Verificador de colores

Esteban Alvarado Vargas*, Olman Castro Hernández†, Martín Calderón Blanco‡
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica
Email: * estalvgs1999@estudiantec.cr, † olcastro@estudiantec.cr, ‡ martinrolo22@estudiantec.cr,

***Abstract*—The virtualization of services has been something very useful in Software Services, those will be given in the most in case for Cloud services. For this project, the idea is to explore a Virtual Machine and a Container and use the together to see performance and how they can interact together. This project is a simple Client Server an algorithm that count pixels in an image. The pixels have been higher than a user input value. The project was implemented with a Client in Docker and a Server in an Amazone Web Server.**

***Palabras clave*—Containers, VMs, Virtual Machine, Docker, AWS, C lenguage, Demons**

I. Introducción

Gracias al cloud computing ha sido posible generar avances importantes en los recursos informáticos y la democratización de estos. Esta técnica consiste en proporcionar infraestructura informática, de red, almacenamiento, servicios, plataformas y aplicaciones en general a los usuarios de cualquier red. Brindando los recursos necesarios para desarrollar desde cualquier parte del mundo. [1]

Dentro de estos servicios podemos mencionar uno de los más nos incumbe en este proyecto, la virtualización. La virtualización consiste en crear múltiples entornos simulados o recursos dedicados en un solo sistema de hardware físico; por medio del hipervisor se consigue una conexión directa al hardware, permitiendo los entornos separados mejor conocidos como máquinas virtuales (**VM**). [1] Este tipo de interacciones con el hardware, son controladas mediante algún tipo de sistema operativo (por lo general alguno basado en Linux). Un sistema operativo consiste en una serie de programas informáticos que gestionan los recursos del hardware. [2]

Así como la virtualización y las **VM**, existen otro tipo de herramientas para los ambientes de software como lo son los contenedores. Estos, al igual que su nombre en la cotidianidad, se encargan de empaquetar y resguardad el software, ambientes de desarrollo y entre otros; con el fin de que estos no se vean alterados y las dependencias de otro tipo de recursos no alteren el correcto funcionamiento del entorno que existe dentro del contenedor tal como muestran las Fig. 1 y 2. [3]

Inmersos en los sistemas **UNIX**, existe programas que son ejecutados en segundo plano para ejecutar acciones específicas, estos no poseen una interfaz gráfica con el fin de que quedan fuera del alcance del usuario y son llamados demonios o servicios en el caso de Windows u otros sistemas operativos. Los demonios son utilizados y creados por medio de SysVinit o Systemd. [4]

En el presente proyecto, se expondrá al lentor una arquitectura cliente-servidor encargada del envío, procesamiento y

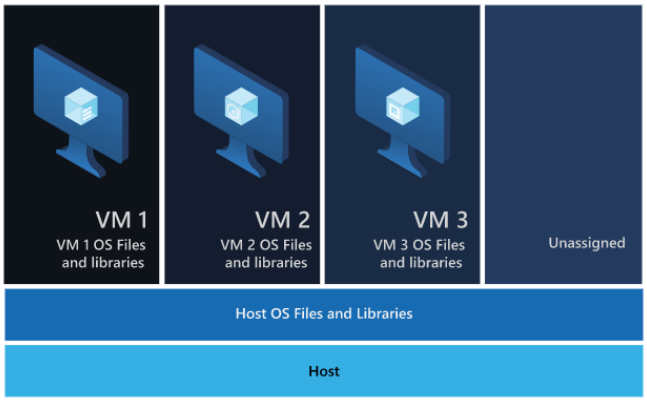

Fig. 1: Diagrama de funcionamiento Máquinas Virtuales [3]

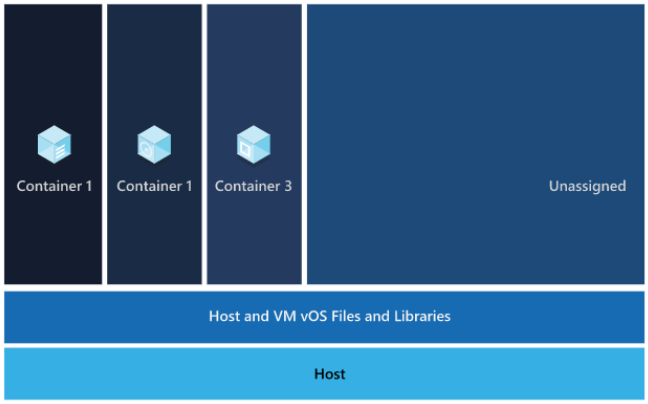

Fig. 2: Diagrama de funcionamiento Contenedores [3]

almacenamiento de imágenes de cualquier tamaño y formato donde:

- El servidor se encontrará en una máquina virtual y estará encargado de recibir las imágenes de manera secuencia, además, poseerá un demonio que se ejecutará al inicio del sistema para cargar todas sus dependencias. Finalmente, este albergará las funciones de start, stop, status, restart y Stop(SO).
- El cliente estará alojado en un contenedor tipo Docker. Este realizará el envío de imágenes de manera secuencial de cualquier tamaño, esta que reciba la entrada de “end” por parte del usurario, además mostrará la respuesta emitida por el servidor.

II. Ambiente de desarrollo

Durante el desarrollo de este proyecto se constituyó un ambiente de desarrollo con procedimientos, herramientas y bibliotecas para escribir, generar, probar y desplegar el sistema.

Los programas se encuentran escritos en lenguaje C. Para la compilación, construcción y generación de los Makefile de los proyectos se empleó la herramienta CMake.

Por la naturaleza web del proyecto, implementamos bibliotecas que nos permitieran aprovechar sus funcionalidades para realizar peticiones (*libcurl*), desplegar un servidor HTTP (*ulfius*), codificar y decodificar archivos en base 64 (*libb64*) y para trabajar con archivos de imagen (*stbimage*).

Para la contenerización del cliente, se construyó una imagen para Docker y este mismo para correr un contenedor en la máquina local. El servidor corre en una máquina virtual con el sistema operativo Ubuntu 20.04, desplegada en Amazon Web Services. Para que el servicio de imágenes se ejecute como un demonio en el servidor se utilizó *systemd* y se escribió un archivo de configuración para el servicio.

Para el manejo de versiones se usó GitHub. En el repositorio se encuentran: el código fuente del cliente y del servidor con los archivos necesarios, así como las instrucciones para correr los programas.

III. Atributos

La presente sección muestra los atributos de ingeniería desarrollados en el proyecto, así como los principales temas que se abarcaron.

A. Aprendizaje continuo

Con el fin de lograr alcanzar los objetivos necesarios para el proyecto, se debió de investigar acerca de las máquinas virtuales ofrecidas por los principales centros de Cloud Computing. Así como:

- Contenerización.
- Máquinas virtuales en **AWS** (Amazon Web Services).
- Mejores prácticas y uso del lenguaje C, así como el uso de punteros y espacios en memoria.
- Procesamiento de imágenes en C, Servidores Web en C, así como el uso de demonios.

Por lo que cada estudiante, según la sección asignada del proyecto, realizó una investigación para obtener los conocimientos necesarios en cada una de estas áreas y brindo ejemplos a los compañeros de como debía de desarrollarse cada una de ellas.

B. Herramientas de Ingeniería

Las herramientas de ingeniería utilizadas en el proyecto son:

- Contenedores con Docker.
- Máquinas virtuales en **AWS**.
- Bibliotecas para el manejo y comunicación del cliente con el servidor: `libcurl`, `ulifus`, `libb64`.
- Manejo de imágenes con la biblioteca de manejo de imágenes para C: `stb image`.
- Compilación y construcción del sistema mediante CMake.

IV. Detalles del diseño

A. Diagrama de Arquitectura

La arquitectura usada es la de cliente-servidor, en la cual el cliente se encuentra en un contenedor de Docker corriendo en la máquina local. Por otra parte, el servidor se ejecuta en una máquina virtual, tal como se muestra en la Figura 3. Además, se usa la herramienta `systemd` para que el programa corra como un servicio demonio en el sistema, por lo tanto, cada vez que se reinicie el servidor, *ImageServer* comenzará a ejecutarse. La máquina virtual de Linux es proporcionada por AWS, mientras que el cliente se ejecuta de forma Local.

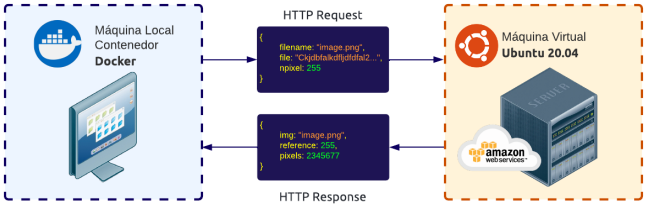

Fig. 3: Diagrama de arquitectura

B. Diagrama de Secuencia

El algoritmo de funcionamiento del proyecto, se muestra en la Figura 4, tal como se observa, la consola solicita al usuario, la IP y el puerto del servidor, el nombre del archivo de imagen que desea procesar, y el valor del píxel de referencia como parámetro para ser enviados como un formulario al servidor. El controlador recibe los datos de la solicitud del cliente. Los datos del archivo de imagen son almacenados en el servidor, se cuenta la cantidad de píxeles que son mayores al valor proporcionado por el usuario, para finalmente devolver la cantidad total de píxeles al usuario por medio del controlador y mostrarlo en la consola.

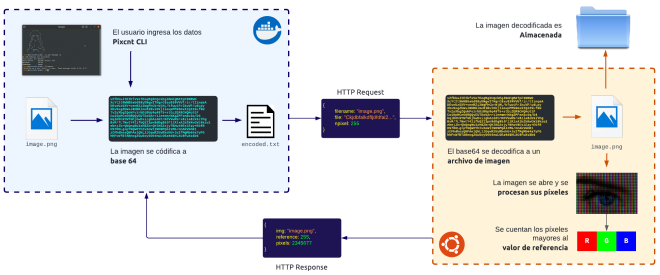

Fig. 4: Algoritmo de conteo de píxeles

El diagrama de secuencia (Figura 5) presenta el flujo de los procesos y la interacción de los módulos.

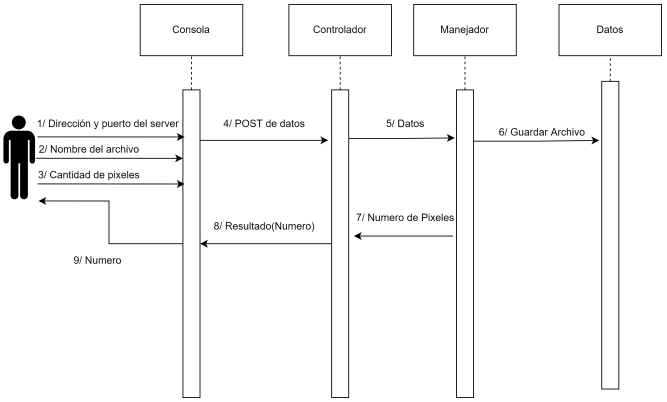

Fig. 5: Diagrama de secuencia

C. *Diagrama de componentes*

El sistema consta de cuatro módulos o capas: interfaz de usuario, presentación, manejo y datos. La Figura **??** presenta el diagrama de componentes del sistema. La interfaz maneja la entrada del usuario y los envía a la capa de presentación, que recibe el formulario con los datos y pasa el nombre del archivo, su contenido y el valor de referencia a la capa de manejo. La capa de manejo se encarga de decodificar la imagen y pasa el archivo a la capa de datos para que almacene la imagen en memoria. Luego, la capa de manejo realiza el procedimiento para contar los píxeles y pasa el resultado a la capa de presentación para que envíe la respuesta a la capa de interfaz y esta pueda mostrarlo en la consola.

Además, se desarrollaron dos módulos adicionales, uno para leer el archivo de configuración del servidor y otro para mantener un sistema de logs.

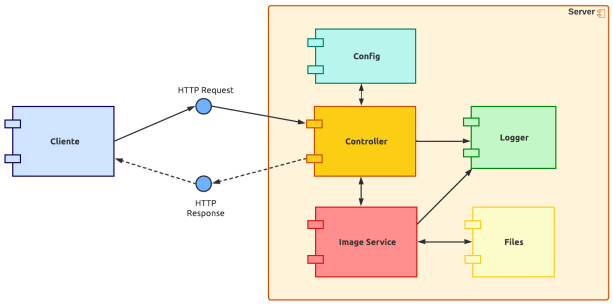

Fig. 6: Diagrama de componentes

V. Uso del proyecto

Para correr el servicio de imágenes debe compilar y construir los dos componentes del sistema: cliente y servidor.

1) Clonar el repositorio:

```
$ git clone
    https://github.com/ce-box/
    CE4303-pixel-counter.git
```

2) Ingresar al directorio del proyecto:

```
$ cd CE4303-pixel-counter
```

3) *Para construir el servidor:*
   a) Ingresar al directorio del servidor:
   ```
   $ cd server
   ```
   b) Ejecutar el archivo de instalación:
   ```
   $ ./install.sh
   ```
   c) Para correr el programa:
   ```
   $ ./build/ImageServer
   ```
   d) Si va a ejecutar el programa como un servicio:
      i) Primero, se debe actualizar la ruta base del proyecto en el archivo de configuración `ImageServer.service`.
      ii) Copiar el archivo a la carpeta de systemd:
      ```
      $ sudo cp ImageServer.service
                    /lib/systemd/system
      ```
      iii) Ejecutar los siguientes comandos:
      ```
      $ systemctl daemon-reload
      $ systemctl enable ImageServer
      $ systemctl start ImageServer
      ```
      iv) Si se necesita ajustar alguna configuración, debe realizarse en el archivo `server.config`.
4) *Para correr el cliente*:
   a) Debe tener instalado **Docker** y **Docker-compose** en su máquina local.
   b) Ingresar al directorio del cliente:
   ```
   $ cd client
   ```
   c) Ejecutar el archivo:
   ```
   $ ./run.sh
   ```
   d) Se abrirá la interfaz de texto en la que deberá ingresar la URL del servidor, la ubicación de la imagen y el valor del píxel de referencia. La Figura 7 muestra este comportamiento.

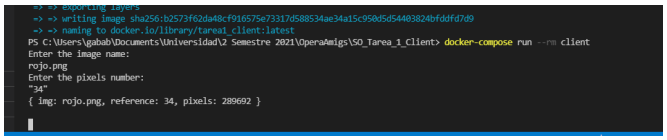

Fig. 7: Ejemplo de ejecución del programa cliente

VI. Tabla de actividades

A continuación se presentan las tablas de actividades según estudiante:

Cuadro I: Tareas asociadas a Martín Calderón

Tarea	Descripción	Hora
Solicitar datos al usuario	Programa para solicitar los datos requeridos al usuario	1h
Codificar imagen a base 64	Convertir el archivo de imagen en un archivo de texto codificado en base 64	1h
Crear formulario	Construir el contenido del mensaje que se envía al servidor	40m
Enviar request al servidor web	Realizar la llamada al servidor utilizando curl	50m
Mostrar el resultado	Presentar el resultado en la consola	15m
Despliegue en Docker	Configurar el Dockerfile para crear la imagen y correr el contenedor	1h 30m

Cuadro II: Tareas asociadas a Esteban Alvarado

Tarea	Descripción	Hora
Cargar imagen	Investigar y utilizar bibliotecas en C para cargar una imagen y acceder a sus píxeles	3h
Contar píxeles	Desarrollar un método que reciba el nombre del archivo imagen y un número, y cuente cuántos píxeles son mayores al valor dado.	1h
Servidor Web	Implementar el servidor web con el framework Ulfius	1h
Guardar archivos	Almacenar el contenido de un archivo de imagen en la memoria del servidor	30m
Comportamiento de demonio	Escribir el archivo de servicio y configurar el servidor como un demonio en la máquina virtual	20m

Cuadro III: Tareas asociadas a Olman Castro

Tarea	Descripción	Hora
Conversión base 64 a imagen	Implementar un método que decodifique la imagen recibida en base 64	1h
Configuración	Realizar la lectura del archivo de configuración y setear los valores en los módulos	2h
Sistema de logging	Implementar un sistema de logs que permita visualizar lo que sucede en el sistema	2h 30m
Despliegue en VM	Configurar el servidor para que corra en la VM de AWS	40m

VII. Conclusiones

Al lograr mantener el cliente y servidor en sus propios ambientes, ha sido posible evitar errores de dependencia e identificar de una mejor manera en que sector se ubican los errores; además, mediante la codificación y decodificación, es posible el envío de datos de en mayor tamaño, sin pérdida de información y a mayores velocidades. Finalmente, es posible cumplir con todos los objetivos del proyecto descritos a través del documento. El servidor, logra el comportamiento de un demonio proporcionando recursos al cliente y el cliente se encuentra en un ambiente virtualizado.

VIII. Sugerencias y recomendaciones

De la experiencia al desarrollar este proyecto recomendamos investigar adecuadamente los principios del procesamiento de imágenes, cómo están constituidos y la estructura de los distintos formatos, además de profundizar en el uso de otras bibliotecas en C para el manejo de archivos de imagen como *ImageMagick*, *PIL* y *OpenCV*.

También, recomendamos utilizar bibliotecas para el manejo del *logging* como `log.c` ya que brindan un conjunto de funcionalidades más robusto y completo que las implementadas en este sistema. Con la misma idea, sugerimos hacerlo también para el archivo de configuración, explorando bibliotecas como `libconfig` o `config file parsers` que expanden la cantidad de ajustes que se pueden tener.

References

[1] Y. Luo, "Virtualization and Cloud Computing", Berlin: Springer, Berlin, Heidelberg, 2012.[E-book]. Available: Springer.

[2] A. Tanenbaum, "Modern Operating Systems", India: Pearson Education, 2012. [E-book]. Available: Pearson education.

[3] Microsoft Azure, "¿Qué es un contenedor?", 2019. [Online]. Available on: https://azure.microsoft.com/es-es/overview/what-is-a-container/#overview [Accessed: Mar. 14, 2022].

[4] Medium (J. Torres), "Servicios y demonios en Linux", 2013. [Online]. Available on: https://medium.com/jmtorres/servicios-y-demonios-en-linux-a424366336ac .[Accessed: Mar. 14, 2022].