no según lo establecido en un archivo denominado *configuracion.config*. Posteriormente, se recibe el archivo enviado por el cliente, y se procede según la confiabilidad de la dirección ip: si es confiable, se procesa la imagen para determinar el color dominante, y esta se almacena en la carpeta correspondiente; si la ip no es confiable, la imagen no es procesada, y es almacenada en una carpeta exclusiva para este tipo de clientes. Finalmente, el servidor

espera por un nuevo archivo proveniente del mismo cliente. Si en algún punto del proceso se pierde la conexión con el cliente, entonces el servidor volverá a esperar por una nueva conexión.

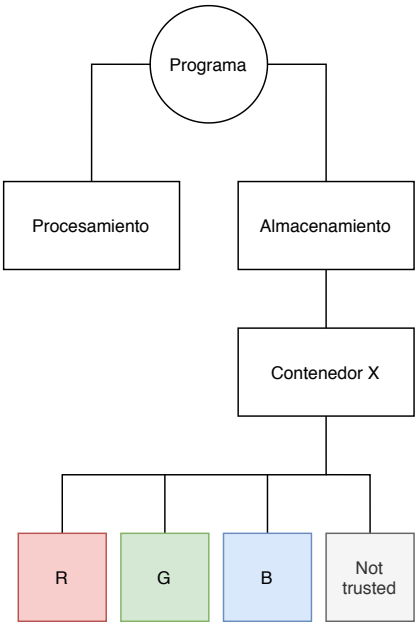

Figura 3: Estructura de carpetas del servidor

En la figura 3 se muestra un diagrama con la estructura de carpetas del servidor. Se tiene una carpeta para imágenes que deben ser procesadas denominada Procesamiento, las imágenes que son enviadas por el cliente son almacenadas aquí, y luego de ser procesadas son movidas a su carpeta correspondiente. Se tiene una carpeta denominada Almacenamiento, la cual contiene a todas las carpetas correspondientes a cada contenedor creado a partir de la imagen generada con el Dockerfile. Estas carpetas llevan por nombre Contenedor X, donde la X representa el número de contenedor, empezando con el 1 y sumando una unidad cada vez que se crea un nuevo contenedor que comparta el volumen de Almacenamiento. Dentro de cada carpeta de un contenedor, se encuentran 4 subcarpetas, donde se almacenan las imágenes con color predominante rojo (R), verde (G), azul (B), o aquellas que vienen de un cliente no confiable (Not trusted).

En la figura 4 se muestra un ejemplo del archivo *configuracion.config*, la primera línea tiene que contener el título *trusted hosts*, el cual debe ser seguido por todas las direcciones pertenecientes a esta categoría. Después de las direcciones confiables, debe existir un espacio en blanco seguido por el título *not trusted*, después de esto se deben listar todas las direcciones que no sean confiables.

En la figura 5 se muestra un diagrama de secuencia que explica la interacción que existe entre el usuario, el cliente y el servidor. El usuario ejecuta el cliente e ingresa la dirección ip y puerto en el que se encuentra el servidor, posteriormente, el cliente intenta realizar una conexión utilizando esta información. Una vez establecida la conexión, el usuario ingresa la ruta de la imagen que desea enviar, el cliente realiza una verificación utilizando el nombre de la imagen

```
trusted hosts
192.168.1.14
192.168.1.12
127.0.0.1
192.168.1.17
192.168.1.9
172.17.0.1

not trusted
192.168.1.11
192.168.1.10
192.168.1.3
```

Figura 4: Estructura archivo configuracion.config

para conocer si el archivo tiene extensión *.png*, ya que solo se acepta este formato. Realizada la verificación, se envía un mensaje al servidor con el nombre y tamaño del archivo, para posteriormente iniciar con la transferencia del mismo, el cual se divide en paquetes de 1023 bits y son enviados uno por uno. Cada vez que se envía un paquete, el cliente espera la respuesta del servidor, donde se indica si la transferencia de la información fue exitosa o si ocurrió algún error y se necesita que el cliente envíe nuevamente el paquete. Cuando se completa el envío del archivo, entonces el cliente le indica el servidor que la transferencia a finalizado, y espera la respuesta del servidor con la clasificación de la imagen, para informar al usuario del resultado.

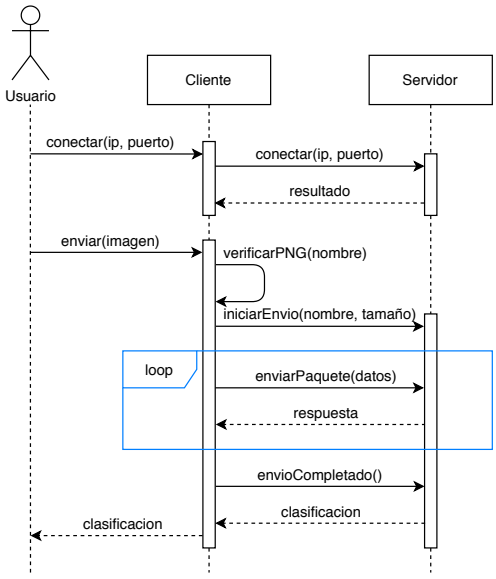

Figura 5: Diagrama de secuencia del sistema

V. Instrucciones de uso

En esta sección se explicarán las instrucciones de uso para el programa. Todas estas instrucciones deben ser ejecutadas en una terminal de Linux. Utilizando el archivo *makefile* se puede compilar el código fuente para poder ejecutarlo, para realizar esto se debe abrir una terminal en la ruta en la que se encuentra este archivo y escribir el comando *make*. Se necesita tener instalado GCC y Docker.

V-A. *Servidor*

Después de compilar el archivo *server.c*, si se desea ejecutar el servidor en la máquina local se debe abrir una terminal en la ruta en la que se encuentra el archivo compilado, y se debe ingresar el comando *./server* para iniciarlo. Si se desea ejecutar en un contenedor de Docker, es necesario crear la imagen a partir del archivo Dockerfile, para lo cual se debe ejecutar el siguiente comando:

docker build -t [nombreImagen] .

donde [nombreImagen] corresponde al nombre deseado para la imagen. Posteriormente, se debe crear un contenedor a partir de esta imagen, para lo cual se debe ejecutar el siguiente comando:

docker run -p [pLocal]:8080 -v [rutaLocal]:/psot1-dstorage/ –detach –name [nombre] [nombreImagen]

En este comando [pLocal] es el puerto al que será mapeado el puerto 8080 del contenedor, [rutaLocal] es la ruta donde se desea montar el volumen del contenedor, [nombre] corresponde al nombre del contenedor y [nombreImagen] al nombre de la imagen seleccionado en el paso anterior.

El servidor necesita un archivo de configuración donde se listen las rutas confiables y no confiables, cuya estructura se muestra en la figura 4, el nombre de este archivo debe ser *configuracion.config*, y se debe ubicar en la misma carpeta en la que se encuentra el archivo compilado y el Dockerfile. Si el servidor es ejecutado desde un contenedor, es necesario copiar el archivo después de crear el contenedor, para lo cual se debe ejecutar el siguiente comando:

docker cp configuracion.config [nombre]:/configuracion.config

El espacio [nombre] corresponde al nombre asignado al contenedor. Una vez terminado este proceso, el servidor se encuentra listo para recibir imágenes.

V-B. *Cliente*

Después de compilar el archivo *client.c*, se debe abrir una terminal en la ruta en la que se encuentra el archivo compilado, y se debe ingresar el comando *./client [direccionIP]* para ejecutarlo, donde [direccionIP] es la dirección en la que se encuentra el servidor. Posteriormente, el programa le solicitará al usuario ingresar el puerto del servidor. Una vez realizados estos pasos, se podrá ingresar la ruta de la imagen a enviar, la cual es relativa a la carpeta donde se encuentra el archivo compilado. Una vez enviada la imagen, se mostrará en la terminal el resultado de la clasificación, y se podrá enviar una nueva imagen.

VI. BITÁCORA

En la Tabla I se muestran las actividades, fecha y duración invertida en cada etapa del desarrollo de la solución.

Actividad	Fecha	Duración (Horas)
Investigación Docker.	08/9/2020	2
Investigación Docker.	09/9/2020	2
Repaso programación en C. Creación de un servidor y un cliente básicos con sockets.	10/9/2020	3
Funcionamiento de la bibllioteca libpng.	11/9/2020	3
Obtener cada canal de una imagen y calcular el más predominante. Lectura y escritura de archivos binarios. Envío de archivos binarios a través de sockets.	12/9/2020	6
Funciones para la creación de los directorios que necesita el servidor.	14/9/2020	1
Clasificación y almacenamiento en el servidor según el canal dominante de una imagen.	15/9/2020	3
Creación del Dockerfile y pruebas de su funcionamiento.	16/9/2020	2
Pruebas del servidor en la imagen creada con el Dockerfile, conexión del cliente con el servidor en el contenedor.	17/9/2020	2
Validación de direcciones ip con el archivo configuracion.config	18/9/2020	2
Manejo de desconexiones durante el proceso de envío	19/9/2020	2
Documentación	20/9/2020	2
Documentación	21/9/2020	2
Total		32

Tabla I: Bitácora de trabajo

VII. CONCLUSIONES

Herramientas como Docker facilitan el control de los ambientes en los que se desarrollan o ejecutan aplicaciones, ya que simplifica la instalación de dependencias y replicación de estos ambientes. La capacidad de tener un ambiente de desarrollo, pruebas y producción, cada uno aislado del otro, es una ventaja para el programador ya que le permite ahorrar tiempo y equipo, además, tener la capacidad para replicarlos siempre que desee, inclusive en diferentes ambientes, a través de la simplicidad de Docker es una ventaja increíble.

La utilización de Docker para el despliegue del servidor, y la conexión con un cliente que se ejecuta en la máquina local, fue una experiencia enriquecedora para el proceso de aprendizaje, ya que permitió un contacto de primera mano con herramientas de virtualización de procesos, lo que es importante para un Ingeniero en Computadores.

REFERENCIAS

[1] Red Hat. (Sin fecha) Contenedores: ¿Qué es Docker? Consultado en https://www.redhat.com/es/topics/containers/what-is-docker

[2] Serrano, I. (2017). Curso de Docker Gratis [Archivo de video]. Consultado en https://www.youtube.com/playlist?list=PLrb1e2Mp6N_tXQryuDVzOq4SLQKqVv1uz

[3] Edureka! (2018). Dockerfile Tutorial with Example — Creating your First Dockerfile — Docker Training — Edureka [Archivo de video]. Consultado en https://youtu.be/2lU9zdrs9bM