UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

Tema:

APLICACIÓN DEL SISTEMA DE CONTENEDORES DOCKER, COMO ALTERNATIVA A SISTEMAS DE VIRTUALIZACIÓN PARA MEJORAR EL TESTEO DE APLICACIONES EN UN ENTORNO DE DESARROLLO C#.

Trabajo de Integración Curricular, Modalidad: Proyecto de Investigación, presentado previo la obtención del título de Ingeniera en Tecnologías de la Información.

ÁREA: Hardware y redes

LÍNEA DE INVESTIGACIÓN: Sistemas administradores de recursos

Autor: Lizbeth Carolina Arboleda Bonilla

Tutor: Ing. David Omar Guevara Aulestia

Ambato - Ecuador

septiembre - 2022

APROBACIÓN DEL TUTOR

En calidad de tutor del Trabajo de Integración Curricular con el tema: APLICACIÓN DEL SISTEMA DE CONTENEDORES DOCKER, COMO ALTERNATIVA A SISTEMAS DE VIRTUALIZACIÓN PARA MEJORAR EL TESTEO DE APLICACIONES EN UN ENTORNO DE DESARROLLO C#, desarrollado bajo la modalidad Proyecto de Investigación realizado por la señorita Lizbeth Carolina Arboleda Bonilla, estudiante de la Carrera de Tecnologías de la Información, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, me permito indicar que la estudiante ha sido tutorada durante todo el desarrollo del trabajo hasta su conclusión, de acuerdo a lo dispuesto en el Artículo 17 del Reglamento para la ejecución de la Unidad de Integración Curricular y la obtención del título de tercer nivel, de grado en la Universidad Técnica de Ambato y sus reformas y el numeral 7.4 del respectivo instructivo.

Ambato, septiembre 2022

Firmado electrónicamente por:
DAVID OMAR GUEVARA AULESTIA

.....

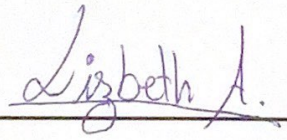
Ing. David Omar Guevara Aulestia

TUTOR

AUTORÍA

El presente trabajo de Integración Curricular titulado: APLICACIÓN DEL SISTEMA DE CONTENEDORES DOCKER, COMO ALTERNATIVA A SISTEMAS DE VIRTUALIZACIÓN PARA MEJORAR EL TESTEO DE APLICACIONES EN UN ENTORNO DE DESARROLLO C# es absolutamente original, auténtico y personal. En virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato, septiembre 2022



Lizbeth Carolina Arboleda Bonilla

C.C 1803651452

AUTOR

APROBACIÓN DEL TRIBUNAL DE GRADO

En calidad de par calificador del Informe Final del Trabajo de Integración Curricular presentado por la señorita Lizbeth Carolina Arboleda Bonilla, estudiante de la Carrera de Tecnologías de la Información, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, bajo la Modalidad Proyecto de Investigación, titulado APLICACIÓN DEL SISTEMA DE CONTENEDORES DOCKER, COMO ALTERNATIVA A SISTEMAS DE VIRTUALIZACIÓN PARA MEJORAR EL TESTEO DE APLICACIONES EN UN ENTORNO DE DESARROLLO C#, nos permitimos informar que el trabajo ha sido revisado y calificado de acuerdo al Artículo 19 del Reglamento para la ejecución de la Unidad de Integración Curricular y la obtención del título de tercer nivel, de grado en la Universidad Técnica de Ambato y sus reformas y al numeral 7.6 del respectivo instructivo. Para cuya constancia suscribimos, conjuntamente con la señora Presidente del Tribunal.

Ambato, septiembre 2022.

Firmado electrónicamente por:
ELSA PILAR URRUTIA

Ing. Pilar Urrutia, Mg.

PRESIDENTE DEL TRIBUNAL

Firmado electrónicamente por:
DENNIS VINICIO CHICAIZA CASTILLO

Ing. Dennis Vinicio Chicaiza Castillo

PROFESOR CALIFICADOR

Firmado electrónicamente por:
EDWIN HERNANDO BUENANO VALENCIA

Ing. Mg. Edwin Hernando Buenaño Valencia

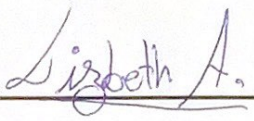
PROFESOR CALIFICADOR

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato, para que haga uso de este Trabajo de Integración Curricular como un documento disponible para la lectura, consulta y procesos de investigación.

Cedo los derechos de mi Trabajo de Integración Curricular en favor de la Universidad Técnica de Ambato, con fines de difusión pública. Además, autorizo su reproducción total o parcial dentro de las regulaciones de la institución.

Ambato, septiembre 2022.



Lizbeth Carolina Arboleda Bonilla

C.C 1803651452

AUTOR

DEDICATORIA

El presente proyecto es dedicado a mi madre, Anita, ella siempre ha estado para mí y me ha apoyado en todo momento, es un pilar fundamental en mi vida y con su cariño me ha ayudado a salir adelante ante todo obstáculo que se me ha presentado, Gracias por enseñarme a ser fuerte y a siempre salir adelante.

En memoria de mi tío y abuela, Luis y Olga, por demostrarme que puedo ser capaz de todo lo que me propongo y a luchar por lo que de verdad vale la pena y me gusta, siempre están presentes en mi memoria y corazón y jamás los olvidaré.

A familiares y amigos, quienes formaron parte de mi vida, y que de alguna forma aportaron para llegar a este momento. Muchas gracias.

Lizbeth Carolina Arboleda Bonilla

AGRADECIMIENTO

A Dios porque él siempre ha estado a mi lado ayudándome a no rendirme.

A mi madre, Anita, quien siempre ha apoyado mis proyectos y me ha impulsado a seguir adelante, nada de lo que soy hoy en día lo hubiera logrado si ella no estaría conmigo, es mi motor y siempre voy a estar agradecida por nunca dejarme sola y por siempre confiar en mí.

Un agradecimiento especial para mi tutor, Ing. David Guevara, por ser un excelente tutor y brindarme sus conocimientos para que este proyecto sea posible, por su preocupación en el tema y por estar dispuesto a aclarar mis dudas y solventarlas de la mejor manera. Por su calidad de persona, gracias por todo.

ÍNDICE

APROBACIÓN DEL TUTOR ..... **ii**

AUTORÍA ..... **iii**

APROBACIÓN DEL TRABAJO FINAL DE GRADO ..... **iv**

ÍNDICE ..... **viii**

ÍNDICE DE FIGURAS ..... **xi**

ÍNDICE DE TABLAS ..... **xiv**

ÍNDICE DE ANEXOS ..... **xv**

RESUMEN EJECUTIVO ..... **xvii**

ABSTRACT ..... **xviii**

BIBLIOGRAFÍA **80**

ANEXOS **84**

CAPÍTULO I.- MARCO TEÓRICO **1**

1.1 Tema de investigación **1**

1.1.1 Planteamiento del problema **1**

1.2 Antecedentes investigativos **2**

1.3 Fundamentación teórica **4**

1.4 Objetivos **12**

1.4.1 Objetivo general **12**

1.4.2 Objetivos específicos 12

CAPÍTULO II.- METODOLOGÍA 13

2.1 Materiales 13

2.2 Métodos 13

2.2.1 Modalidad de la investigación 13

2.2.2 Población y muestra 13

2.2.3 Recolección de información 13

2.2.4 Procesamiento y análisis de datos 14

CAPÍTULO III.- RESULTADOS Y DISCUSIÓN 15

3.1 Análisis y discusión 15

3.1.1 Determinación del método correcto de Benchmarking 15

3.1.2 Determinación del método de virtualización 15

3.1.3 Determinación del software de virtualización 15

3.1.4 Determinación de GNU/Linux 17

3.1.5 Determinación de software de medición de rendimiento Linux 18

3.1.6 Instalación de Docker en Linux 19

3.1.7 Uso y gestión de Docker 20

3.1.8 Uso y gestión KVM 22

3.1.9 Despliegue de entorno de desarrollo Docker y KVM 23

3.2 Desarrollo de la propuesta 23

3.2.1 Metodología Kanban 23

3.2.2 Desarrollo de las tareas 26

3.2.3 Instalación máquina virtual por KVM 29

3.2.4 Ejecución de la aplicación en Docker y máquina virtual 30

3.2.5 Toma de datos de rendimiento, velocidad, tiempo de arranque 45

3.2.6 Benchmarking 65

CONCLUSIONES Y RECOMENDACIONES 79

4.1 Conclusiones 79

4.2 Recomendaciones 79

ÍNDICE DE FIGURAS

Figura 3.1 Tablero Kanban 24

Figura 3.2 Tablero Kanban Actualizado por prioridades 25

Figura 3.3 Flujo de Trabajo 25

Figura 3.4 Imagen en Docker Hub 27

Figura 3.5 Docker iniciado y conectado. 28

Figura 3.6 Tiempo de instalación del contenedor Docker. 28

Figura 3.7 Máquina Virtual Iniciada y conectada. 29

Figura 3.8 Tiempo de instalación de la máquina virtual AlmaLinux. 30

Figura 3.9 Tiempo de inicio de la aplicación en Docker. 31

Figura 3.10 Inicio del programa en Docker. 31

Figura 3.11 Tiempo en agregar un libro en el programa en Docker. 32

Figura 3.12 Agregar Libro en programa en Docker. 33

Figura 3.13 Libro Agregado en programa en Docker. 33

Figura 3.14 Tiempo en actualizar un libro en el programa en Docker. 34

Figura 3.15 Actualizar un libro en el programa en Docker. 34

Figura 3.16 Libro Actualizado en el programa en Docker. 35

Figura 3.17 Tiempo en borrar un libro en el programa en Docker. 35

Figura 3.18 Borrar un libro en el programa en Docker. 35

Figura 3.19 Tiempo de inicio de programa en AlmaLinux. 38

Figura 3.20 Inicio de Programa en AlmaLinux. 38

Figura 3.21 Tiempo en agregar un libro en el programa en AlmaLinux. 39

Figura 3.22 Agregar Libro en el programa en AlmaLinux. 40

Figura 3.23 Libro Agregado en el programa en Almalinux. 40

Figura 3.24 Tiempo en actualizar un libro en el programa en AlmaLinux. 41

Figura 3.25 Actualizar Libro en el programa en AlmaLinux. 41

Figura 3.26 Libro Actualizado en el programa en AlmaLinux. 42

Figura 3.27 Tiempo en borrar un libro en el programa en AlmLinux. 42

Figura 3.28 Borrar Libro en el programa en AlmaLinux. 42

Figura 3.29 Glances en Docker. 46

Figura 3.30 Glances en AlmaLinux. 48

Figura 3.31 Htop en Docker. 51

Figura 3.32 Htop en AlmaLinux. 54

Figura 3.33 Stacer principal en Docker. 57

Figura 3.34 Stacer procesos en Docker. 58

Figura 3.35 Stacer principal en AlmaLinux. 61

Figura 3.36 Stacer procesos en AlmaLinux. 62

Figura 3.37 Código Dockerfile. 67

Figura 3.38 Código docker-compose.yml. 67

Figura 3.39 Tiempo inicio programa en nuevo contenedor. 68

Figura 3.40 Inicio programa en nuevo contenedor. 69

Figura 3.41 Tiempo agregar libro en el programa en el nuevo contenedor..... 69

Figura 3.42 Agregar libro en el programa en el nuevo contenedor. 70

Figura 3.43 Libro Agregado en el programa en el nuevo contenedor. 70

Figura 3.44 Tiempo en actualizar un libro en el programa en el nuevo contenedor. 71

Figura 3.45 Actualizar libro en el programa en el nuevo contenedor. 71

Figura 3.46 Libro Actualizado en el programa en el nuevo contenedor. 72

Figura 3.47 Tiempo en borrar un libro en el programa en el nuevo contenedor. 72

Figura 3.48 Libro borrado en el programa en el nuevo contenedor. 72

Figura 3.49 Htop en el nuevo contenedor..... 73

Figura 3.50 Glances en el nuevo contenedor. 74

Figura 3.51 Stacer principal en el nuevo contenedor. 75

Figura 3.52 Stacer procesos en el nuevo contenedor. 76

Figura 3.53 Imagen publicada en Docker Hub. 78

ÍNDICE DE TABLAS

Tabla 3.1 Comparación de Software de Virtualización[34] 15

Tabla 3.2 Comparación Distros Linux[35]..... 17

Tabla 3.3 Comparación software de medición Linux[36]..... 18

Tabla 3.4 Comandos Docker[37]..... 20

Tabla 3.5 Comandos KVM[38] 22

Tabla 3.6 Comparación ejecución CRUD en Docker y máquina virtual. 45

Tabla 3.7 Comparación Glances Docker y Máquina Virtual..... 51

Tabla 3.8 Comparación Htop Docker y máquina virtual. 56

Tabla 3.9 Comparación Stacer Docker y máquina virtual. 64

Tabla 3.10 Datos Benchmarking rendimiento Docker y máquina virtual 65

Tabla 3.11 Datos finales nuevo contenedor. 76

Tabla 3.12 Cuadro Resumen de Docker, máquina virtual y nuevo contenedor.. 77

ÍNDICE DE ANEXOS

Anexo 3.1 Gráfico de las tomas hechas de agregar libro en contenedor Docker.36

Anexo 3.2 Gráfico de las tomas hechas de actualizar libro en contenedor Docker. 37

Anexo 3.3 Gráfico de las tomas hechas de borrar libro en contenedor Docker. 37

Anexo 3.4 Gráfico de las tomas hechas de agregar libro en máquina virtual AlmaLinux. 43

Anexo 3.5 Gráfico de las tomas hechas de actualizar libro en máquina virtual AlmaLinux. 44

Anexo 3.6 Gráfico de las tomas hechas de borrar libro en máquina virtual AlmaLinux. 45

Anexo 3.7 Gráfico porcentaje CPU de Glances en Docker..... 47

Anexo 3.8 Gráfico porcentaje Memoria de Glances en Docker. 47

Anexo 3.9 Gráfico porcentaje SWAP de Glances en Docker. 48

Anexo 3.10 Gráfico porcentaje CPU de Glances en AlmaLinux.. 49

Anexo 3.11 Gráfico porcentaje Memoria de Glances en AlmaLinux. 50

Anexo 3.12 Gráfico porcentaje SWAP de Glances en AlmaLinux. 50

Anexo 3.13 Gráfico porcentaje CPU de htop en Docker.. 52

Anexo 3.14 Gráfico porcentaje Memoria de htop en Docker. 53

Anexo 3.15 Gráfico porcentaje SWAP de htop en Docker. 53

Anexo 3.16 Gráfico porcentaje CPU de htop en AlmaLinux..... 55

Anexo 3.17 Gráfico porcentaje Memoria de htop en AlmaLinux. 55

Anexo 3.18 Gráfico porcentaje SWAP de htop en AlmaLinux. 56

Anexo 3.19 Gráfico porcentaje CPU de stacer en Docker. 59

Anexo 3.20 Gráfico porcentaje Memoria de stacer en Docker. 59

Anexo 3.21 Gráfico porcentaje SWAP de stacer en Docker. 60

Anexo 3.22 Gráfico porcentaje CPU de stacer en AlmaLinux. 63

Anexo 3.23 Gráfico porcentaje Memoria de stacer en AlmaLinux..... 63

Anexo 3.24 Gráfico porcentaje SWAP de stacer en AlmaLinux. 64

Anexo 4.1 Ficha Bibliográfica 1 86

Anexo 4.2 Ficha Bibliográfica 2 87

Anexo 4.3 Ficha Bibliográfica 3 89

Anexo 4.4 Ficha Bibliográfica 4 91

Anexo 4.5 Clase libro CRUD 91

Anexo 4.6 Controlador CRUD 92

Anexo 4.7 Pantalla Agregar Libro 93

Anexo 4.8 Pantalla Principal 94

Anexo 4.9 Pantalla actualizar libro 95

Anexo 4.10 Programa c# 96

RESUMEN EJECUTIVO

La tecnología con el pasar de los días ha ido avanzando de una manera incalculable hasta el punto de que el mundo está rodeado de herramientas tecnológicas que facilitan el diario vivir de las personas. Las tecnologías de la información se van actualizando y es necesario capacitarse constantemente para siempre estar a la vanguardia de cada nueva implementación que exista. En el entorno informático es necesario saber cómo funciona hardware y software para poder brindar el mejor servicio a quien lo necesite; debido a la demanda que las tecnologías han tenido en estos últimos años las empresas han optado por mejorar sus recursos informáticos entrando al mundo de la reducción de recursos.

Un método innovador para limitar el uso de recursos, disminuir costos y mejorar el rendimiento de las empresas es la virtualización, ya que una empresa lo que desea es crecer en todos los campos incluidos en la parte tecnológica. Los desarrolladores de las empresas han optado en reducir el uso de recursos informáticos por medio de máquinas virtuales o contenedores para el uso de múltiples sistemas operativos en un solo servidor logrando así la ejecución de aplicaciones desarrolladas en las mismas sin consumir demasiados recursos físicos y virtuales.

El presente proyecto pretende demostrar el uso de máquinas virtuales y contenedores en el testeo de aplicaciones específicamente desarrolladas en C#; realiza un Benchmarking comparativo entre los dos métodos de disminución de recursos demostrando cuál es el ideal en cuanto a tiempo de arranque, velocidad, rendimiento y por supuesto uso de recursos. Genera 2 ambientes: El primero un contenedor Docker Linux con las dependencias necesarias para aplicaciones desarrolladas en C# y el segundo una máquina virtual de AlmaLinux 8.6 instalado por medio de KVM, en los dos se testeo la misma aplicación y se capturaron datos determinando cuál es el mejor para testeo de aplicaciones generando conclusiones para que futuras empresas lo empiecen a aplicar.

Palabras clave: Docker, KVM, máquina virtual, contenedores, c#, testeo aplicaciones, benchmarking.

ABSTRACT

Technology over the days has been advancing in an incalculable way to the point that the world is surrounded by technological tools that facilitate people's daily lives. Information technologies are being updated and it is necessary to be constantly trained to always be at the forefront of each new implementation that exists. In the computer environment it is necessary to know how hardware and software work in order to provide the best service to those who need it; due to the demand that technologies have had in recent years, companies have chosen to improve their computer resources by entering the world of resource reduction.

Within virtualization, we want to limit the use of resources to reduce costs and improve the performance of companies, since what a company wants is for its computer environment to be firm and sustainable to get ahead. The developers of the companies have chosen to use virtualization through virtual machines or containers for the use of multiple operating systems on a single server for the execution of applications developed on them without consuming too many resources.

This project aims to demonstrate the use of virtual machines and containers in the testing of applications specifically developed in C#, it carries out a comparative Benchmarking between the two virtualization methods demonstrating which is optimal in terms of boot time, speed, performance, and use of resources. It generates 2 environments: The first is a Docker AlmaLinux 8.6 container and the second a AlmaLinux 8.6 virtual machine installed by means of KVM, in both the same application was tested and data was captured determining which one is best for testing applications generating firm conclusions for future companies to start applying it.

Key Words: Docker, KVM, virtual machine, containers, c#, testing of applications, benchmarking.

CAPÍTULO I.- MARCO TEÓRICO

1.1 Tema de investigación

APLICACIÓN DEL SISTEMA DE CONTENEDORES DOCKER, COMO ALTERNATIVA A SISTEMAS DE VIRTUALIZACIÓN PARA MEJORAR EL TESTEO DE APLICACIONES EN UN ENTORNO DE DESARROLLO C#.

1.1.1 Planteamiento del problema

La informática se ha convertido en uno de los activos principales de las organizaciones y empresas dado que en la actualidad todo gira en base al manejo de información en Internet, esto conlleva estar propensos a diferentes tipos de amenazas como: virus, malware, ciberataques, entre otros[1]. Estas amenazas atentan la seguridad y fiabilidad de las empresas, ante esto se debe crear un plan de contingencia y se debe reforzar la seguridad en línea. Entre las formas óptimas de controlar los datos que se manejan de manera virtual se encuentran los Docker.

En todo el mundo, a partir de la pandemia mundial el Internet ha sido parte fundamental para poder salir adelante, sin embargo Ecuador no se encuentra al margen de los avances tecnológicos como otros países, se puede decir que la mayoría de empresas no están familiarizadas con los nuevos avances tecnológicos de gran importancia que ayudarían a optimizar de manera adecuada su funcionamiento de software[2], por ello, los nuevos ingenieros en Tecnologías de la información y comunicación deberán adentrarse más y explorar cómo ha ido avanzando la tecnología en otros países para mejorar los recursos de las empresas en las que futuramente trabajarán.

Los entornos de virtualización han ayudado a millones de empresas a reducir sus costos en hardware y facilitar el uso de múltiples sistemas operativos en un solo servidor[3] sin embargo con el avance tecnológico se ha determinado que existen nuevas aplicaciones que ayudarían de una manera más avanzada a las organizaciones y una de las herramientas que más ha ayudado es la implementación de Dockers, facilita la movilidad, escalabilidad, optimización y seguridad de los sistemas en las

empresas[4] y es por eso que la mayoría de empresas extranjeras de gran tamaño están optando por migrar su sistema a contenedores lo cual con el tiempo se está convirtiendo en el futuro del almacenamiento y flexibilidad en la nube.

No es ninguna novedad que muchas empresas han decidido cambiarse de Windows a Linux debido a las grandes características que esté brinda de acuerdo con el rendimiento de Sistema Operativo (S.O.) y velocidad de flujo de datos[5], sin embargo, por la falta de conocimiento o miedo a experimentar algo nuevo algunas empresas ecuatorianas se privan de mejorar sus sistemas y avanzar hacia el futuro de la tecnología. Las empresas deberían ver con la misma importancia que ven sus finanzas a la arquitectura informática, ya que si el sistema informático se cae podría generar grandes pérdidas a la misma volviéndola vulnerable, por eso día a día los ingenieros deben tener una capacitación continua y buscar nuevas medidas para que los sistemas sean seguros y fiables para la organización donde los implementen.

1.2 Antecedentes investigativos

Como antecedente investigativo se puede mencionar:

Según Marco Castillo en su trabajo de titulación “Seguridad informática, evaluación, comparación e implementación de equipos de seguridad”[6], trabajo realizado en la Universidad de las Américas, asegura que:

- Los ataques de seguridad a diario sobrepasan las estimaciones; sin embargo, la seguridad informática es un proceso continuo, que requiere anticipar hasta lo imprevisible, tener unos buenos hábitos y tomar buenas precauciones ayudarán de manera efectiva a que la empresa no sufra ataques a la información.

De igual forma menciona que la seguridad informática está compuesta por dos aspectos importantes los cuales son la seguridad computacional y la seguridad de redes [6] las cuales protegen la información de cualquier tipo de amenaza que se pueda presentar ya sea externa o interna.

Según José Luís Pacheco en su proyecto de titulación “Estudio comparativo entre una arquitectura con microservicios y contenedores Dockers y una arquitectura tradicional (monolítica) con comprobación aplicativa” [7], realizado en la Universidad de Guayaquil en el cual explica la importancia de usar correctamente los microservicios en base a contenedores para mejorar la arquitectura tradicional, y brinda algunos conceptos claves para mejorar el control de las empresas como lo es con la virtualización, da a conocer que:

- Una mala elección de arquitectura a la hora de la entrega de un sistema puede llegar a representar insatisfacción para el consumidor final y grandes pérdidas para la empresa.
- La virtualización de servidores ahorra espacio empresarial y electricidad, lo que a su vez reduce el gasto de capital empresarial.

Según Pol Ponsico Martín en su tesis de grado” Tecnología de Contenedores Docker” [8] presentada en la Universidad Politécnica de Cataluña en octubre de 2017, concluye que:

- Los contenedores son mucho más livianos que las máquinas virtuales debido a que las máquinas virtuales necesitan obligatoriamente la instalación del sistema operativo, asignación de disco, unidad central de procesamiento (CPU) y memoria de acceso aleatorio (RAM) para funcionar, mientras que un contenedor de Docker sólo necesita el sistema operativo que está en la máquina en la que se ejecuta el contenedor.

Según Rubén Cayetano en su tesis de grado” Software Containerization with Docker” [9] realizada en la Universidad de Ciencias Aplicadas de Turku, menciona el uso eficiente de Dockers para realizar contenerización en cualquier tipo de empresa, también realiza una comparación entre máquinas virtuales y contenedores en la cual concluye que:

- La ventaja de los contenedores es que proporcionan un nivel de aislamiento reducido en comparación con las máquinas virtuales y a un costo asequible, debido a que utilizan el Kernel del sistema operativo que ya se está ejecutando.
- Los contenedores no reemplazan la virtualización tradicional, pero son una alternativa viable cuando no se requiere de un aislamiento completo o cuando se necesita un sistema operativo diferente al sistema operativo principal para virtualizarlo.
- Docker ha evolucionado significativamente en los últimos años. La base de usuarios sigue creciendo y muchas grandes empresas han anunciado que están utilizando activamente Docker como parte de sus propios sistemas.

Según Jon-Anders Kabbe en su investigación" Security analysis of Docker containers in a production environment" [10] para su masterado en Tecnología en Ciencias de la Computación realizado en la Universidad Noruega de Ciencias y Tecnología en junio de 2017, analiza el uso de Dockers en entornos productivos para verificar su eficacia y adaptabilidad, de igual forma brinda conceptos básicos de cómo trabajar con Dockers y cuál es la correcta forma de implementarlos, compara la seguridad entre máquinas virtuales y Dockers, entre los conceptos más importantes menciona que:

- Los principales actores de la industria de Tecnologías de las Información (TI) están transformando su infraestructura en servicios más pequeños y están utilizando contenedores como base.
- Los Docker reducen las superficies de ataques debido a que las características de seguridad estándar del Kernel de Linux se han implementado en estos contenedores. Trabaja bajo el principio de seguridad por defecto la cual solo permite que las aplicaciones o sistemas tengan acceso específicamente a los recursos que se le han sido asignados.

1.3 Fundamentación teórica

Tecnologías de la información

La Tecnología de la Información (TI) se desarrolló a partir de avances científicos producidos en el campo de la informática. Es un conjunto de tecnologías que posibilitan el acceso, la producción, el procesamiento y la comunicación de información presentada bajo diferentes códigos los cuales pueden ser: texto, imágenes, audio, video[11]. Se puede destacar a Internet como el medio más optimo al hablar de TI debido a su alto impacto en avances tecnológicos, es el medio más factible hoy en día que usa el hombre para comunicarse. Las TI hoy en día están más adentradas en la sociedad debido a que la mayoría de las personas poseen una computadora o teléfono y se han convertido en elementos indispensables para un diario vivir.

Se define a la tecnología de la información como las nuevas tecnologías usadas para recuperar, procesar, almacenar y comunicar cualquier tipo de información logrando así mejorar los procesos operativos de las empresas y mejorando su productividad[12]. Las TI se fundamentan en 3 medios básicos[11]:

- Microelectrónica: Unión entre electricidad y electrónica.
- Informática: Se encarga de la manipulación de la información.
- Telecomunicaciones: Sistemas tecnológicos por donde se transmite la información.

Virtualización

El término virtualización fue creado por IBM en la década de los 60, con el principal objetivo de aumentar los niveles de compartición y reducción de costos de recursos informáticos[13], por lo que con el pasar de los años ha ido evolucionando para mejorar y lograr su objetivo principal tanto que hoy en día es uno de los métodos más efectivos para disminuir costos.

La virtualización es una técnica que permite que una o dos máquinas se ejecuten en una sola máquina física. Cada máquina que se ejecuta se denomina máquina virtual y estas poseen de forma independiente un grupo de recursos hardware y una copia de su propio sistema operativo[14]. Es una estrategia o metodología para dividir los recursos de TI en múltiples entornos de ejecución mediante la aplicación de uno o más de los

siguientes conceptos o tecnologías: particionamiento de software y hardware, tiempo compartido, simulación parcial o completa de ordenadores, emulación, calidad de servicio, entre otras[13]. Lo que básicamente desea hacer la virtualización es integrar y compartir recursos entre una máquina virtual y una real.

Existen diferentes tipos de virtualización que se acoplan a las necesidades de cada usuario[13]:

- Virtualización de servidores: Simulación de un entorno computacional.
- Virtualización de dispositivos de I/O: Control de enrutamiento de llamadas de I/O.
- Virtualización de memoria: Conjunto de memorias presentadas como pool generando memoria compartida.
- Virtualización de almacenamiento: Abstracción de almacenamiento físico en lógico.
- Virtualización de aplicación: Encapsulación de las aplicaciones en el sistema operativo donde se ejecutan para gestión, portabilidad y compatibilidad.
- Virtualización de escritorio: Aislar ambiente de escritorio basándose en el modelo cliente-servidor.
- Virtualización de redes: Combinar recursos de hardware, software y funcionalidades de red en una sola red virtual.

Arquitectura Informática

Según la Real Academia Española (RAE) Informática es el "conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático de la información por medio de computadoras"[15]. Por lo que se puede decir que la informática es la ciencia que estudia el procesamiento de la información. La informática va ligada de la mano con la tecnología por lo que gracias a la informática existe la comunicación directa entre tecnología y ser humano.

La arquitectura de computadoras o arquitectura informática abarca la especificación del repertorio de instrucciones y las unidades hardware (medios de almacenamiento, dispositivos de comunicación, equipos) que implementan dichas instrucciones[16].

Contenedores

Un contenedor es una unidad estándar de software que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de forma rápida y fiable de un entorno informático a otro. El contenedor más conocido es Docker, es un paquete de software independiente, ejecutable y ligero que contienen todo lo necesario para ejecutar una aplicación o máquina[17]. Docker es una tecnología de código abierto lanzada en el 2013. Es un proyecto automatizado que implementa aplicaciones en contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos (Linux y últimamente Windows)[18]. Hoy en día las empresas están optando por cambiar su arquitectura informática por una más ligera y ahí es donde entran Dockers ya que ayuda a generar nuevos entornos informáticos de forma segura y rápida.

Docker Hub

Docker Hub brinda un registro público para que los usuarios almacenen y compartan aplicaciones en contenedores. Es el repositorio más grande y oficial de imágenes para Docker publicadas por desarrolladores. Al año se descargan miles de millones de imágenes por usuarios para la gestión de datos, implementación de sitios web, gestión de datos y tareas comerciales o personales[19]. Es recomendable al momento de realizar un contenedor crear la imagen desde el repositorio oficial Docker Hub el cual por años ha ido almacenando imágenes seguras con diferentes características que ayudan a los usuarios a realizar tareas específicas en sus contenedores.

C#

Fue creado por Microsoft y es un lenguaje de programación moderno basa principalmente en la orientación a objetos y componentes, tiene como base a C, C++, Visual Basic y Java[20]. Su base fundamental radica en brindar a los desarrolladores

y programadores un lenguaje que permita crear aplicaciones sólidas, duraderas y seguras que se puedan ejecutar en .NET[21]. C# ha adaptado y mejorado facilidades de sus lenguajes base para convertirse en un lenguaje fácil de aprender orientado a objetos y añadiendo funcionalidades para que los códigos corran de manera más rápida de lo conocido. Ha evolucionado tanto que muchos desarrolladores lo usan para crear sistemas operativos, aplicaciones de escritorio y juegos.

Entre las varias características que tiene el lenguaje se puede resaltar las siguientes[22]:

- Sencillez: Elimina elementos innecesarios de .NET.
- Modernidad: Añade de forma automática elementos necesarios para mejorar la programación actual.
- Seguridad: Evita que se generen errores que a futuro serian difíciles de detectar.
- Compatible: Fácil de trabajar gracias a sus lenguajes base.
- Versionable: Tiene mejoras continuas de acuerdo con las necesidades que se vayan presentando con el pasar del tiempo.

Docker Destop

Es una aplicación que permite compartir y crear aplicaciones en contenedores para los entornos de Windows o Mac. Incluye todo lo necesario para que Docker funcione en el sistema operativo como: Docker CLI, Docker Engine, Docker Content Trust, Docker Compose, Kubernetes y Credential Helper[23]. Funciona para cualquier tipo de lenguaje de desarrollo que se elija ya que tiene acceso a todas las plantillas certificadas y a la biblioteca de imágenes de Docker Hub logrando así que todos los desarrolladores trabajen en una comunidad integral y repositorio seguro.

Entre las principales características se pueden resaltar[23]:

- Contiene cualquier tipo de aplicación multiplataforma.
- Fácil configuración e instalación de un entorno de desarrollo Docker,
- Capacidad para alternar entre desarrollos,
- Rendimiento confiable y rápido.

.NET

Fue creada por Microsoft como una plataforma de desarrollo de código abierto, en el cual se pueden construir y ejecutar diferentes tipos de aplicaciones y programas en diferentes lenguajes como lo son C#, F#, Visual C++ o Visual Basic[24], sin embargo, el lenguaje más común es C#.

.NET al momento de descargar contiene una variedad de programas que ayudan a[24]:

- El código C# se entienda en la computadora.
- Brinda herramientas que facilitan a que un software se ejecute en pantalla.
- Proporciona todo tipo de datos para que se los pueda definir y almacenar la información que se necesite.

ASP.NET

ASP.NET amplia .NET con bibliotecas y herramientas para crear específicamente programas para la web[25]. Permite crear varios tipos de aplicaciones web entre las comunes: painas web, microservicios, API REST y concentradores. Se puede decir que ASP.NET es un marco y .NET es una plataforma por lo que entre las herramientas y bibliotecas más básicas que ASP.NET integra a .NET están[25]:

- Framework base para analizar solicitudes web en C#.
- Plantillas de páginas web dinámicas con c#.
- Bibliotecas para patrones como Model View Controller (MVC).
- Sistema de autenticación.
- Extensiones de editor para funciones específicas para el desarrollo de páginas web.

Microsoft Visual Studio

Pueden Microsoft Visual Studio es un entorno de desarrollo integrado (IDE) creado por Microsoft que usa varios aspectos y tipos de desarrollo de software como aplicaciones móviles, web, de escritorio, servicios web entre otros. Facilita el proceso

de desarrollo de software en diferentes lenguajes de programación ya que es una plataforma de desarrollo ágil que se puede usar para depurar, compilar, editar y publicar aplicaciones[26]. Entre las funcionalidades que más resaltan y por las que es reconocido Microsoft Visual Studio es porque contiene copiladores, diseñadores gráficos, diseñador de esquemas de bases de datos herramientas de finalización de código, refactorización de código, generador de perfiles de código y muchas más funciones y extensiones que facilitan la creación y desarrollo de software.

La familia Microsoft Visual Studio posee 3 elementos principales los cuales son[27]:

- Visual Studio: IDE para desarrolladores .NET y C++ en Windows.
- Visual Studio para MAC: IDE para desarrolladores .NET en macOS.
- Visual Studio Code: Editor de código fuente compatible con Windows, macOS y Linux.

CRUD

Un modelo básico está compuesto de 4 funciones principales: Crear, leer, actualizar y eliminar (CRUD)[28] las cuales son necesarias para implementar una buena aplicación de almacenamiento de datos persistente el cual significa que los datos se conservaran siempre[29], aunque se perdía cualquier energía por lo que CRUD ayuda a que el modelo sea completo y utilizable en cualquier plataforma.

- **Crear:** Función que sirve para agregar un nuevo objeto.
- **Leer:** Función que sirve para ver un objeto sin poder editarlo es decir solo se lo recupera.
- **Actualizar:** Función que sirve para cambiar información de un objeto.
- **Eliminar:** Función que sirve para eliminar un objeto.

Benchmarking

El Benchmarking o evaluación comparativo es un proceso por el cual se mide el rendimiento de procesos de un software frente a otro considerado como el mejor de su clase. El objetivo principal es identificar los aspectos positivos de mejora de un

software para ajustarlos a otro y mejorarlo en base a uno ya mejorado[30], lo que propone es comparar rendimiento y alcances para superar el ya establecido como “el mejor” y si no logra superar pretende generar nuevas ideas para que crezca en el mercado y sea un competidor potencial.

Para que se realice un buen benchmarking se debe seguir los siguientes pasos[31]:

1. Plan: Definir un tema y estudiar el proceso.
2. Recoger: Recopilar información.
3. Analizar: Comparar datos recopilados.
4. Adaptar: Desarrollar planes de acción.

Kanban

Kanban es una de las principales implementaciones de metodología ágil para software. Fue creado en base al Sistema de Producción de Toyota y permite controlar que no existan cuellos de botella en los procesos realizados y en caso de que existan los soluciona de forma rentable y a una velocidad optima[32]. Su principal objetivo es gestionar el trabajo a medida que va avanzando el proceso visualizándolo de forma real como en flujos de trabajo.

Se basa en el principio básico de divide y vencerás por lo que se pueden identificar diferentes etapas de desarrollo. El requisito fundamental es dividir en tres grandes partes el proyecto: "Por hacer", "En progreso" y "Terminado[32]. Para eso se puede separar en las siguientes categorías.

1. Pila de Producto
2. Requisitos
3. Diseño
4. Desarrollo
5. Pruebas
6. Despliegue
7. Hecho

Aunque Kanban también posee 6 fases las cuales ayuda a controlar el proyecto de forma sencilla[33]:

1. Visualizar el flujo de trabajo.
2. Limitar el WIP (trabajo en curso).
3. Gestionar el flujo.
4. Hacer explícitas las políticas de los procesos.
5. Implementar bucles de retroalimentación.
6. Mejorar en colaboración, evolucionar experimentalmente.

1.4 Objetivos

1.4.1 Objetivo general

Aplicar el sistema de contenedores Docker como alternativa a sistemas de virtualización para mejorar el testeo de aplicaciones en un entorno de desarrollo C#.

1.4.2 Objetivos específicos

- Configurar un sistema de contenedores Docker y virtualización para un entorno de desarrollo C#.
- Emplear una metodología de Benchmarking para comparar el rendimiento entre el sistema de contenedores Docker y virtualización en un entorno de desarrollo C#.
- Implementar Docker para establecer las mejores prácticas de uso e implementación en un entorno de desarrollo C#.

CAPÍTULO II.- METODOLOGÍA

2.1 Materiales

Debido a la naturaleza del proyecto de investigación para la recolección de la información se utilizó fichas bibliográficas las cuales resumen tesis similares al problema y tema planteado, logrando así tener un enfoque más claro de lo que se realizó.

2.2 Métodos

2.2.1 Modalidad de la investigación

El proyecto de investigación es bibliográfico-documental porque se basa en investigaciones previas del tema planteado, en libros, tesis y artículos científicos que hacen referencia a las variables planteadas para poder construir y desarrollar la investigación.

2.2.2 Población y muestra

Por la naturaleza del problema no se utilizó población y muestra ya que la investigación es netamente bibliográfica- documental.

2.2.3 Recolección de información

Las fichas bibliográficas mostradas en los Anexos 4.1,4.2,4.3,4.4 realizan comparaciones y netamente buscan la diferencia entre usar máquinas virtuales y usar Docker para virtualización, exponen aspectos importantes sobre los mismos, pero al momento de realizar la comparación se centran en una comparación global y no especifica de las características óptimas para un testeo de aplicaciones. Muestran varias herramientas que pueden ayudar a dicho testeo sin embargo ninguna lo ha puesto a práctica y se basan solo en investigaciones ya realizadas y en las conclusiones ya establecidas. En este trabajo investigativo se demuestra que, para el testeo de aplicaciones, Docker puede reemplazar a las máquinas virtuales en cuanto ahorro de recursos, realizando un estudio en el cual se compara a profundidad el rendimiento,

rapidez, flexibilidad, entre otras características que ayudan a demostrar el objetivo planteado. Se basa en una comparación fundamentada guiada por la práctica realizada y documentación investigada. Las soluciones que brinda aportan a que nuevas empresas analicen mejor su arquitectura y empiecen a implementar Docker desde su creación.

2.2.4 Procesamiento y análisis de datos

De acuerdo con la información recolectada de temas similares al planteado se puede decir que:

- Se debe realizar una investigación sobre los beneficios que tiene Docker sobre virtualización para poder detallar sus ventajas frente a máquinas virtuales.
- Se debe tomar en cuenta que si se desea comparar contenedores Docker con máquinas virtuales se debe buscar un software especializado a realizar dicha comparación en cuanto a rendimiento, rapidez, fluidez, entre otros.
- Se debe tener en cuenta que para entender mejor la comparación que se está haciendo entre Docker y máquinas virtuales no deben faltar pruebas de testeo de aplicaciones en C# para entender de forma más visual la comparación.
- Al momento de realizar una comparación de tipo Benchmarking se debe brindar alternativas de cómo mejorar la mejor opción para establecer una mejora continua.
- Se debe realizar una implementación de la imagen final de Docker para corroborar que los objetivos se cumplieron.

CAPÍTULO III.- RESULTADOS Y DISCUSIÓN

3.1 Análisis y discusión

3.1.1 Determinación del método correcto de Benchmarking

Para el presente proyecto se decidió realizar un Benchmarking comparativo ya que es el ideal para realizar comparaciones entre dos objetos que poseen características similares y que brindan el mismo fin, por lo que ayudó a que se pueda realizar una comparación de las características más importante y que resaltan más entre máquinas virtuales y contenedores para así lograr determinar cuál es el mejor y como se lo podría mejorar para que continue sobresaliendo en cuanto a rendimiento, velocidad y uso de recursos. A partir de esta determinación en todos los procesos de comparación se realizó un Benchmarking comparativo.

3.1.2 Determinación del método de virtualización

Para el presente proyecto se eligió la virtualización de aplicaciones debido a que es la que más se acopla a las necesidades del proyecto ya que en esta virtualización se puede ejecutar las aplicaciones de cualquier tipo y medir su rendimiento, velocidad, recursos y tiempo de arranque. Ayuda a resolver problemas de incompatibilidad y permite que se interactúe netamente con la aplicación y se monitoree su funcionamiento y rendimiento dentro del contenedor logrando así que los usuarios tengan un control completo de la aplicación que están usando o ejecutando.

3.1.3 Determinación del software de virtualización

Para la determinación del software de virtualización se decidió comparar los 5 mejores softwares según el mercado y ver sus características en cuanto a tipo (1 o 2), virtualización (alojado, nativo o emulado), arquitectura, Sistema Operativo (S.O) Host y Guest. La tabla 3.1 muestra los resultados obtenidos.

Tabla 3.1 Comparación de Software de Virtualización[34]

	TIPO	VIRTUALIZACIÓN	ARQUITECTURA	S.O. HOST	S.O. GUEST
QEMU	2	Hipervisor alojado y emulado	x86-64	MacOS, Windows y Linux	Windows y Linux
KVM	1	Hipervisor nativo	x86-64	Linux	Windows y Linux
Vmware Workstation Pro	2	Hipervisor alojado	X86-64	Windows a partir de Windows Server 2012 y Linux	Windows a partir de Windows XP y Linux
Oracle VirtualBox	2	Hipervisor alojado	x86-64	Windows a partir de Windows Server 2012 y Linux	Windows a partir de Windows XP y Linux
Microsoft Hyper-V Manager	2	Hipervisor native	x86-64	Windows 10	Windows a partir de Windows Vista y Linux

Elaborado por: El investigador

De acuerdo al análisis comparativo realizado en la tabla 3.1 se decidió utilizar el software de virtualización KM ya que es un hipervisor de tipo 1 eso quiere decir que usa directamente el hardware del host logrando que la virtualización brinde un mejor rendimiento, también KVM es compatible con Windows y Linux lo cual facilitará la ejecución de varias máquinas virtuales para la posterior comparación, de igual forma KVM tiene un diseño fácil de usar y permite ejecutar las máquinas virtuales de forma

sencilla, clara y soporta todas las distros de GNU/Linux en el cual se puede ejecutar Visual Studio Code para poder ejecutar la aplicación a testear.

3.1.4 Determinación de GNU/Linux

Para la determinación de GNU/Linux se decidió comparar las distros más usadas en los ultimo 3 años en donde se determino que fueron 7. Se las compararon en base a las características más relevante que son distro base, arquitectura, gestor de paquetes, tipo y usos. La tabla 3.2 muestra los resultados obtenidos.

Tabla 3.2 Comparación Distros Linux[35]

	Distro Base	**Arquitectura**	**Gestor de Paquetes**	**Tipo**	**Usos**
Debian	No existe	X86-64, Power PC, MIPS	APT	Gratuito	Multiusos
Ubuntu	Debian	X86-64, Power PC, Sparc64	APT	Comercial Gratuito	Producción
Red Hat Linux	No existe	X86-64, Power PC	APT	Comercial	Producción
Almalinux	Red Hat Linux	X86-64, Power PC	DNF	Gratuito	Multiusos
Fedora	Red Hat Linux	X86-64, Power PC	DNF	Gratuito	Multiusos
CentOS	Red Hat Linux	X86-64	DNF	Gratuito	Multiusos

Elaborado por: El investigador

Con base al análisis realizado en conjunto con la tabla 3.2 se eligió que la distro que se va a usar va a ser Almalinux debido a que es una distro fácil de usar y que no se necesita tener previo conocimiento de Linux para usarlo, de igual forma como es

código abierto existen varios foros que pueden ayudar a saber cómo usarlo, la ventaja principal y por la que se eligió es que es multiusos lo que ayudara a poder testear de forma correcta cualquier aplicación que se corra en ella y es compatible con la mayoría de Linux que existen debido a que está basado en Red Hat Linux, otra ventaja importante por la que se eligió esta distro es porque es la más actualizada y posee documentación actualizada.

3.1.5 Determinación de software de medición de rendimiento Linux

Para la determinación de software de medición de rendimiento de Linux se decidió comparar los más usados en el último año y se determinó que son 7. En este caso según las investigaciones previas se demuestra que para obtener un resultado favorable al hablar de medición de rendimiento mínimo se debe usar de 2 a 3 software externos por lo que se decidió usar 3 software. La tabla 3.3 muestra la comparación en cuanto a categoría, medición de CPU, disco, memoria y compatibilidad.

Tabla 3.3 Comparación software de medición Linux[36]

	Categoría	**CPU**	**Disco**	**Memoria**	**Compatibilidad**
Stacer	GNU y línea de comandos	X	X	X	Debian, AlmaLinux y RPM, pero la AppImage está disponible para todas las distribuciones de Linux
Htop	Línea de comandos	X	X	X	Todas las distribuciones de Linux
Monitorix	Interfaz web	X		X	Todas las distribuciones de Linux

Xfce	GNU	X	X	X	Ubuntu o Linux basado en Debian
Conky	GNU	X	X	X	Ubuntu o Debian
Glances	Interfaz web y línea de comandos	X	X	X	Ubuntu, AlmaLinux o Debian y para la demás distro Linux por Github
Ctop	Línea de comandos	X		X	Cualquier contenedor Docker

Elaborado por: El investigador

Con base al análisis hecho en la tabla 3.3 se ha decidido trabajar con Glances, htop y Stacer para la toma de tiempo de rendimiento y uso tanto del contenedor como de la máquina virtual debido a que las aplicaciones seleccionadas son compatibles con las distribuciones Linux que se usan en el proyecto y miden CPU, memoria y disco.

3.1.6 Instalación de Docker en Linux

Existen varias distribuciones de Linux en las cuales se puede instalar Docker, la mayoría sigue los mismos pasos solo se cambia la url de repositorio. El Linux que se eligió fue Fedora por su flexibilidad al momento de realizar aplicaciones.

Primero se debe instalar el paquete dnf-plugins-core para administrar los repositorios y hacerlos estables, se lo instala con el siguiente comando:

sudo dnf -y install dnf-plugins-core

Una vez instalado el paquete agregar el repositorio de Docker:

sudo dnf config-manager --add-repo https://download.docker.com/linux/fedora/docker-ce.repo

Instalar la última versión de Docker con el siguiente comando:

sudo dnf install docker-ce docker-ce-cli containerd.io docker-compose-plugin

Se crea un grupo de Docker, pero no se asignará a ningún usuario por lo que se debe asignar manualmente pero antes se debe iniciar Docker.

sudo systemctl start docker

Agregar al usuario actual al grupo Docker:

sudo usermod -aG Docker [usuario]

Salir e iniciar sesión con el usuario Docker y ya se puede usar Docker con cualquier imagen del repositorio.

3.1.7 Uso y gestión de Docker

La tabla 3.4 explica los comandos más usados en Docker.

Tabla 3.4 Comandos Docker[37]

Comando	**Definición**
docker create	Crear un Docker (solo lo crea no lo inicia)
docker rename	Renombra el Docker
docker run	Crea e inicia un Docker en una operación de una imagen
docker rm	Elimina un Docker
docker update	Actualiza los recursos del Docker
docker start	Inicia un Docker
docker stop	Detiene un Docker

docker restart	Reinicia un Docker
docker pause	Congela un Docker en ejecución
docker unpause	Descongela un Docker en ejecución
docker kill	Mata a un Docker en ejecución
docker attach	Conecta a un Docker en ejecución
docker ps	Muestra los Docker que se están ejecutando
docker logs	Obtiene los logs de un Docker
docker inspect	Muestra toda la información de los objetos un Docker
docker events	Muestra los eventos de un Docker
docker port	Muestra el puerto púbico de un Docker
docker top	Muestra los procesos que están corriendo en un Docker
docker stats	Muestra las estadísticas de los recursos usados en un Docker
docker images	Muestra todas las imágenes
docker import	Crea una imagen de un tarball
docker build	Crea una imagen de un Dockerfile
docker commit	Crea una imagen de un contenedor y lo pausa temporalmente
docker rmi	Remueve la imagen
docker load	Carga una imagen de un archivo tar
docker save	Guarda una imagen a un archivo tar a STDOUT con todas las capas padre, tags y versiones.
docker history	Muestra el historial de una imagen
docker -version	Muestra la versión de Docker
docker –help	Muestra la ayuda de Docker
docker info	Muestra la información de un Docker
docker search	Buscar imágenes en Docker Hub

docker pull	Descargar imágenes desde Docker Hub

Elaborado por: El investigador

3.1.8 Uso y gestión KVM

KVM para controlar las máquinas virtuales utiliza virsh, en la tabla 3.5 se puede observar los distintos comandos más sus definiciones:

Tabla 3.5 ***Comandos KVM***[38]

Comando	**Definición**
virsh create	Crea una máquina virtual a partir de un archivo de configuración
virsh help	Muestra la lista de opciones de virsh
vish list	Muestra la lista de máquinas virtuales cargadas en el hipervisor
virsh start	Enciende una máquina virtual
virsh suspend	Suspende una máquina virtual
virsh resume	Reanuda una máquina virtual suspendida
virsh stop	Apaga una máquina virtual
virsh destroy	Apaga de manera inmediata una máquina virtual y la elimina
virsh reboot	Reinicia una máquina virtual
virt-top	Muestra la carga, estadísticas y rendimiento de una máquina virtual
virsh vol-resize	Redimensiona el volumen de una máquina virtual
virsh attach-disk	Asocia un volumen a una máquina virtual
virsh detach-disk	Desasocia un volumen a una máquina virtual
virsh edit	Edita una máquina virtual
virsh net-list –all	Listar todas las interfaces de red

virsh pool-list	Lista la información del pool de almacenamiento
virsh vol-create	Crea un nuevo volumen de almacenamiento
virsh net-create	Añade una nueva red virtual
virsh net-undefined	Borra una red virtual
virsh snapshot-create-as	Crea una instancia
virt-clone	Clona una máquina virtual
virsh migrate	Migra una máquina virtual a otra anfitriona

Elaborado por: El investigador

3.1.9 Despliegue de entorno de desarrollo Docker y KVM

Para el presente proyecto se decidió instalar un contenedor Docker Linux que soporte Visual Studio Code y un navegador web, de igual forma se instaló una máquina virtual Linux con las mismas características que el contenedor para así poder testear su rendimiento.

3.2 Desarrollo de la propuesta

3.2.1 Metodología Kanban

Para el presente proyecto se eligió la metodología Kanban la cuál ayuda a mejorar y visualizar el flujo continuo del proceso de desarrollo del proyecto con base a tableros visuales que monitorean las actividades que se realiza. Kanban posee 6 fases que ayudan a que el proyecto sea monitorizado y desarrollado de la mejor forma.

- **Visualizar el flujo de trabajo**

Para visualizar el flujo de trabajo se creó el siguiente tablero Kanban con la aplicación Trello en donde se especifican todas las actividades a desarrollar, las que se están desarrollando y las finalizadas del proyecto.

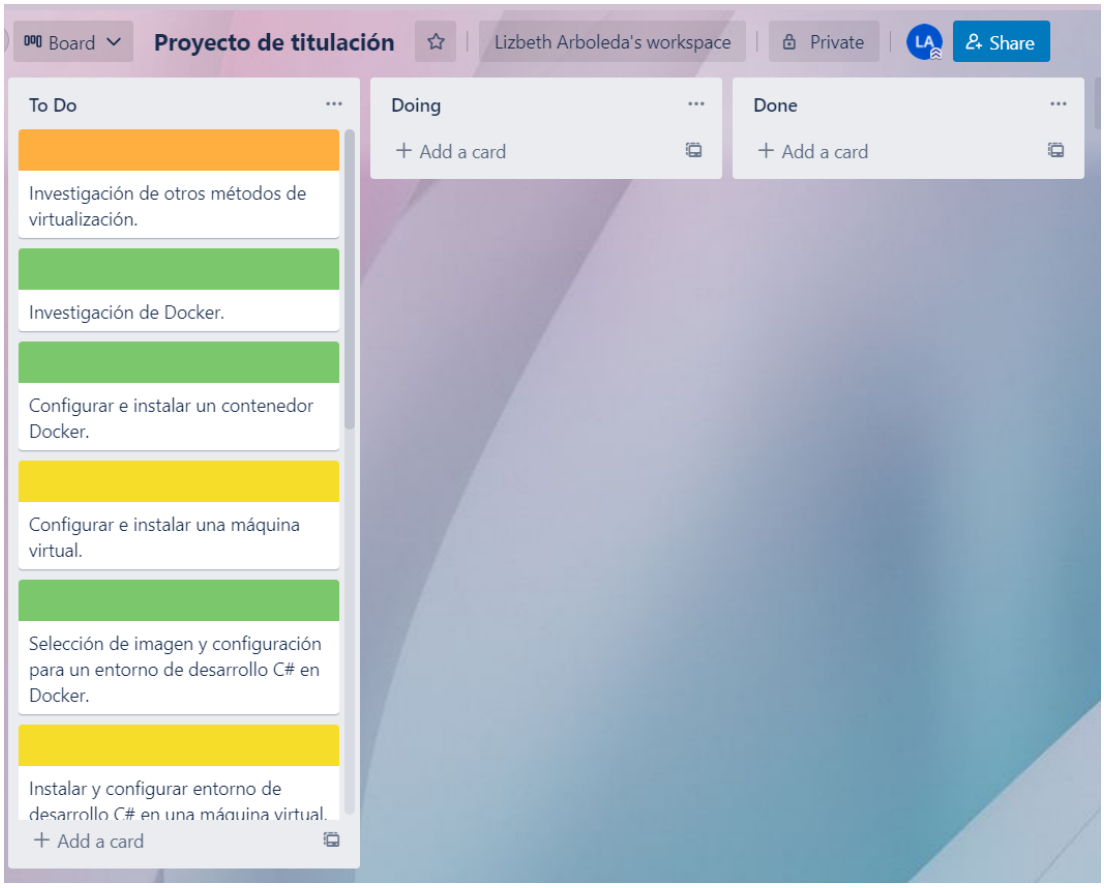


Figura 3.1 Tablero Kanban

La figura 3.1 muestra el inicio de la fase de planificación del proyecto en donde se puede observar todas las actividades que se realizaron en la ejecución del proyecto que conforme se avanzó se siguieron moviendo a las diferentes fases.

- **Limitar el WIP**

Los WIP que más tiempo tomaron y que se toman como prioridad son las de testeo de Docker y máquinas virtuales al igual que el testeo de la aplicación C#. Aquí se debe limitarlos para que no se hagan más tareas antes de que estas se realicen para que no existan tareas acumuladas, estas se pueden priorizar en Trello por colores.

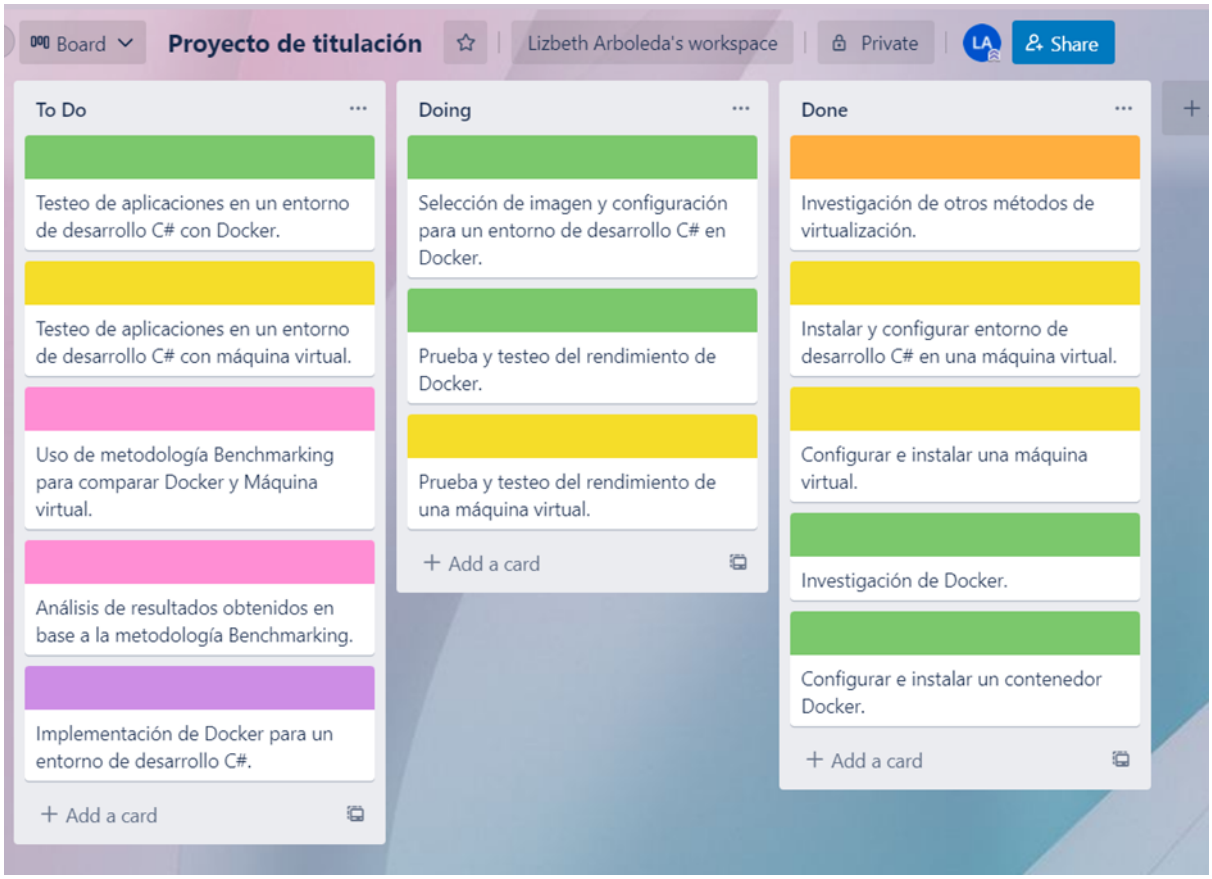


Figura 3.2 Tablero Kanban Actualizado por prioridades

Una vez realizadas las actividades principales el tablero cambio y se puede observar en la figura 3.2 que tiene actividades tanto en "En proceso" como en "Finalizadas" ya que el objetivo de la metodología es llevar un registro actualizado de las actividades conforme se van completando, este tablero ayudó a que el tiempo establecido se cumpla en base a la prioridad de tareas.

- **Gestionar el flujo**

Figura 3.3 Flujo de Trabajo

El flujo de trabajo varia conforme a la realización de actividades ya que el investigador puede aumentar tareas básicas en el tablero Kanban, la figura 3.3 muestra el flujo de la figura 3.2.

- **Hacer explícitas las políticas de los procesos**

Las actividades deben estar bien definidas para garantizar su cumplimiento en los horarios establecidos en el tablero Kanban.

- **Implementar bucles de retroalimentación**

Se visualizó diariamente el tablero Kanban para observar las actividades pendientes y estar atentos para cumplirlos en el tiempo adecuado. De igual forma se tomo apuntes de las actividades que requerían más tiempo y enfoque para darles prioridad.

3.2.2 Desarrollo de las tareas

La aplicación que se testeó fue un CRUD MVC creado en ASP.NET Core con ayuda de los paquetes de Entity Framework. Se puede observar su código en los Anexos 4.5, 4.6, 4.7, 4.8, 4.9, 4.10. Dicha aplicación funciona para el registro de libros en una librería virtual, en esta se puede visualizar todos los libros que existen, se puede agregar un nuevo libro con las características necesarias como lo son: nombre, número de páginas, editorial, autor y precio, de igual forma se puede actualizar cualquier atributo de los libros existentes y se pueden eliminar libros. Es una aplicación que posee su base de datos alojada en un host online gratuito y está montada en GitHub en el siguiente link: https://github.com/lizarboleda99/CRUD_LibrosVSC por lo que se la puede clonar y correr localmente para medir el rendimiento y uso de recursos de la misma.

3.2.2.1 Instalación Docker

Se busco una imagen Docker que cumpla con los requerimientos establecidos.

Figura 3.4 Imagen en Docker Hub

La figura 3.4 muestra la imagen Docker seleccionada para el testeo de la aplicación creada, la imagen contiene una instalación mínima de un Linux de escritorio con software para desarrollar y testear aplicaciones desarrolladas en C#.

Una vez escogida la imagen se procedió a hacer un pull para descargar la imagen Docker, ya descargada se creó un contenedor con dicha imagen para poder utilizarlo. El contenedor posee un software de acceso remoto VNC por lo que al momento de crearlo se asignó una contraseña y el puerto 5901 por el que se comunicará y se podrá conectar remotamente al contenedor, en la figura 3.5 se puede observar que el contenedor es un Fedora Linux con 1.5G de memoria, 1 procesador y 28G de disco. Posee Visual Studio Code con todas las dependencias necesarias para ejecutar aplicaciones desarrolladas en C#, Qterminal para comunicarse directamente con la terminal y Falkon como navegador, la conexión remota se la realiza por VNC.

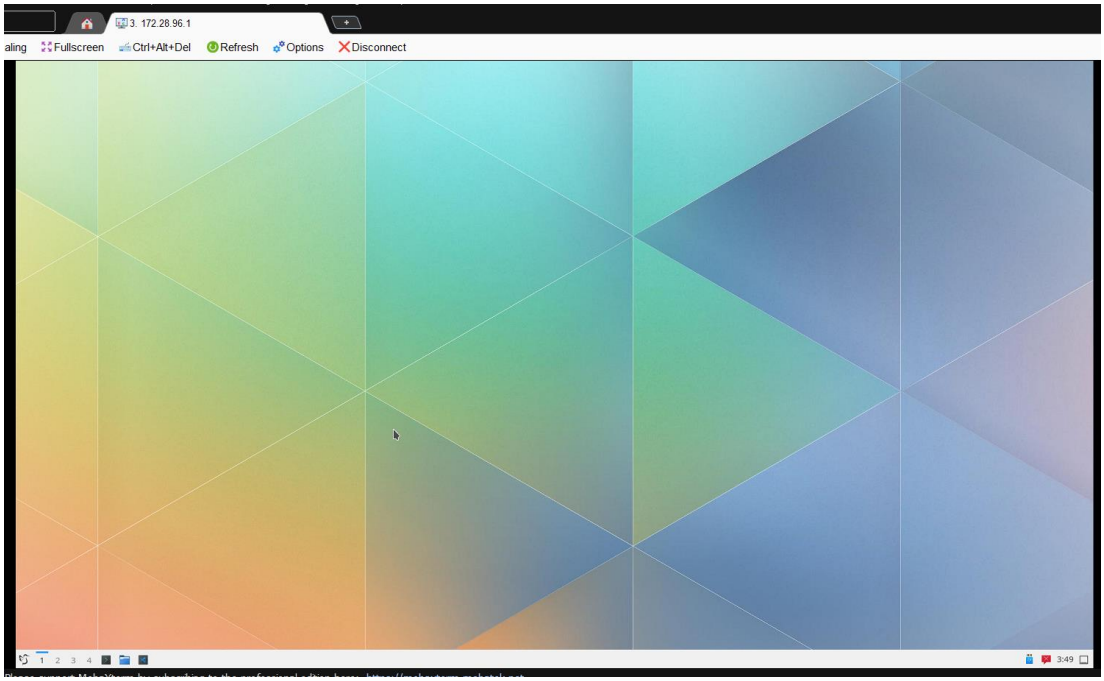


Figura 3.5 Docker iniciado y conectado.

El tiempo de instalación del contenedor fue de 5 minutos como se puede observar en la figura 3.6 la cual detalla en el Lap 1 el tiempo que se tardó en hacer el pull de la imagen, el Lap 2 el tiempo que se tardó en instalar el contendor y el Lap 3 el tiempo que se tardó en iniciar el contenedor.

Figura 3.6 Tiempo de instalación del contenedor Docker.

3.2.3 Instalación máquina virtual por KVM

La máquina virtual se instaló dentro de un host Linux con KVM por lo que se utilizaron comandos Linux KVM para la instalación. En KVM la aplicación virt-manager creó una nueva máquina virtual AlmaLinux a la cual se le especificó el sistema operativo (AlmaLiux 8.6), tamaño de memoria(6G), número de procesadores (4), y tamaño de disco(87G), una vez especificados se indicó la ISO que se deseó instalar y se inició. Fue una instalación desde cero por lo que se siguió un paso a paso de las configuraciones principales de AlmaLinux y se logró instalar correctamente, la conexión remota se la realiza por ssh. La figura 3.7 muestra el entorno gráfico del AlmaLinux instalado por medio de KVM.

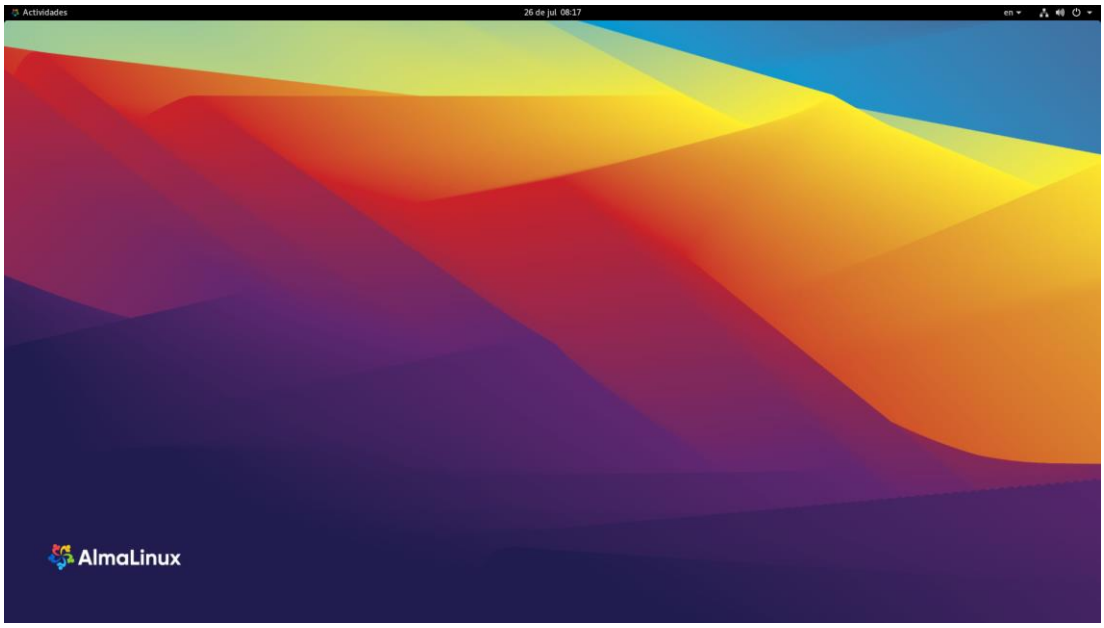


Figura 3.7 Máquina Virtual Iniciada y conectada.

El tiempo de instalación de la máquina virtual AlmaLinux fue de 26 minutos con 27 segundos como se puede observar en la figura 3.8 la cual detalla en el Lap 1 el tiempo que se tardó el virt-manager en crear la máquina, el Lap 2 el tiempo que se tardó en instalar la máquina virtual y el Lap 3 el tiempo que se tardó en iniciar la máquina virtual AlmaLinux.

Figura 3.8 Tiempo de instalación de la máquina virtual AlmaLinux.

3.2.4 Ejecución de la aplicación en Docker y máquina virtual

- **Docker**

Debido a que la imagen Docker previamente posee los elementos necesarios para ejecutar aplicaciones C# solo se procedió a descargar la aplicación creada para testeo. La aplicación se encuentra en GitHub (lizarboleda99/CRUD_LibrosVSC) por lo que se procedió a clonarla. Una vez clonada se la ejecutó y se inició en un tiempo de 1 minuto con 23 seg como se puede observar en la figura 3.9.

Figura 3.9 Tiempo de inicio de la aplicación en Docker.

La figura 3.10 muestra la aplicación ejecutada en el contenedor Docker. Se puede observar que es una librería virtual donde se puede visualizar los libros, agregar un libro, actualizar un libro y borrar un libro.

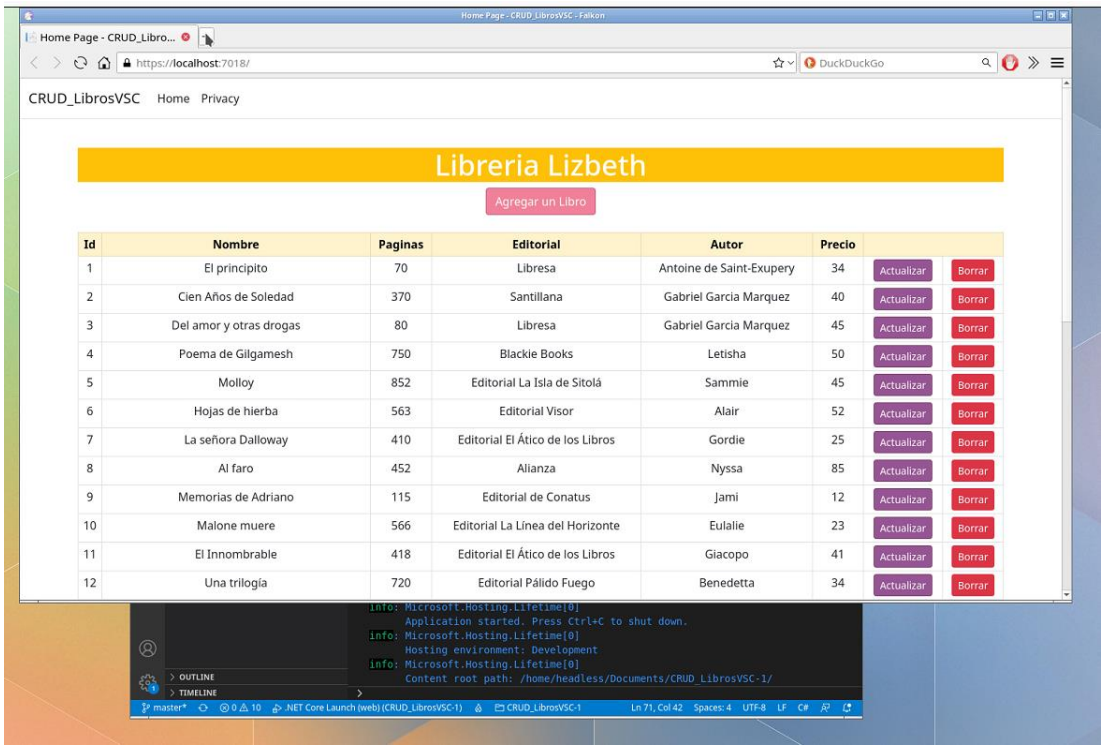


Figura 3.10 Inicio del programa en Docker.

A continuación, se muestra un ejemplo del uso de la aplicación en el contendor Docker, Se agregó, actualizó y eliminó un libro y se tomó el respectivo tiempo de cada uno.

- Agregar Libro

La ejecución de agregar un libro se tardó 27 seg como se puede observar en la figura 3.11.

Figura 3.11 Tiempo en agregar un libro en el programa en Docker.

La figura 3.12 muestra la pantalla de agregar libro dentro del contenedor Docker.

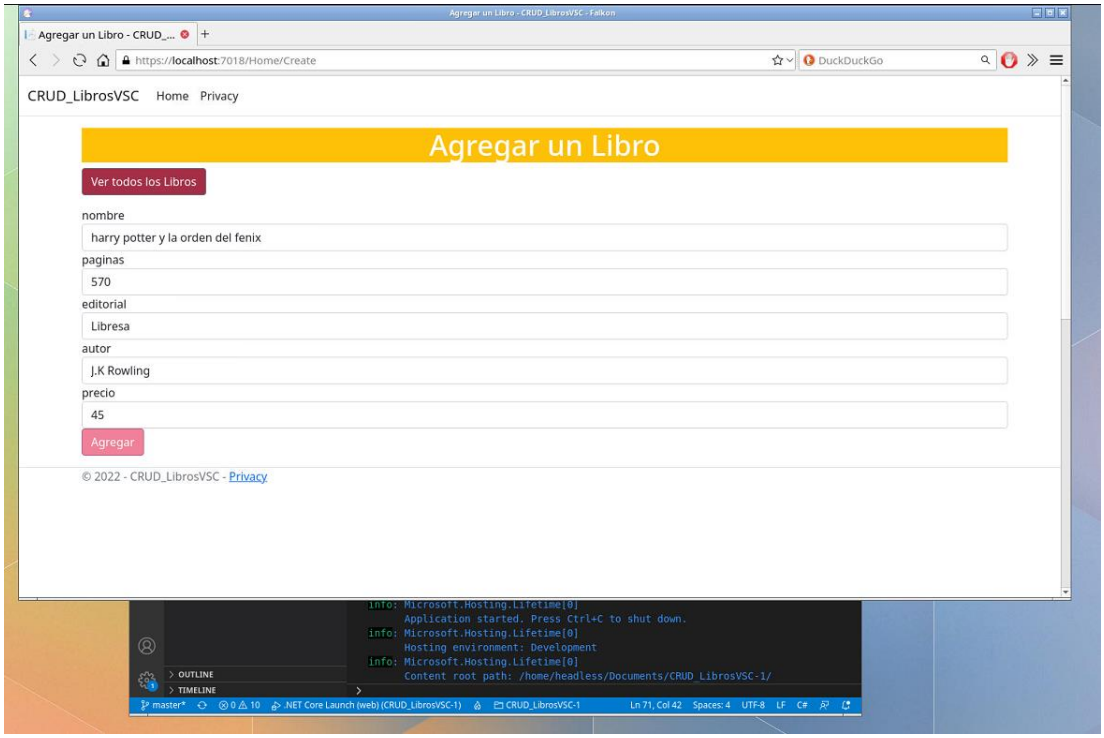


Figura 3.12 Agregar Libro en programa en Docker.

La figura 3.13 muestra un libro ya agregado en la aplicación ejecutada en el contenedor Docker.

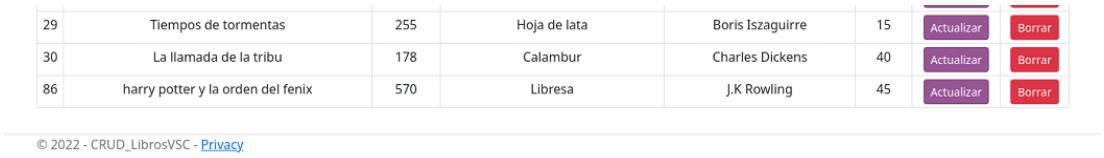


Figura 3.13 Libro Agregado en programa en Docker.

- Actualizar Libro

La ejecución de actualizar libro se tardó 10 seg como se puede observar en la figura 3.14.

Figura 3.14 Tiempo en actualizar un libro en el programa en Docker.

La figura 3.15 muestra la pantalla de actualizar libro de la aplicación dentro de un contenedor Docker.

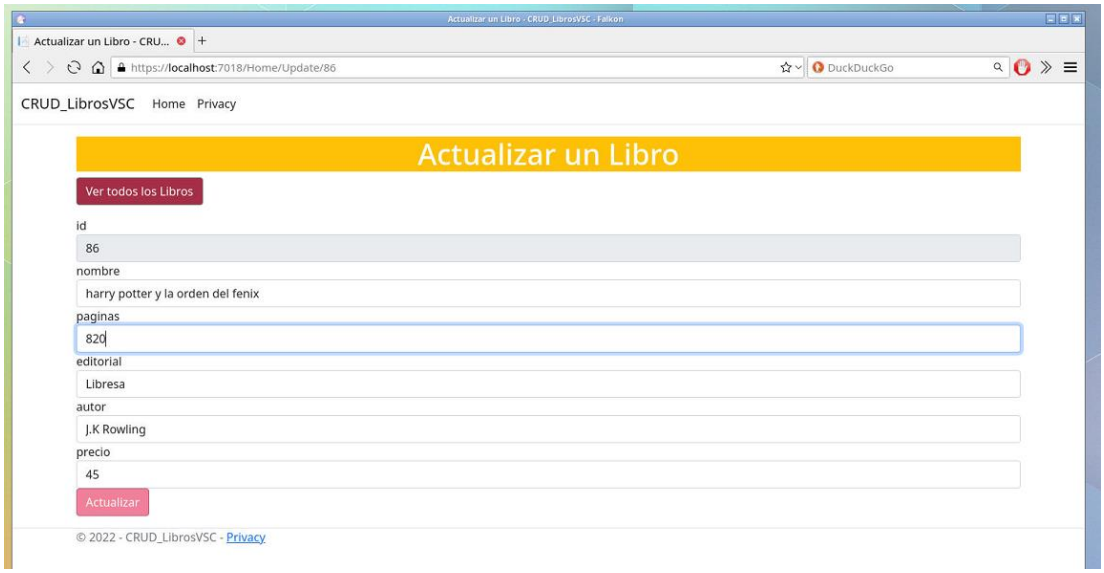


Figura 3.15 Actualizar un libro en el programa en Docker.

La figura 3.16 muestra el libro ya actualizado dentro del programa ejecutado en el contenedor Docker.

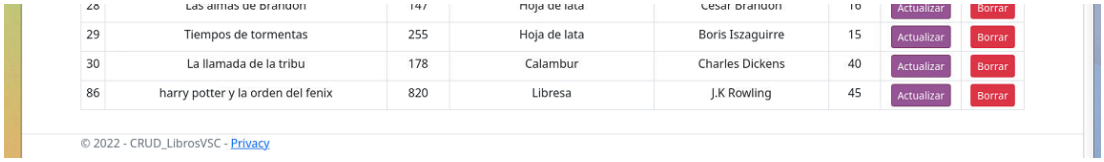


Figura 3.16 Libro Actualizado en el programa en Docker.

- Borrar Libro

La ejecución borrar libro se tardó 4 seg en el contenedor Docker según la figura 3.17.

Figura 3.17 Tiempo en borrar un libro en el programa en Docker.

La figura 3.18 muestra la lista de libros con el libro ya eliminado dentro de la aplicación en un contenedor Docker.

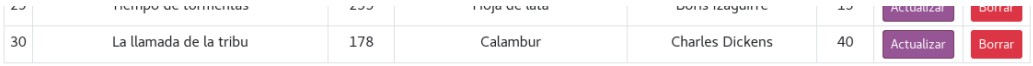
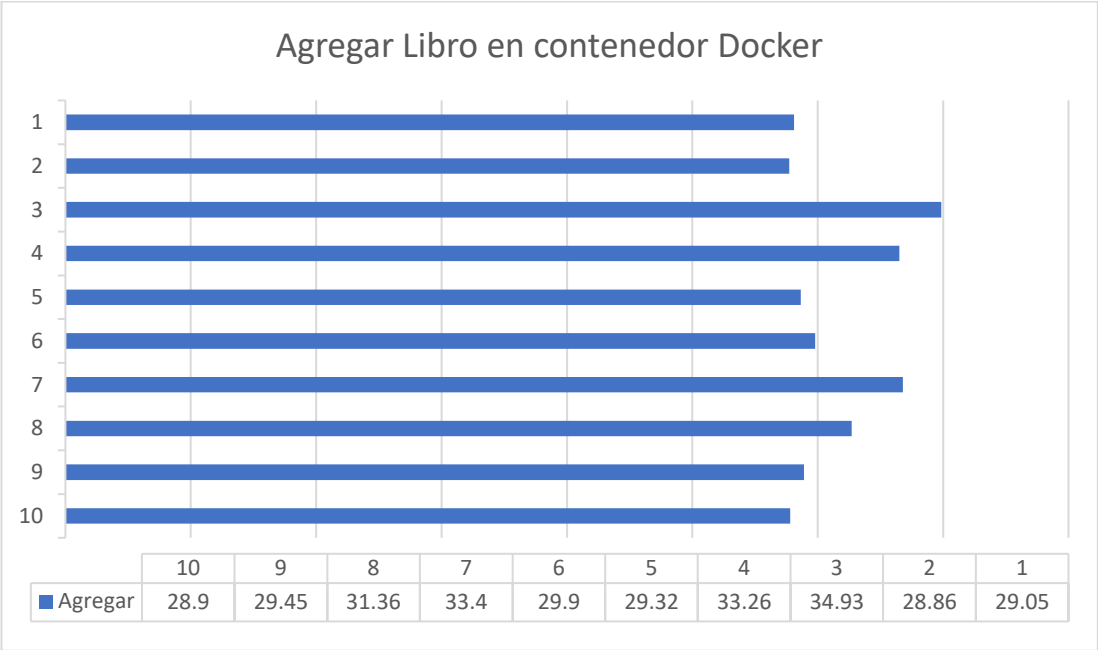


Figura 3.18 Borrar un libro en el programa en Docker.

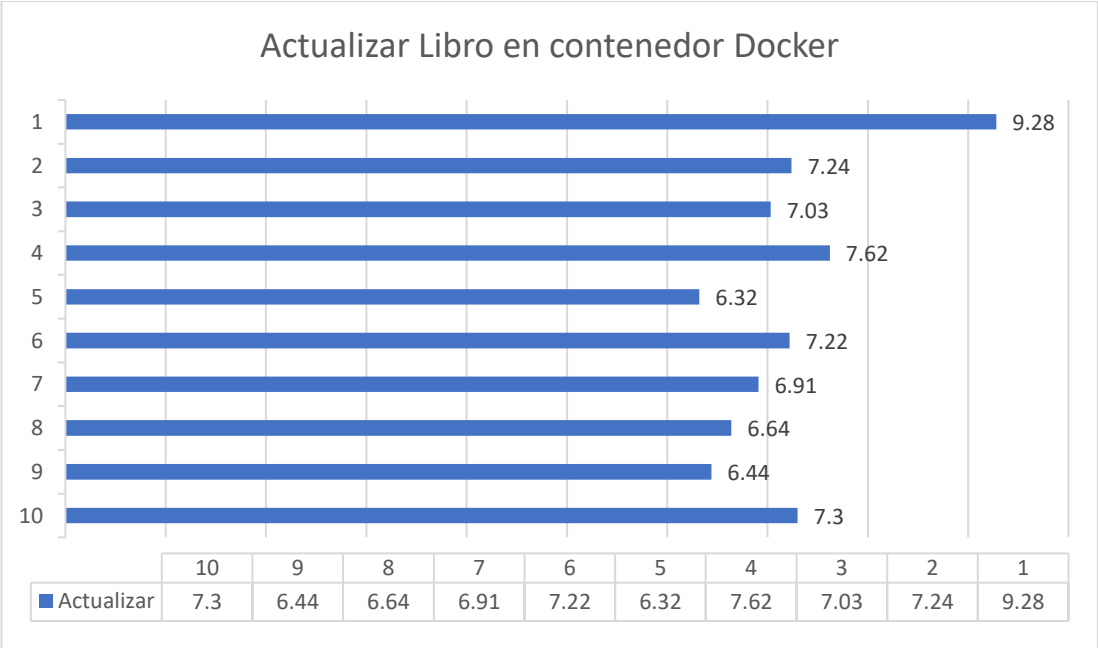
Con base a las figuras 3.9, 3.11, 3.13, 3.15 y 3.17 se determina que para que la aplicación cumpla con todos sus funcionamientos en una máquina virtual debe tardar un tiempo de 2 minutos con 5 segundos y 75 milisegundos (02:05,75).

Este proceso se realizó 10 veces tomando los distintos tiempos para obtener un promedio adecuado de la ejecución de la aplicación en un contenedor Docker y se obtuvieron los resultados de los anexos 3.1, 3.2, 3.3.



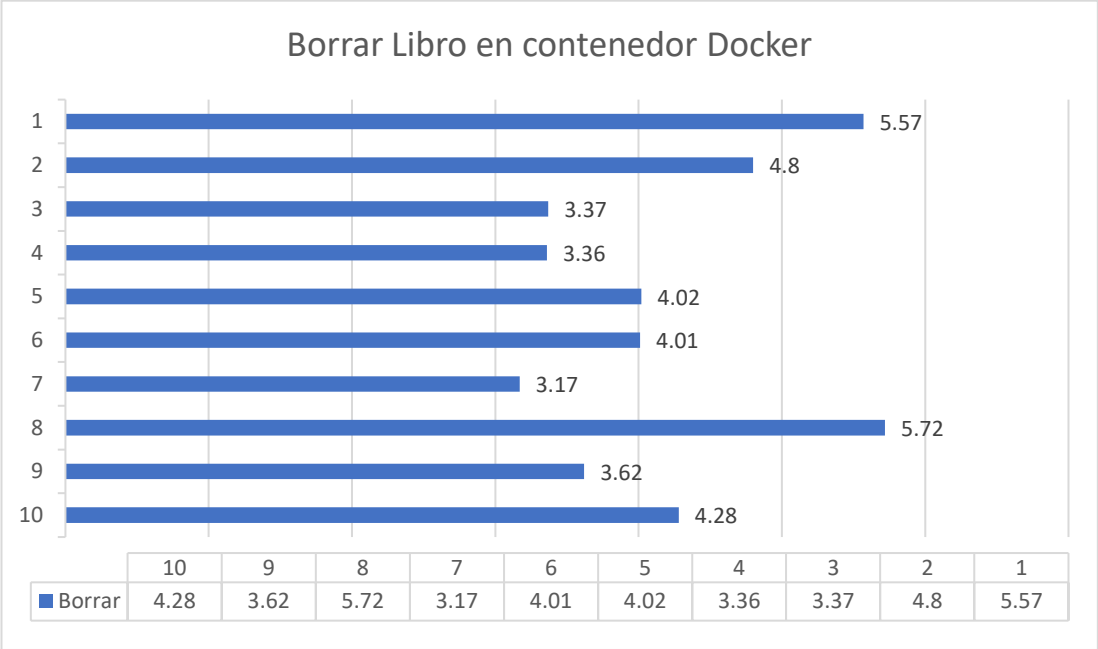
Anexo 3.1 Gráfico de las tomas hechas de agregar libro en contenedor Docker.

Con base al anexo 3.1 se realizó un promedio el cual dio como resultado que dentro de un contenedor Docker el programa en la acción agregar libro se demora alrededor de 28,43 seg.



Anexo 3.2 Gráfico de las tomas hechas de actualizar libro en contenedor Docker.

Con base al anexo 3.2 se realizó un promedio el cual dio como resultado que dentro de un contenedor Docker el programa en la acción actualizar libro se demora alrededor de 7,00 seg.



Anexo 3.3 Gráfico de las tomas hechas de borrar libro en contenedor Docker.

Con base al anexo 3.3 se realizó un promedio el cual dio como resultado que dentro de un contenedor Docker el programa en la acción borrar libro se demora alrededor de 4,92 seg.

- **Máquina Virtual**

Debido a que la máquina virtual es una instalación Linux desde cero se debe instalar todas las dependencias necesarias para poder ejecutar la aplicación C#, para esto se instaló Visual Studio Code en el cual se instaló la extensión para aplicaciones C#. Una vez instalada la extensión se procedió a instalar dotnet para Linux ya que sin el SDK de dotnet la aplicación no correría. Finalizada las instalaciones necesarias se procedió a clonar la aplicación de GitHub (lizarboleda99/CRUD_LibrosVSC) para testearla. El

tiempo de instalación de las dependencias necesarias fue de 48 minutos. Y el tiempo de inicio de la aplicación clonada fue de 1 minuto con 27 seg como se puede observar en la figura 3.19.

Figura 3.19 Tiempo de inicio de programa en AlmaLinux.

La figura 3.20 muestra la aplicación para testear dentro de la máquina virtual AlmaLinux. Se puede observar que es una librería virtual donde se puede visualizar los libros, agregar un libro, actualizar un libro y borrar un libro.

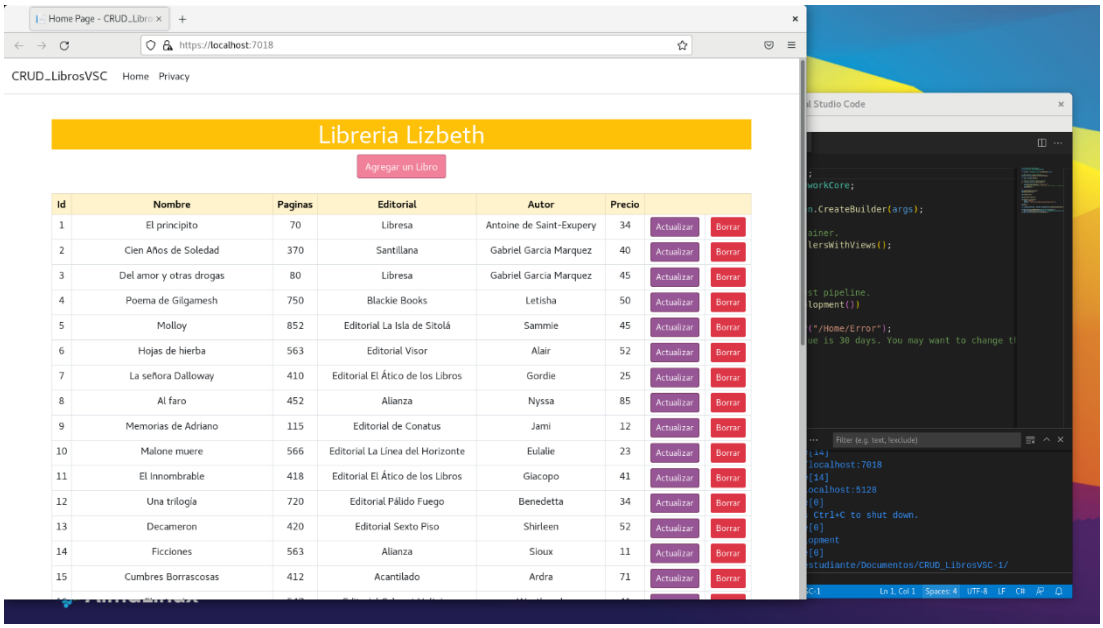


Figura 3.20 Inicio de Programa en AlmaLinux.

A continuación, se muestra un ejemplo del uso de la aplicación en la máquina virtual AlmaLinux. Se agregó, actualizó y eliminó un libro y se tomó el respectivo tiempo de cada uno.

- Agregar Libro

La ejecución de agregar libro tardó 35 seg como se puede observar en la figura 3.21.

Figura 3.21 Tiempo en agregar un libro en el programa en AlmaLinux.

La figura 3.22 muestra la pantalla de agregar libro dentro de la máquina virtual AlmaLinux.

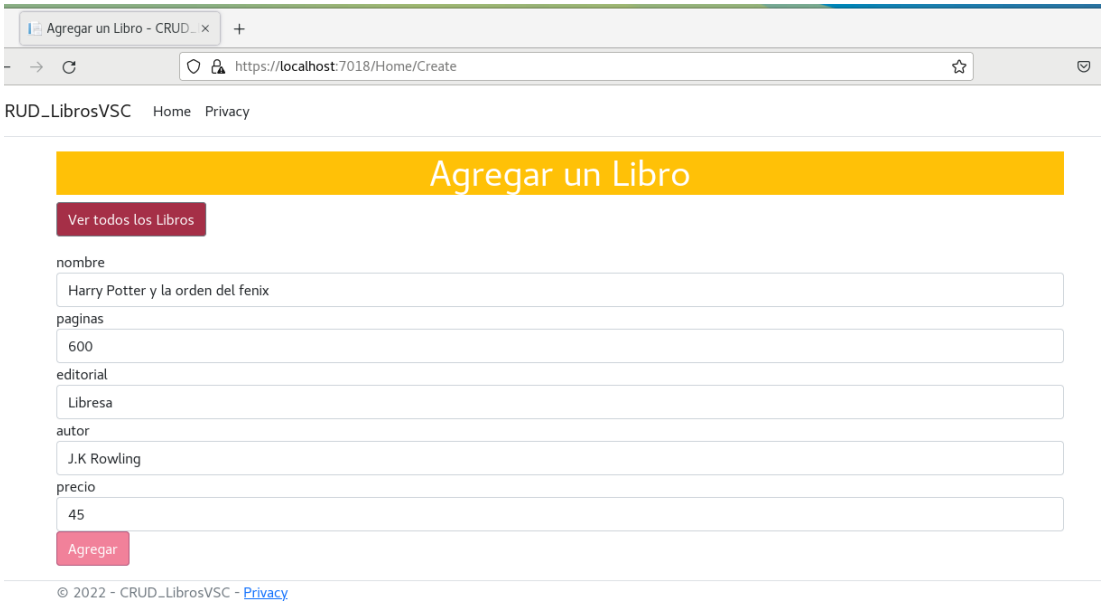
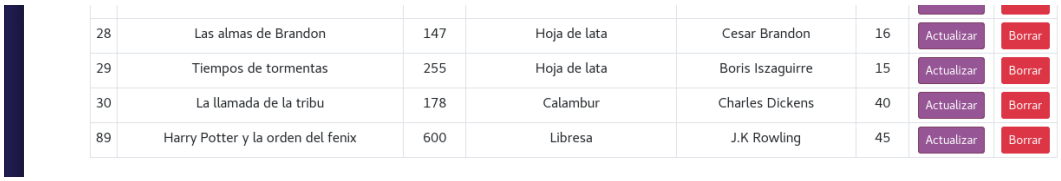


Figura 3.22 Agregar Libro en el programa en AlmaLinux.

La figura 3.23 muestra un libro ya agregado en la aplicación ejecutada en la máquina virtual AlmaLinux.



28	Las almas de Brandon	147	Hoja de lata	Cesar Brandon	16	Actualizar	Borrar
29	Tiempos de tormentas	255	Hoja de lata	Boris Iszaguirre	15	Actualizar	Borrar
30	La llamada de la tribu	178	Calambur	Charles Dickens	40	Actualizar	Borrar
89	Harry Potter y la orden del fenix	600	Libresa	J.K Rowling	45	Actualizar	Borrar

Figura 3.23 Libro Agregado en el programa en Almalinux.

- Actualizar Libro

La ejecución de actualizar libro tardó 14 seg como se puede observar en l figura 3.24.

Figura 3.24 Tiempo en actualizar un libro en el programa en AlmaLinux.

La figura 3.25 muestra la pantalla de actualizar libro de la aplicación dentro de la máquina virtual AlmaLinux.

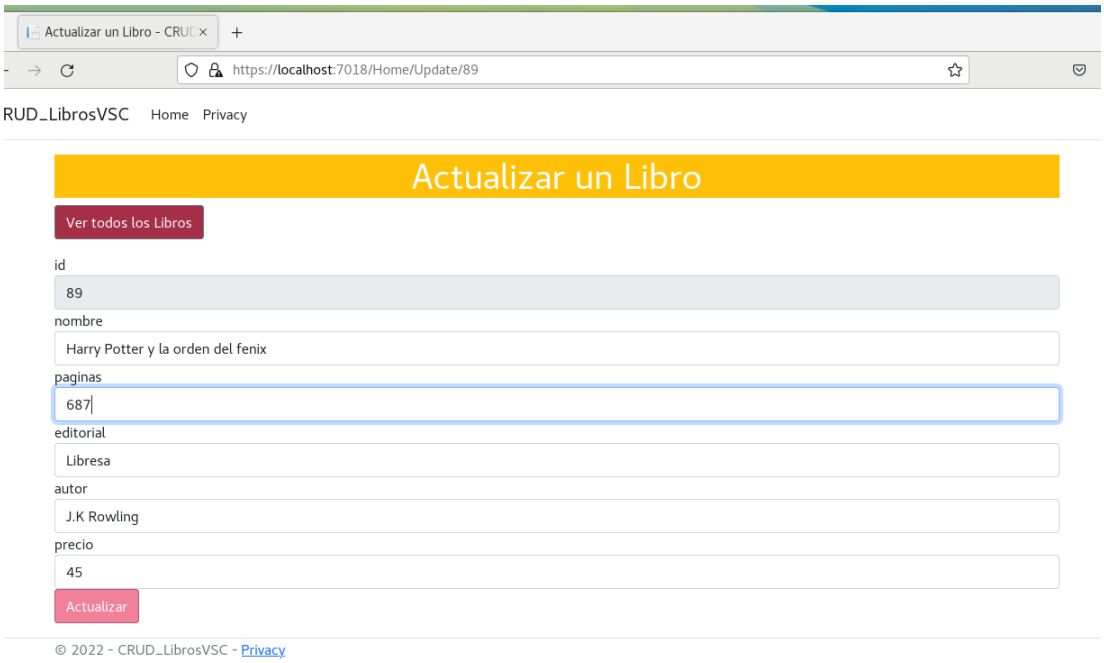


Figura 3.25 Actualizar Libro en el programa en AlmaLinux.

La figura 3.26 muestra el libro ya actualizado dentro del programa ejecutado en la máquina virtual AlmaLinux.

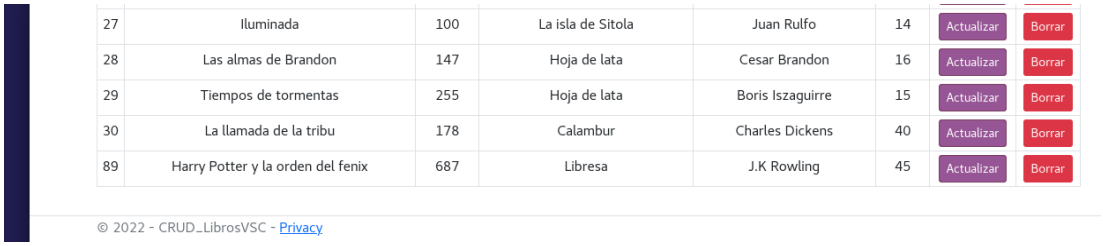


Figura 3.26 Libro Actualizado en el programa en AlmaLinux.

- Borrar Libro

La ejecución de borrar libro tardó 7 seg como se puede observar en la figura 3.27.

Figura 3.27 Tiempo en borrar un libro en el programa en AlmLinux.

La figura 3.28 muestra la lista de libros con el libro ya eliminado dentro de la aplicación en la máquina virtual AlmaLinux.

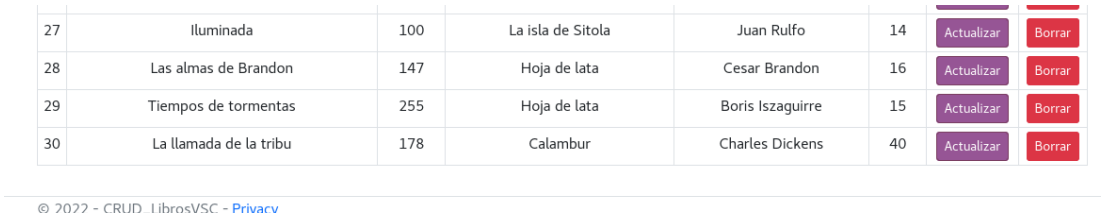
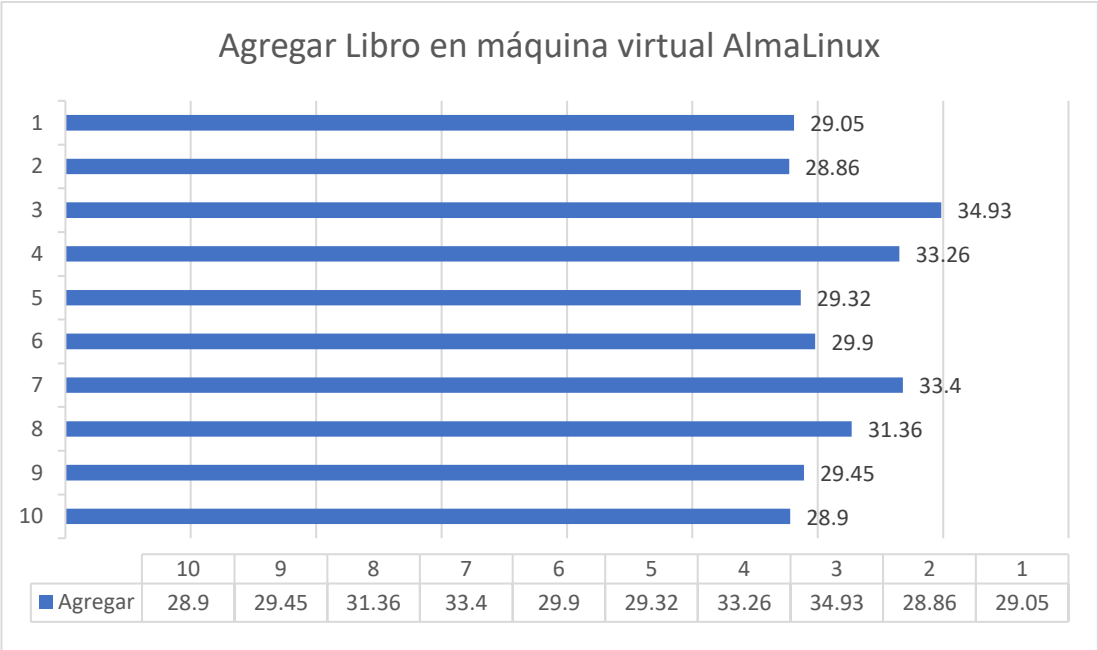


Figura 3.28 Borrar Libro en el programa en AlmaLinux.

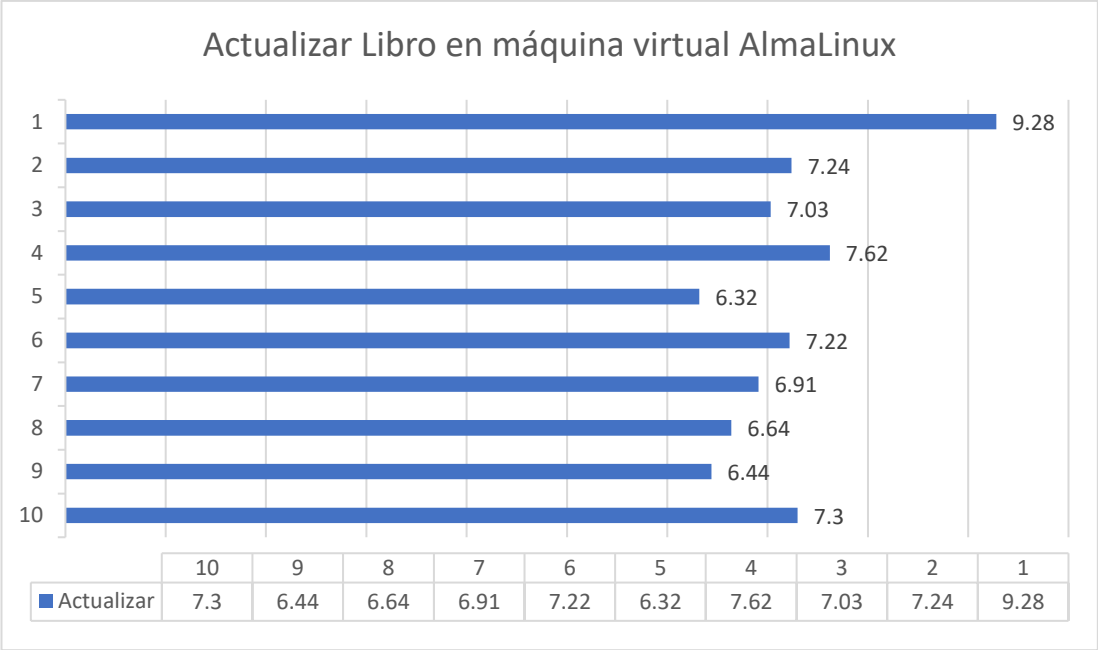
Con base a las figuras 3.19, 3.21, 3.23, 3.25 y 3.27 se determina que para que la aplicación cumpla con todos sus funcionamientos en una máquina virtual debe tardar un tiempo de 2 minutos con 18 segundos y 17 milisegundos (02:18,17).

Este proceso se realizó 10 veces tomando los distintos tiempos para obtener un promedio adecuado de la ejecución de la aplicación en la máquina virtual AlmaLinux y se obtuvieron los resultados de los anexos 3.4, 3.5, 3.6.



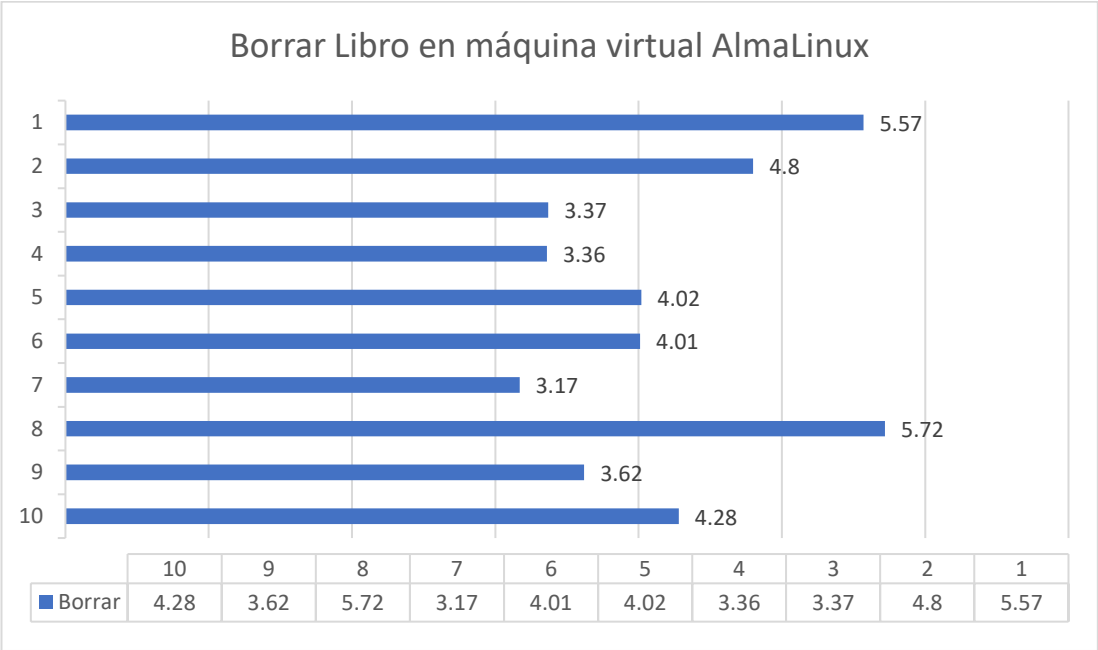
Anexo 3.4 Gráfico de las tomas hechas de agregar libro en máquina virtual AlmaLinux.

Con base al anexo 3.4 se realizó un promedio el cual dio como resultado que dentro de la máquina virtual AlmaLinux el programa en la acción agregar libro se demora alrededor de 35,86 seg.



Anexo 3.5 Gráfico de las tomas hechas de actualizar libro en máquina virtual AlmaLinux.

Con base al anexo 3.5 se realizó un promedio el cual dio como resultado que dentro de la máquina virtual AlmaLinux el programa en la acción actualizar libro se demora alrededor de 7,19 seg.



Anexo 3.6 Gráfico de las tomas hechas de borrar libro en máquina virtual AlmaLinux.

Con base al anexo 3.6 se realizó un promedio el cual dio como resultado que dentro de la máquina virtual AlmaLinux el programa en la acción borrar libro se demora alrededor de 4,32 seg.

Una vez tomado los datos de la aplicación tanto en contenedor Docker como en máquina virtual AlmaLinux se procedió a realizar la respectiva comparación en la tabla 3.6 en cuanto a las acciones que la aplicación realiza que son: inicio de aplicación, agregar, actualizar y borrar libro.

Tabla 3.6 Comparación ejecución CRUD en Docker y máquina virtual.

EJECUCIÓN CRUD		
	DOCKER	**MÁQUINA VIRTUAL**
INICIAR APLICACIÓN	1 minuto 23 seg 40 milisegundos	1 minuto 27 seg 60 milisegundos
AGREGAR LIBRO	28 seg 43 milisegundos	35 seg 86 milisegundos
ACTUALIZAR LIBRO	7 seg	7 seg 19 milisegundos
BORRAR LIBRO	4 seg 92 milisegundos	4 seg 32 milisegundos
TIEMPO TOTAL	2 minutos 3 seg 75 milisegundos	2 minutos 14 seg 97 milisegundos

Elaborado por: El Investigador

Con base a la tabla 3.6 se puede deducir que en un contenedor Docker los tiempos de ejecución de las diferentes acciones de la aplicación a testear son más rápidas que en la máquina virtual AlmaLinux.

3.2.5 Toma de datos de rendimiento, velocidad, tiempo de arranque

Para todo este tipo de prueba se tuvieron abiertas las mismas aplicaciones tanto en Docker como en máquina virtual AlmaLinux, es decir, se estuvo ejecutando Visual Studio Code, localhost con la aplicación web a testear, Stacer y una terminal.

- **Glances**

Se instaló Glances tanto en el contenedor como en la máquina virtual para monitorear el rendimiento de cpu, memoria y disco.

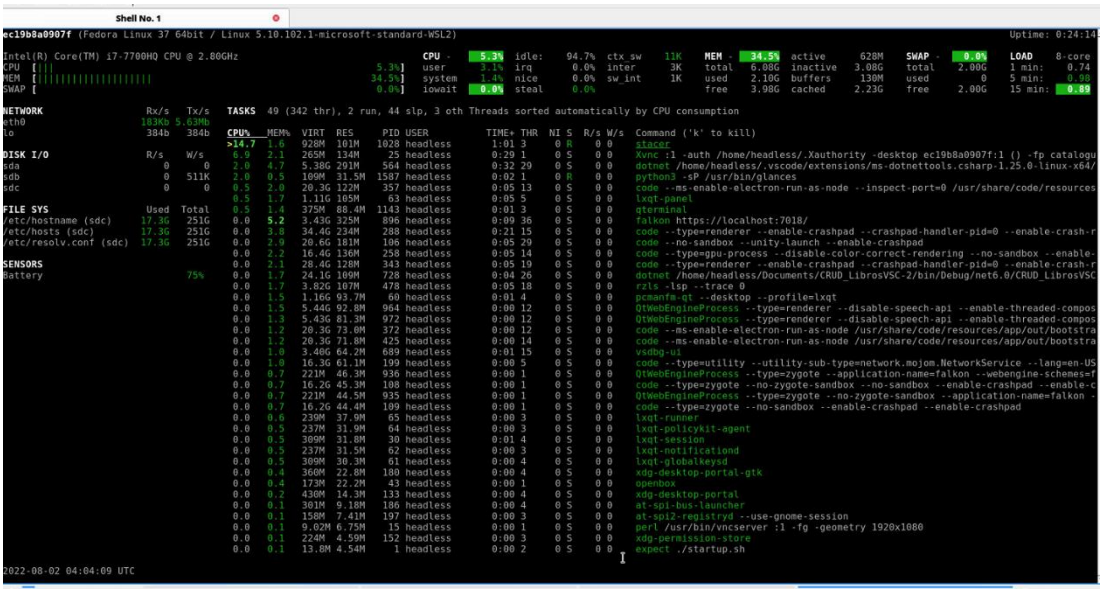
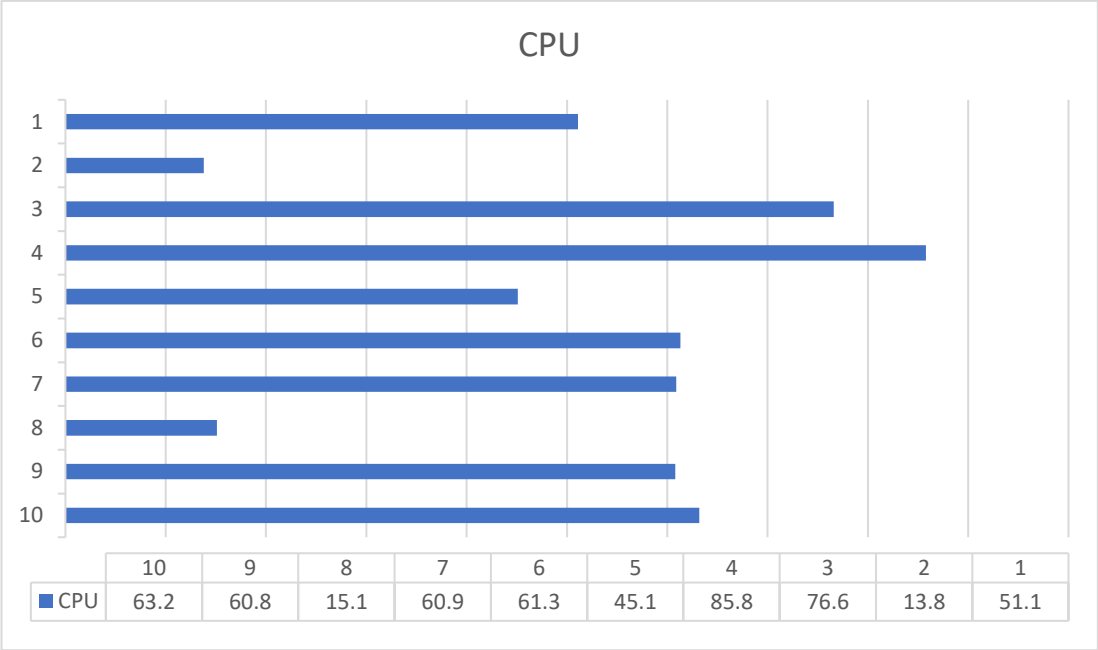


Figura 3.29 Glances en Docker.

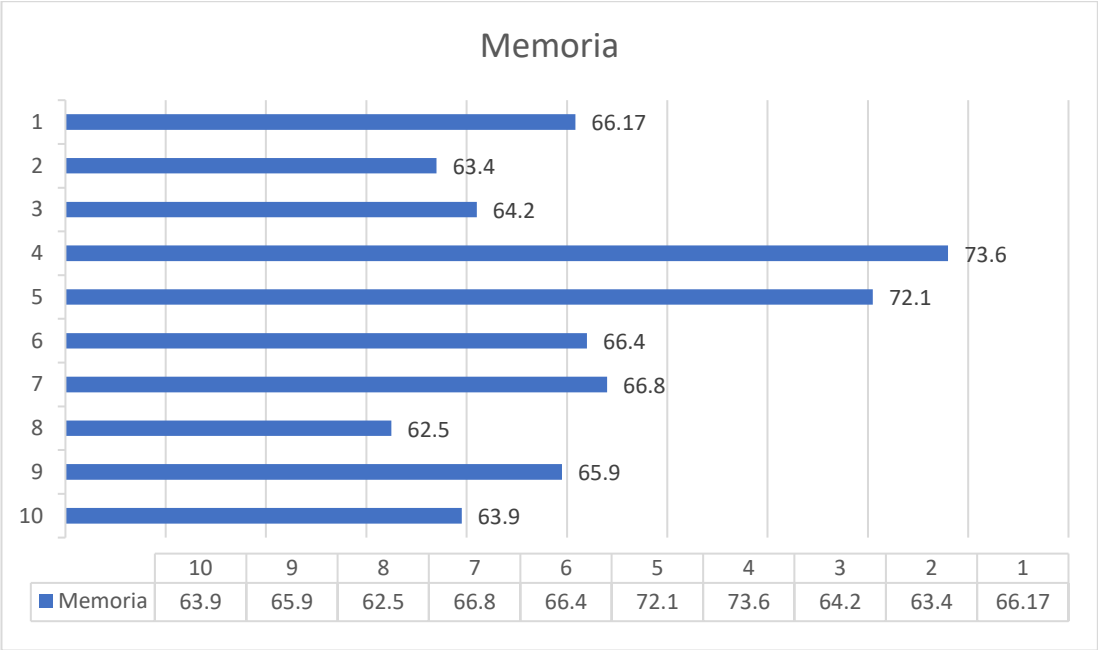
La figura 3.29 es un ejemplo de cómo se monitoreo con Glances el contenedor Docker y se obtuvo los siguientes datos: el uso de cpu es de 5.3% el cual se divide en 3.1% en usuario y 1.4% en sistema lo cual explica que usa lo mínimo para que funcione. En cuanto a memoria muestra que está en uso un 34.5% dividido en activo 268M y que en total en todas las instalaciones previas se ha usado 2.10GB de 6 GB. En cuanto a disco las distros Linux usan SWAP en remplazo a una RAM y en este caso no usa nada de los 2GB disponibles para uso. En cuanto a la aplicación en Visual Studio Code se puede observar que ocupa el 2.0 de cpu, 4.7 de memoria, 5.38G de memoria virtual, 291M de memoria residente y en cuanto a la aplicación ejecutada en localhost se puede observar que ocupa el 0.0 de cpu, 5.2 de memoria, 3.43G de memoria virtual, 325M de memoria residente.

A partir del análisis hecho anteriormente se monitoreo 10 veces el rendimiento del contenedor Docker se obtuvieron los siguientes datos:



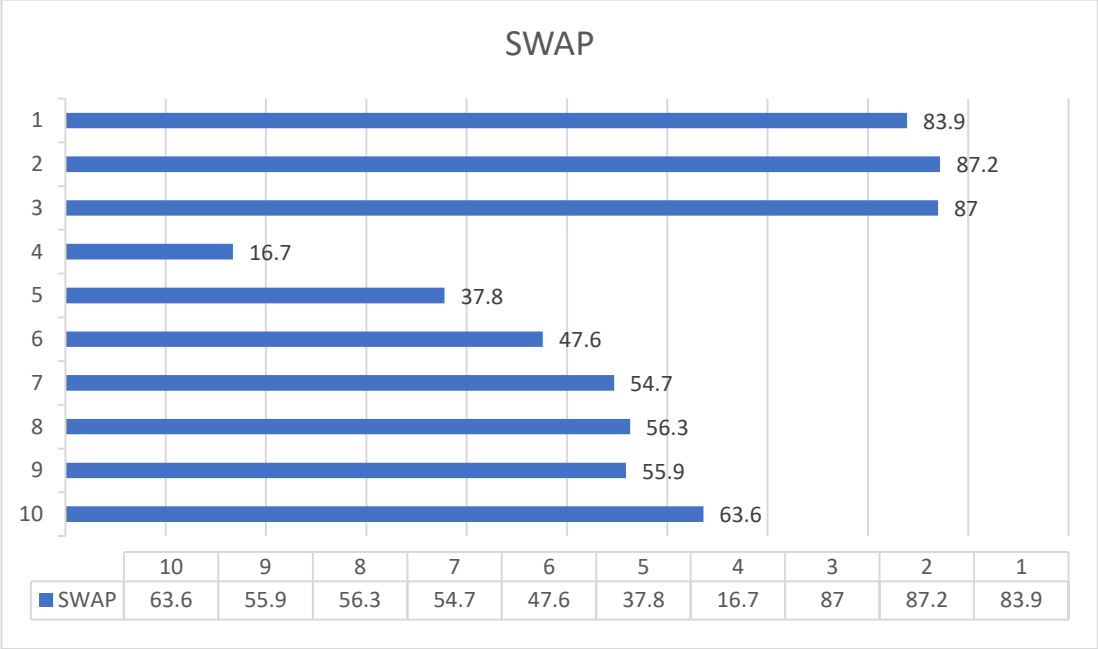
Anexo 3.7 Gráfico porcentaje CPU de Glances en Docker.

Con base al anexo 3.7 se realizó un promedio el cual dio como resultado que el rendimiento de CPU dentro de un contendor Docker es de 53.37%.



Anexo 3.8 Gráfico porcentaje Memoria de Glances en Docker.

Con base al anexo 3.8 se realizó un promedio el cual dio como resultado que el uso de memoria dentro de un contendor Docker es de 66.497%.



Anexo 3.9 Gráfico porcentaje SWAP de Glances en Docker.

Con base al anexo 3.9 se realizó un promedio el cual dio como resultado que el uso de memoria SWAP dentro de un contendor Docker es de 59.07%.

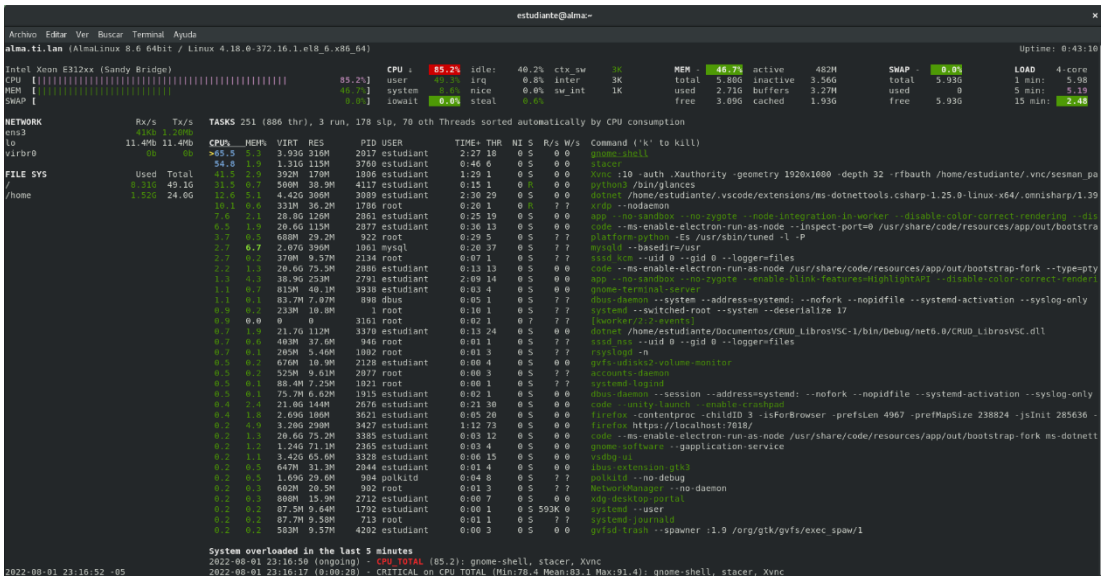
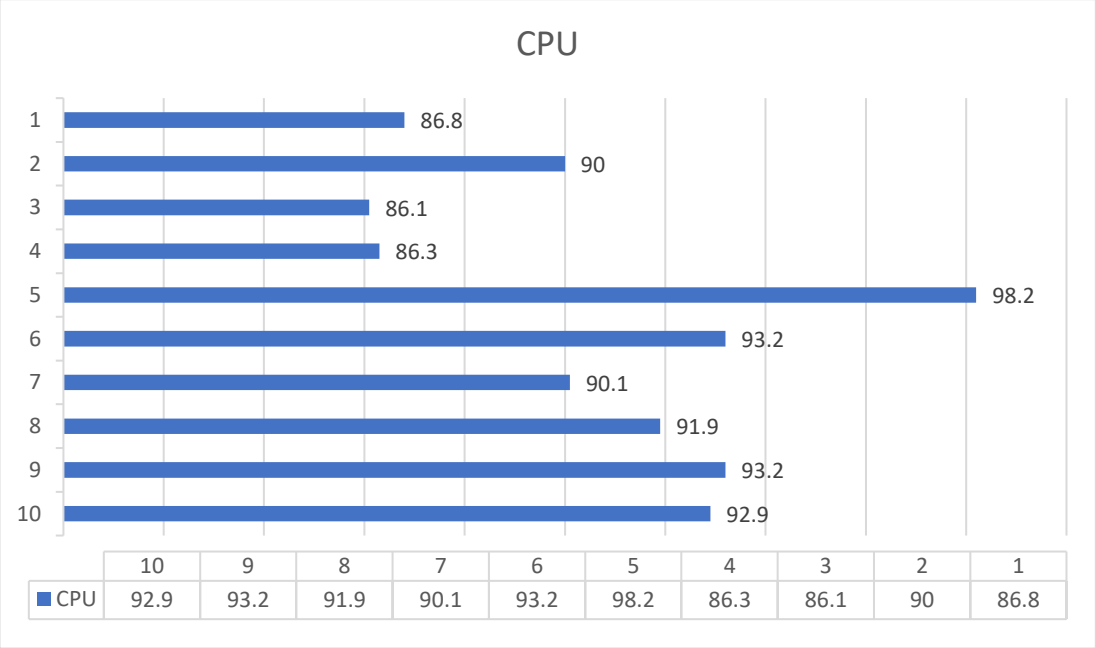


Figura 3.30 Glances en AlmaLinux.

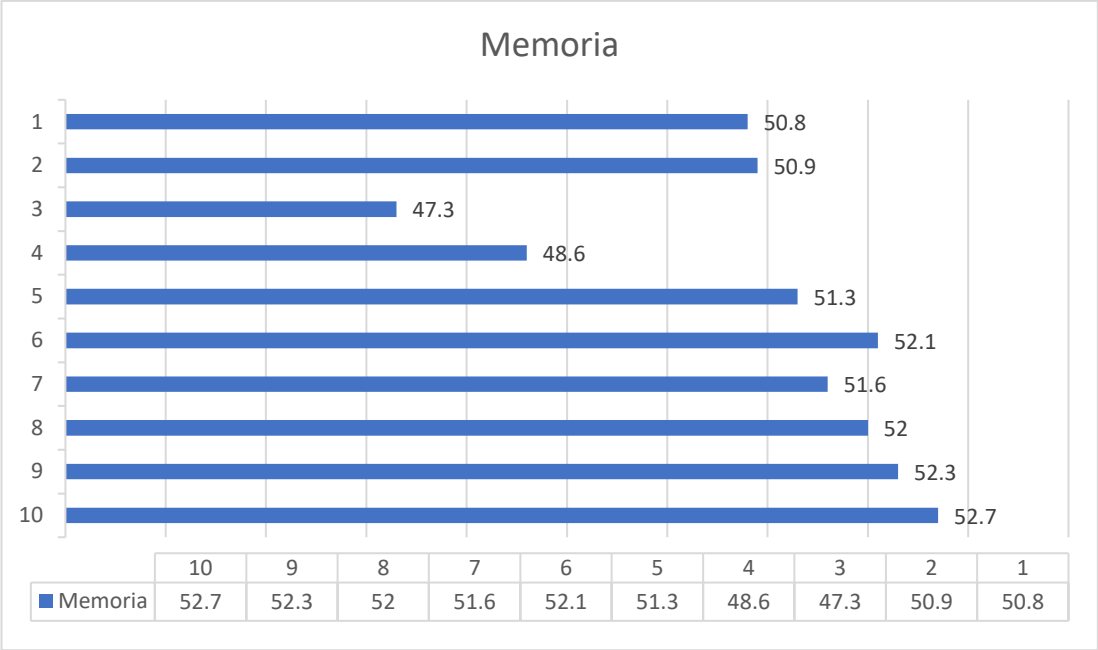
La figura 3.30 es un ejemplo de cómo se monitoreo con Glances en una máquina virtual y se obtuvo los siguientes datos: el uso de cpu es de 85.2% el cual se divide en 49.3% en usuario y 8.6%. En cuanto a memoria muestra que está en uso un 46.7% dividido en activo 482M y que en total en todas las instalaciones previas se ha usado 2.71GB de 5.80 GB. En cuanto a disco el SWAP no usa nada de los 5.93GB disponibles para uso. En cuanto a la aplicación en Visual Studio Code se puede observar que ocupa el 0.7 de cpu, 1.9 de memoria, 21.7G de memoria virtual, 112M de memoria residente y en cuanto a la aplicación ejecutada en localhost se puede observar que ocupa el 0.2 de cpu, 4.9 de memoria, 3.20G de memoria virtual, 290M de memoria residente.

A partir del análisis hecho anteriormente se monitoreo 10 veces el rendimiento de la máquina virtual AlmaLinux y se obtuvieron los siguientes datos:



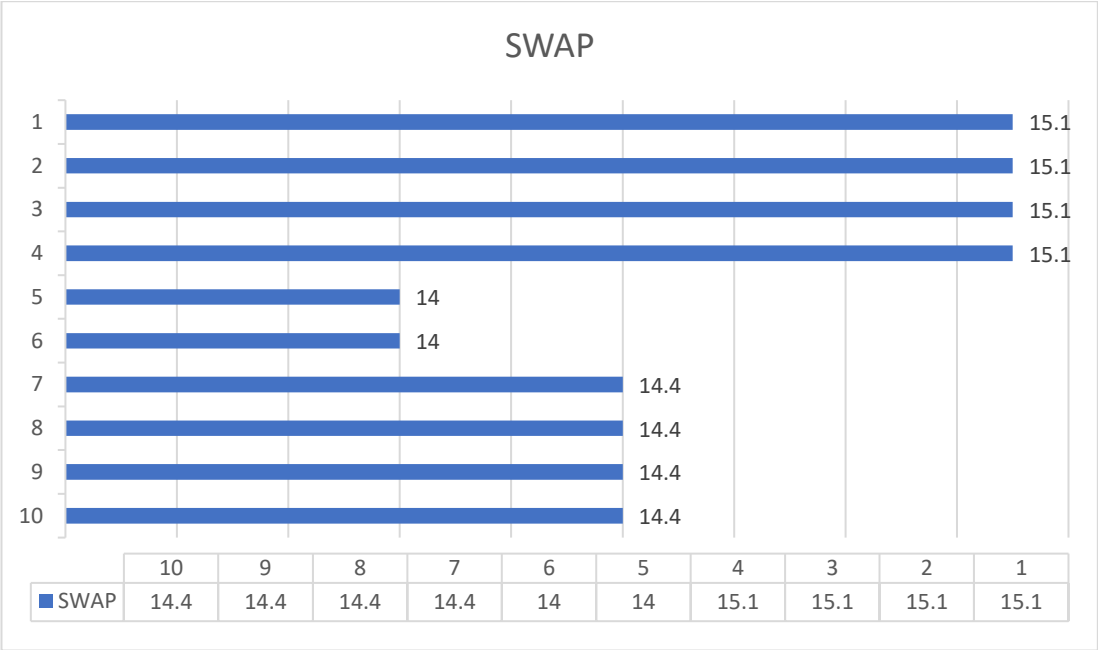
Anexo 3.10 Gráfico porcentaje CPU de Glances en AlmaLinux.

Con base al anexo 3.10 se realizó un promedio el cual dio como resultado que el rendimiento de CPU dentro de la máquina virtual AlmaLinux es de 90.87%.



Anexo 3.11 Gráfico porcentaje Memoria de Glances en AlmaLinux.

Con base al anexo 3.11 se realizó un promedio el cual dio como resultado que el uso de memoria dentro de la máquina virtual AlmaLinux es de 50.96%.



Anexo 3.12 Gráfico porcentaje SWAP de Glances en AlmaLinux.

Con base al anexo 3.12 se realizó un promedio el cual dio como resultado que el uso de memoria SWAP dentro de la máquina virtual AlmaLinux es de 14.6%.

Tabla 3.7 Comparación Glances Docker y Máquina Virtual

GLANCES		
	DOCKER	**MÁQUINA VIRTUAL**
CPU	53.37%.	90.87%.
MEMORIA	66.497%.	50.96%.
SWAP	59.07%.	14.6%.

Elaborado por: El investigador

Con base a la tabla 3.7 se puede deducir que en cuanto a rendimiento de CPU Docker usa lo mínimo para que funcione mientras que la máquina virtual consume casi el 100% pero en cuanto a memoria y disco la máquina virtual funciona de mejor forma que el contenedor Docker.

- **Htop**

Htop ya viene instalado en la mayoría de las distros Linux por lo que solo fue necesario ejecutarlo.

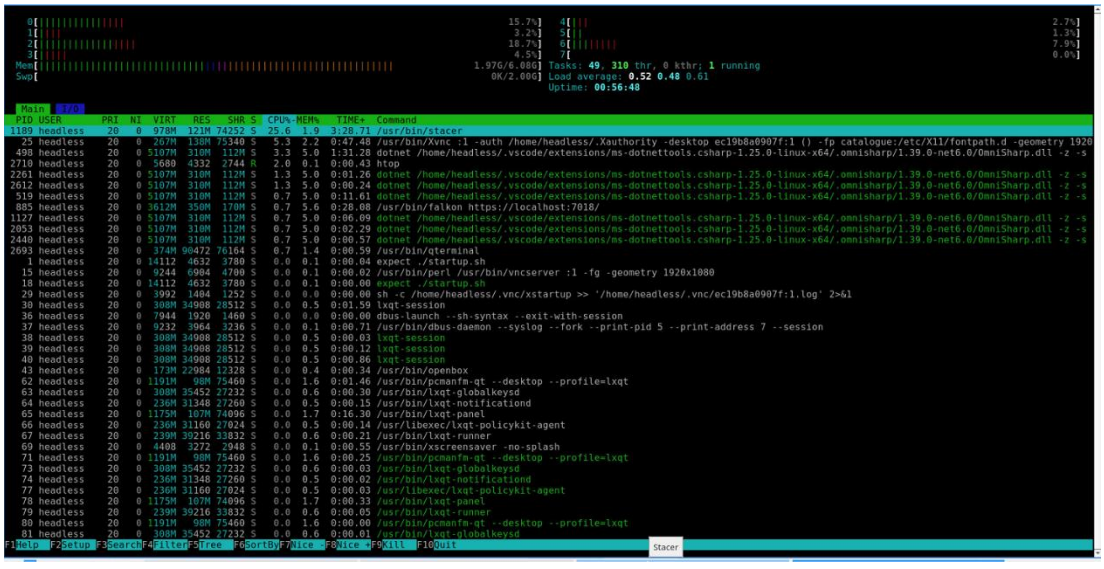
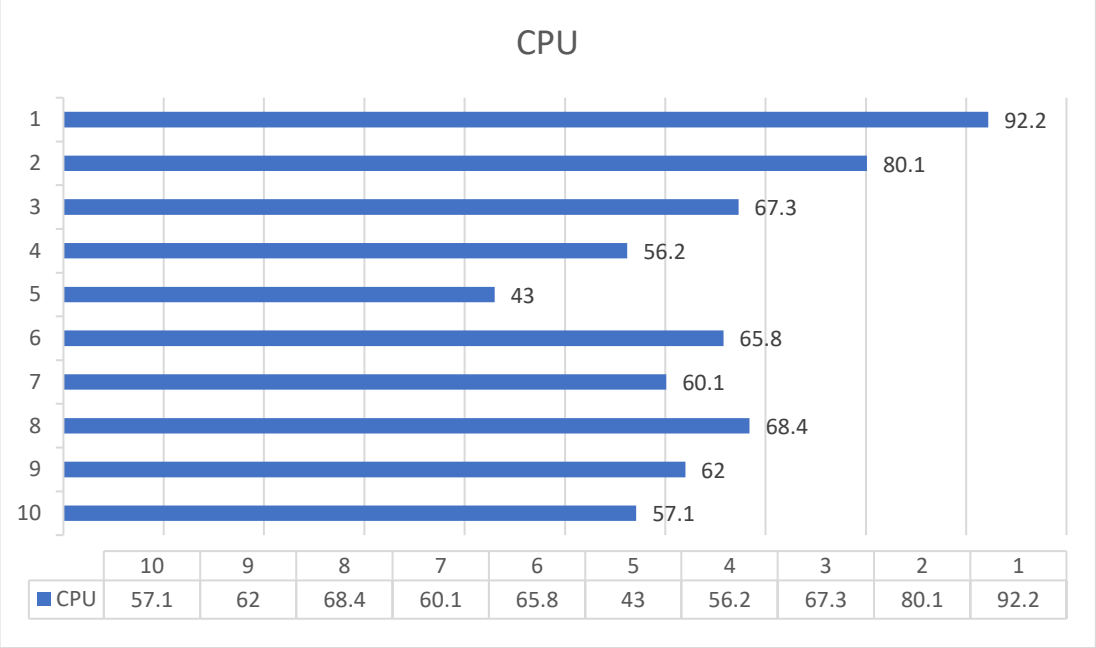


Figura 3.31 Htop en Docker.

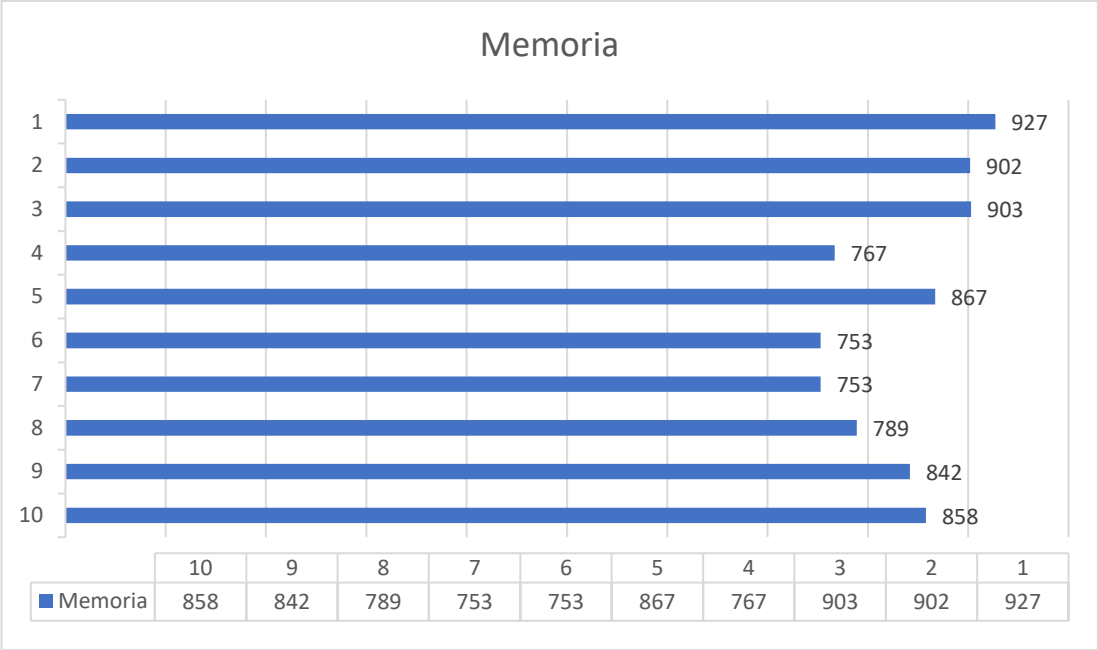
La figura 3.31 es un ejemplo de cómo se monitoreo con htop el contenedor Docker y se obtuvo los siguientes datos: El uso de cpu es del 54% dividido en 7 tareas principales, el uso de memoria es de 1.97 G de 6.08 G, y no existe uso de SWAP de los 2 G disponibles que tiene. En cuanto a la ejecución de Visual Studio Code en cpu el uso es de 3.3, memoria es de 5.0, memoria virtual es de 5107M y 310M de memoria residente. En cuanto a la ejecución de la aplicación en localhost el uso de cpu es de 0.7, de memoria es de 5.6, de memoria virtual es de 3612M y de memoria residente es de 350M.

A partir del análisis hecho anteriormente se monitoreo 10 veces el rendimiento del contenedor Docker se obtuvieron los siguientes datos:



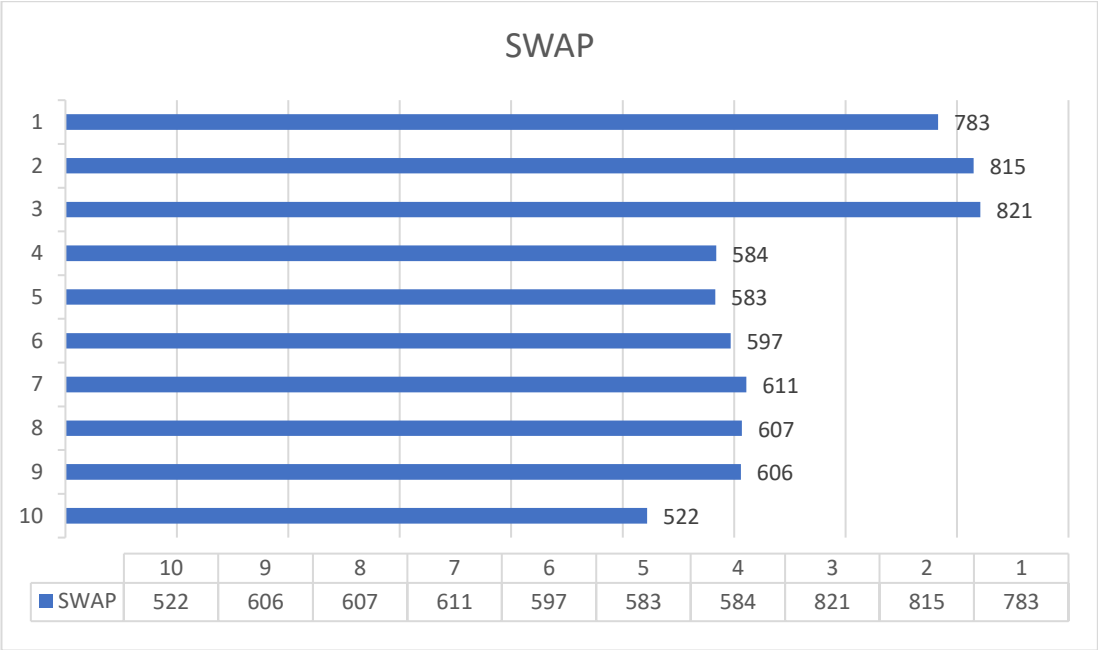
Anexo 3.13 Gráfico porcentaje CPU de htop en Docker.

Con base al anexo 3.13 se realizó un promedio el cual dio como resultado que el rendimiento de CPU dentro de un contendor Docker es de 65.22%.



Anexo 3.14 Gráfico porcentaje Memoria de htop en Docker.

Con base al anexo 3.14 se realizó un promedio el cual dio como resultado que el uso de memoria dentro de un contendor Docker es de 836.1 M.



Anexo 3.15 Gráfico porcentaje SWAP de htop en Docker.

Con base al anexo 3.15 se realizó un promedio el cual dio como resultado que el uso de memoria SWAP dentro de un contendor Docker es de 652.9M.

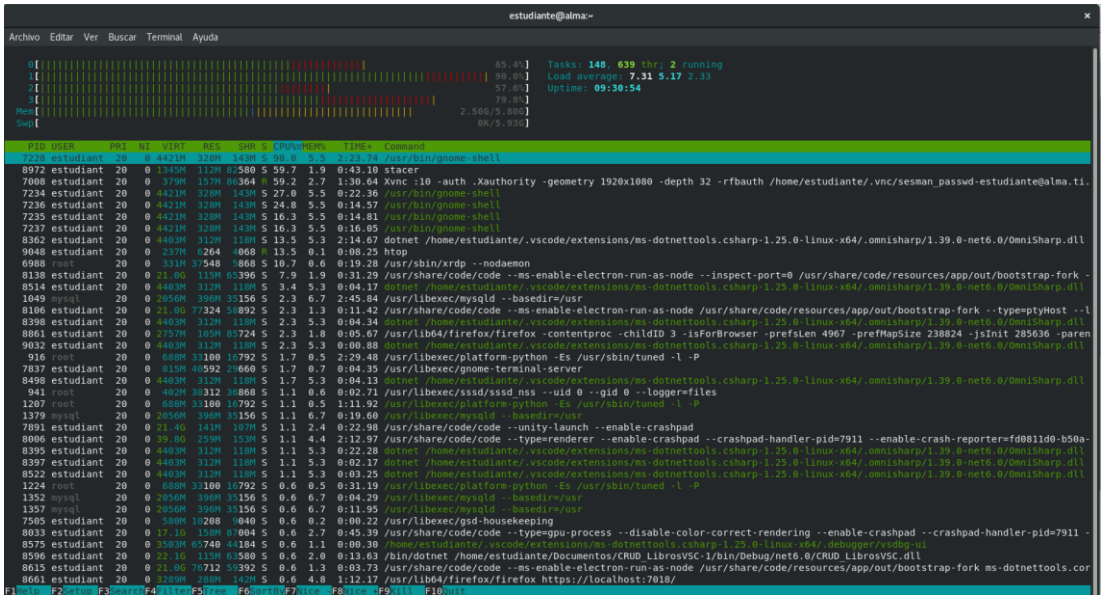
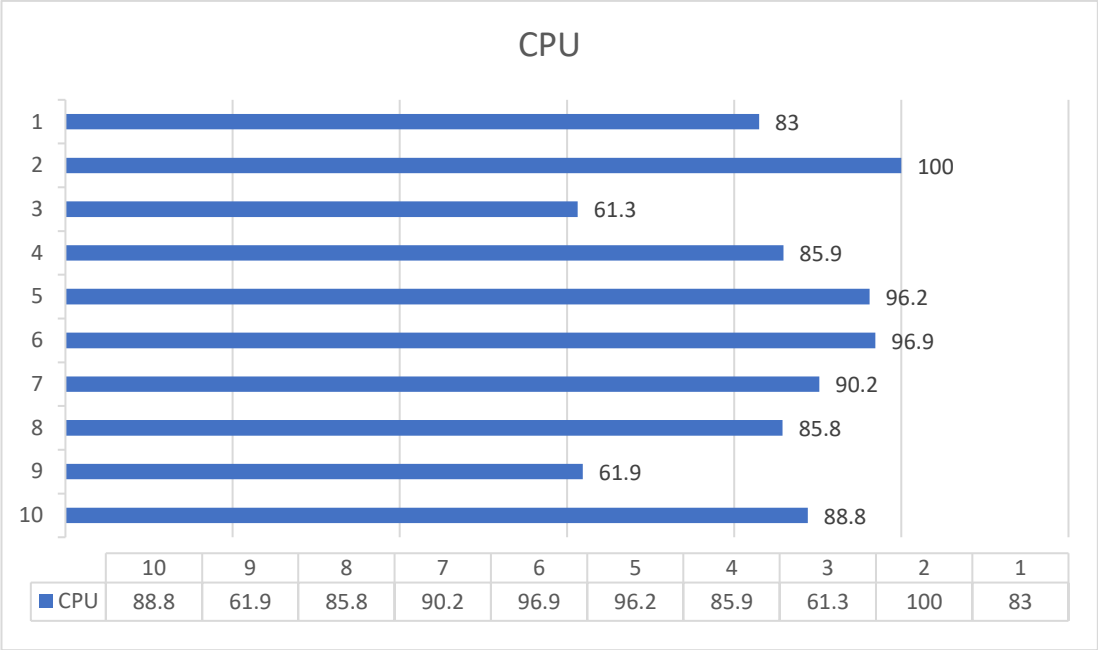


Figura 3.32 Htop en AlmaLinux.

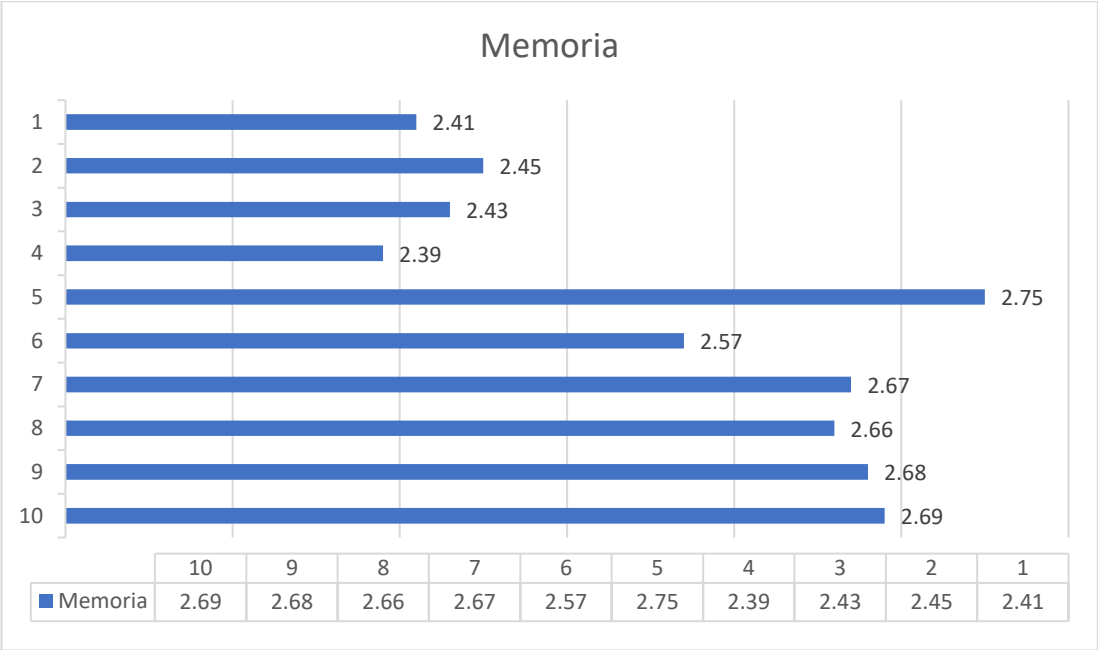
La figura 3.32 es un ejemplo de cómo se monitoreo con htop en una máquina virtual y se obtuvo los siguientes datos: El uso de cpu es del 292.8% dividido en 4 tareas principales, el uso de memoria es de 2.50 G de 5.80 G, y no existe uso de SWAP de los 5.93 G disponibles que tiene. En cuanto a la ejecución de Visual Studio Code en cpu el uso es de 13.5, memoria es de 5.3, memoria virtual es de 4403M y 312M de memoria residente. En cuanto a la ejecución de la aplicación en localhost el uso de cpu es de 0.6, de memoria es de 4.8, de memoria virtual es de 3289M y de memoria residente es de 288M.

A partir del análisis hecho anteriormente se monitoreo 10 veces el rendimiento de la máquina virtual AlmaLinux y se obtuvieron los siguientes datos:



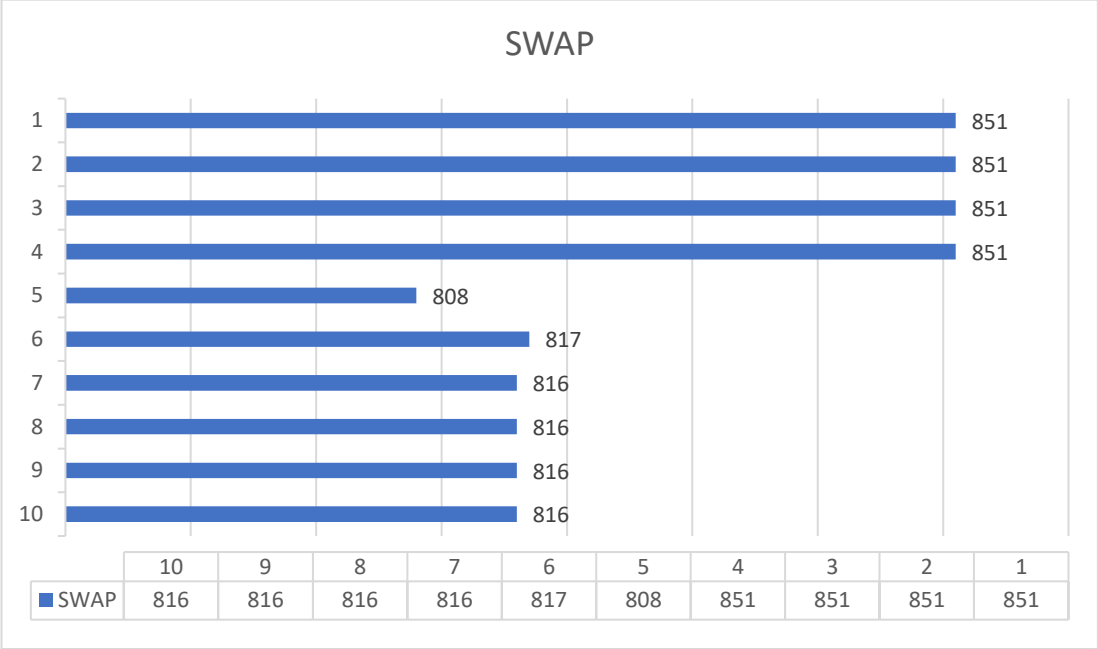
Anexo 3.16 ***Gráfico porcentaje CPU de htop en AlmaLinux.***

Con base al anexo 3.16 se realizó un promedio el cual dio como resultado que el rendimiento de CPU dentro de la máquina virtual AlmaLinux es de 85%.



Anexo 3.17 ***Gráfico porcentaje Memoria de htop en AlmaLinux.***

Con base al anexo 3.17 se realizó un promedio el cual dio como resultado que el uso de memoria dentro de la máquina virtual AlmaLinux es de 2.57G.



Anexo 3.18 Gráfico porcentaje SWAP de htop en AlmaLinux.

Con base al anexo 3.18 se realizó un promedio el cual dio como resultado que el uso de memoria SWAP dentro de la máquina virtual AlmaLinux es de 829.3M.

Tabla 3.8 Comparación Htop Docker y máquina virtual.

HTOP		
	DOCKER	**MÁQUINA VIRTUAL**
CPU	65.22%.	85%
MEMORIA	836.1M	2.57G
SWAP	652.9M	829.3M

Elaborado por: El Investigador

Con base a la tabla 3.8 se puede notar claramente que en cuanto a uso de cpu, memoria y swap Docker le lleva ventaja a la máquina virtual con una diferencia del 20% demostrando que Docker consume menos recursos.

- **Stacer**

Se instaló Stacer tanto en Docker como en la máquina virtual para verificar el rendimiento de cpu, memoria, disco, descarga y subida de archivos. La figura 3.33 muestra la ventana principal del stacker en el contenedor Docker.

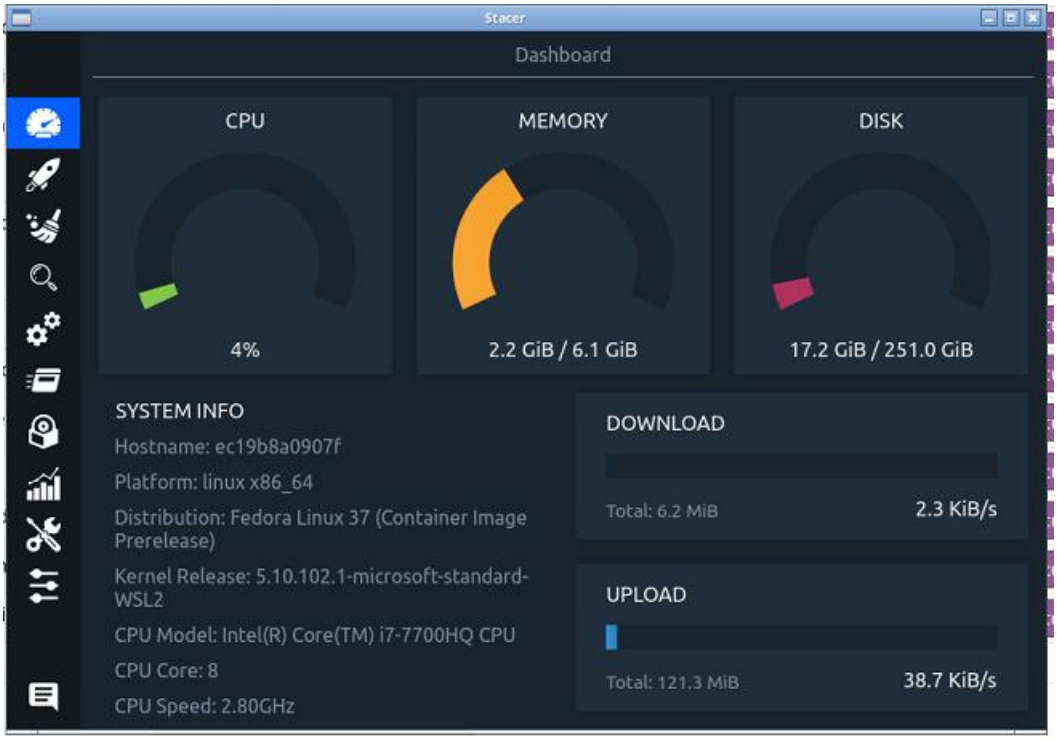


Figura 3.33 Stacer principal en Docker.

La figura 3.34 muestra la ventana de procesos de Stacer ejecutada en el contenedor Docker.

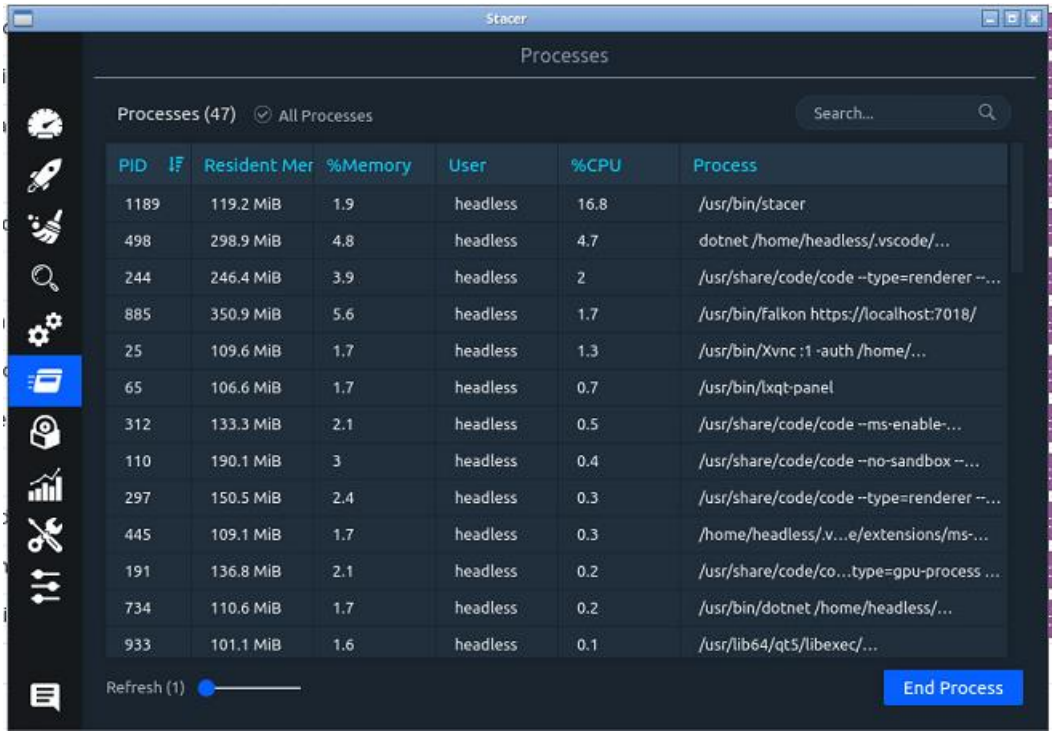
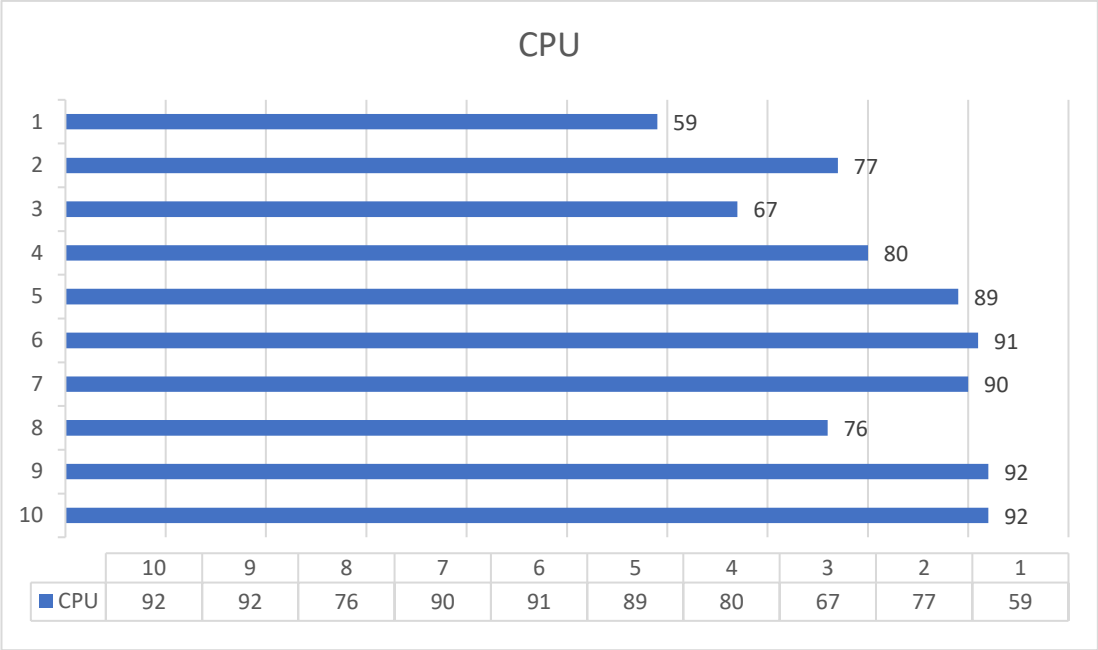


Figura 3.34 Stacer procesos en Docker.

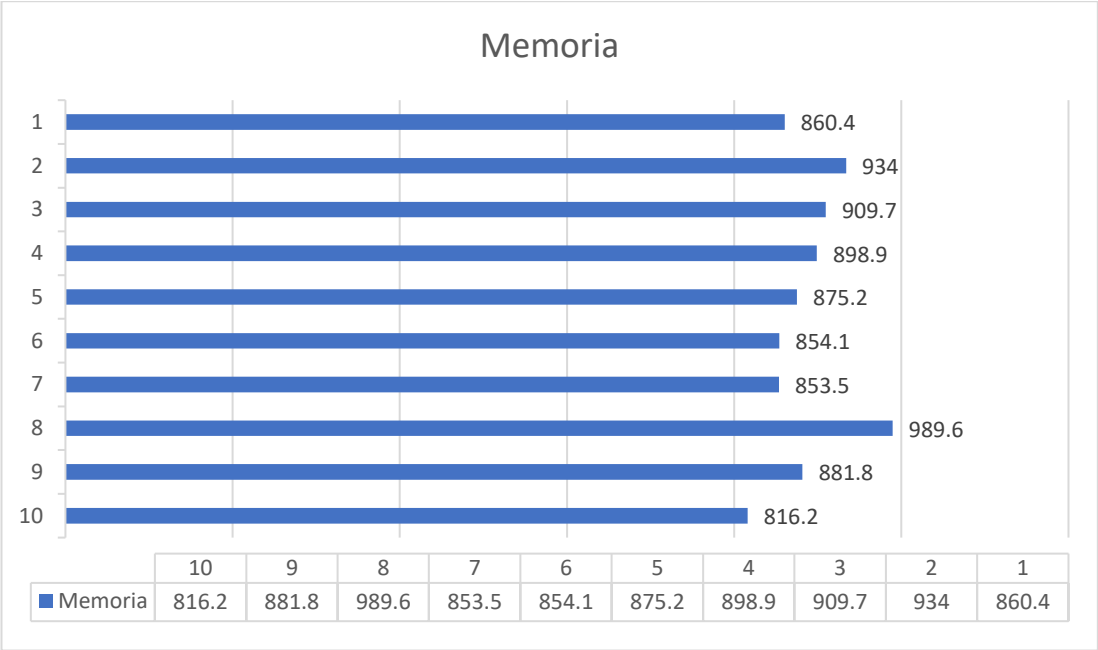
Las figuras 3.33 y 3.34 son un ejemplo de cómo se monitoreo con stacer el contenedor Docker y se obtuvo los siguientes datos: el uso de cpu es de 4%, el uso de memoria es de 2.2 GiB de 6.1 GiB que posee el contenedor, el uso de disco es de 17.2 GiB de 251.0 GiB. En descarga de archivos tiene 2.3 KiB/s y en subida 38.7 KiB/s. Con respecto al uso de Visual Studio Code el uso de cpu es de 4.7 y de memoria es de 4.8, el uso de memoria residente es de 298.9MiB y con respecto a la ejecución del programa en localhost el uso de cpu es de 1.7, el uso de memoria es de 5.6 y el uso de memoria residente es de 350.9MiB.

A partir del análisis hecho anteriormente se monitoreo 10 veces el rendimiento del contenedor Docker se obtuvieron los siguientes datos:



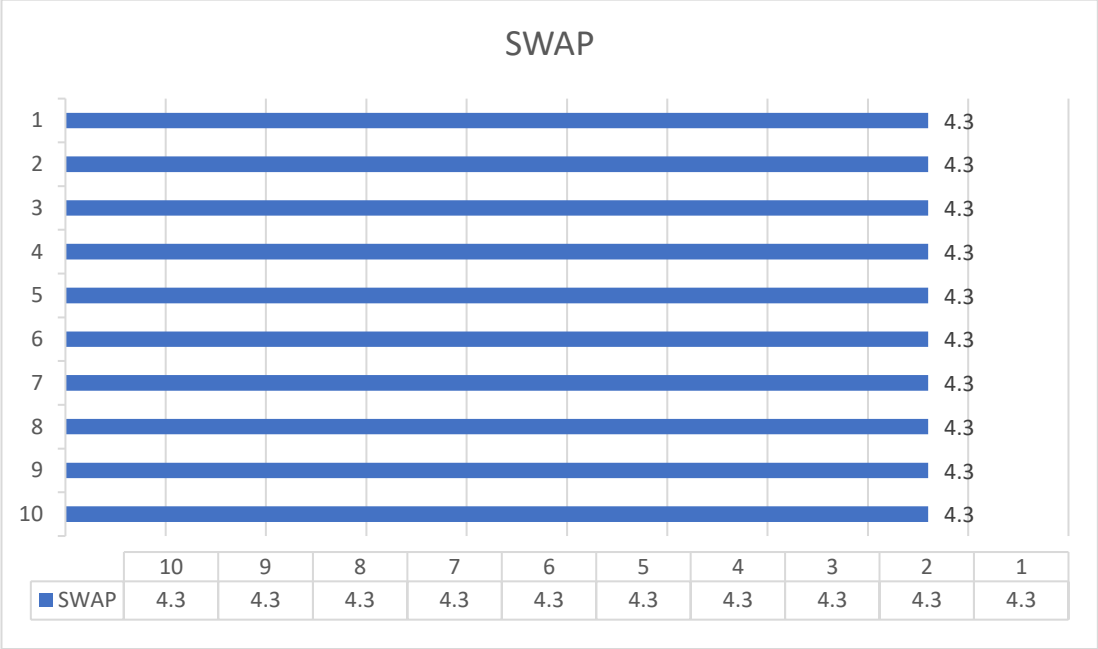
Anexo 3.19 Gráfico porcentaje CPU de stacer en Docker.

Con base al anexo 3.19 se realizó un promedio el cual dio como resultado que el rendimiento de CPU dentro de un contendor Docker es de 81.3%.



Anexo 3.20 Gráfico porcentaje Memoria de stacer en Docker.

Con base al anexo 3.20 se realizó un promedio el cual dio como resultado que el uso de memoria dentro de un contendor Docker es de 887.34M.



Anexo 3.21 Gráfico porcentaje SWAP de stacer en Docker.

Con base al anexo 3.21 se realizó un promedio el cual dio como resultado que el uso de memoria SWAP dentro de un contendor Docker es de 4.3G.

La figura 3.35 muestra la ventana principal del stacker en la máquina virtual AlmaLinux.

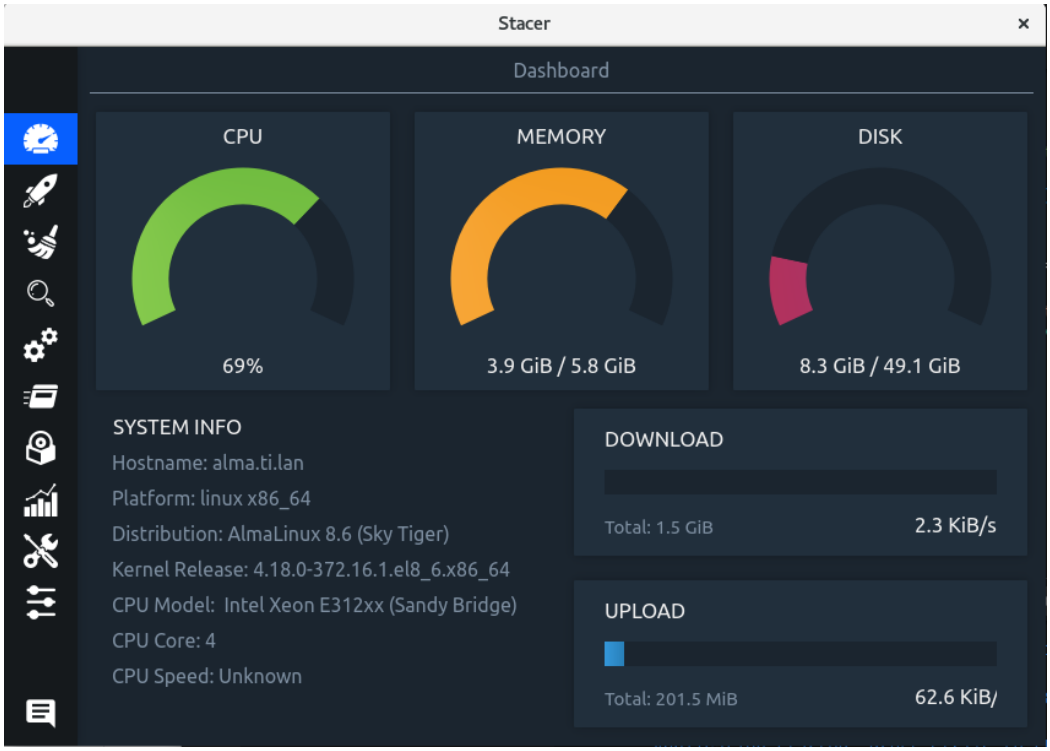


Figura 3.35 Stacer principal en AlmaLinux.

La figura 3.36 muestra la ventana de procesos de Stacer ejecutada en la máquina virtual AlmaLinux.

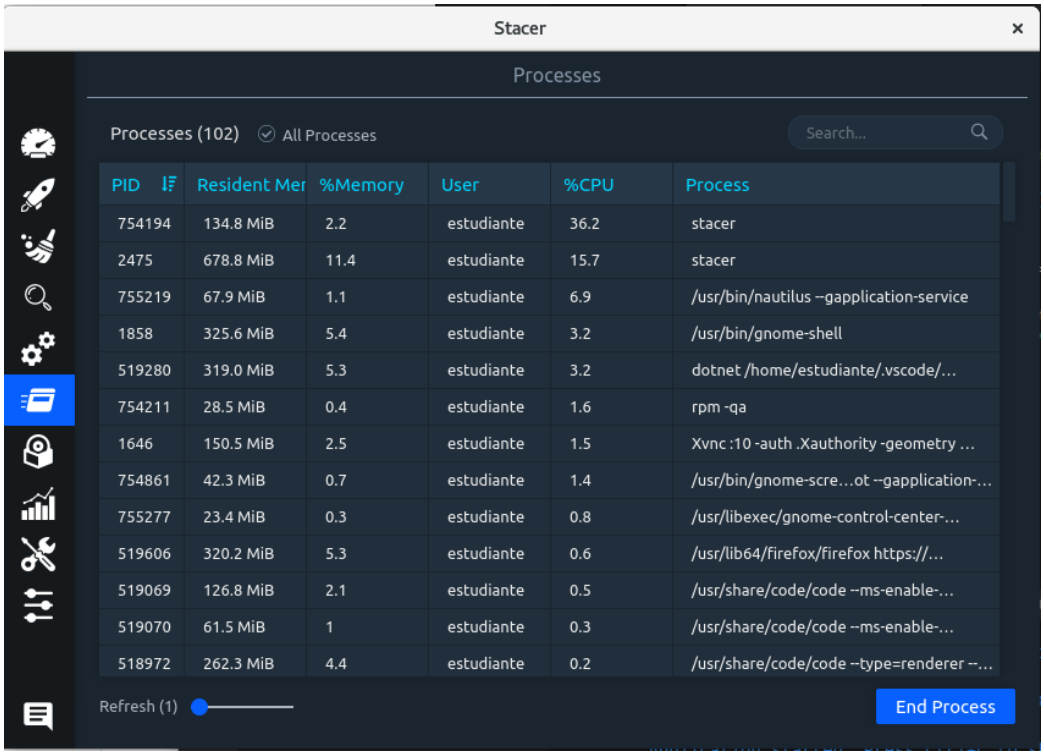
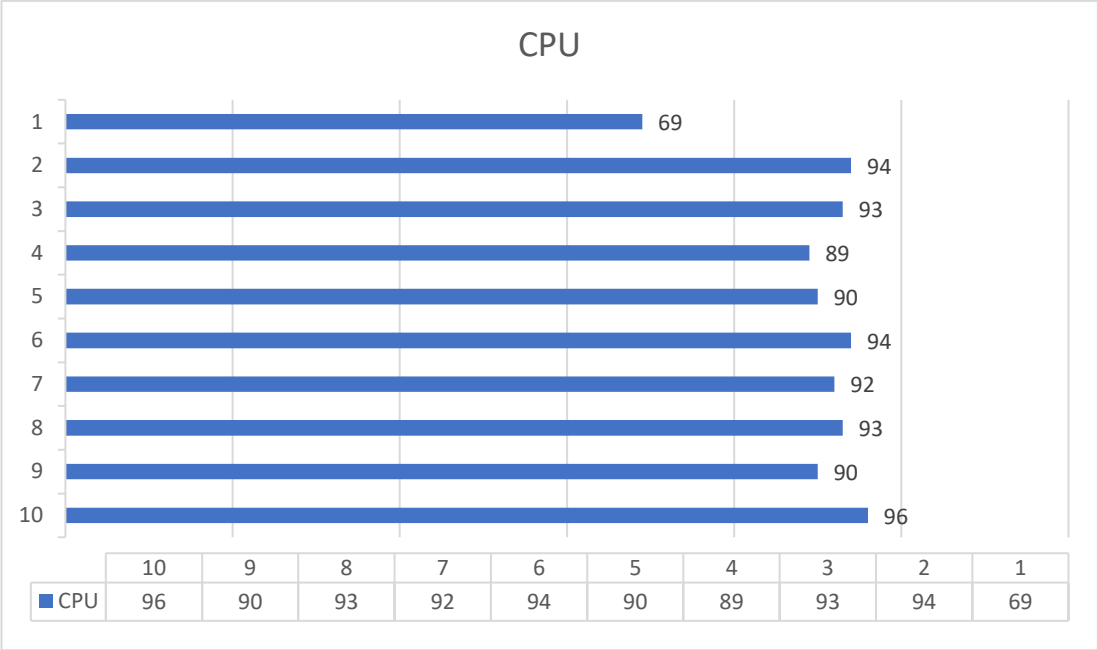


Figura 3.36 Stacer procesos en AlmaLinux.

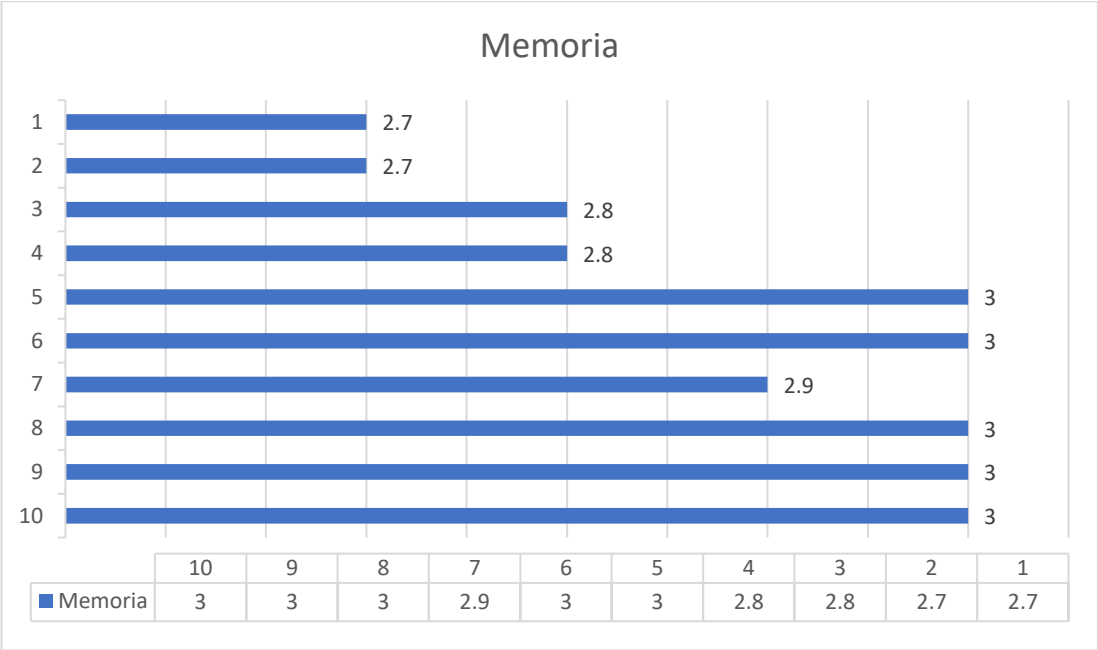
Las figuras 3.35 y 3.36 son un ejemplo de cómo se monitoreo con htop en una máquina virtual y se obtuvo los siguientes datos: el uso de cpu es de 69%, el uso de memoria es de 3.9 GiB de 5.8 GiB que posee la máquina virtual, el uso de disco es de 8.3 GiB de 49.1 GiB. En descarga de archivos tiene 2.3 KiB/s y en subida 62.6 KiB/s. Con respecto al uso de Visual Studio Code el uso de cpu es de 3.2 y de memoria es de 5.3, el uso de memoria residente es de 319.0MiB y con respecto a la ejecución del programa en localhost el uso de cpu es de 0.6, el uso de memoria es de 5.3 y el uso de memoria residente es de 320.2MiB.

A partir del análisis hecho anteriormente se monitoreo 10 veces el rendimiento de la máquina virtual AlmaLinux y se obtuvieron los siguientes datos:



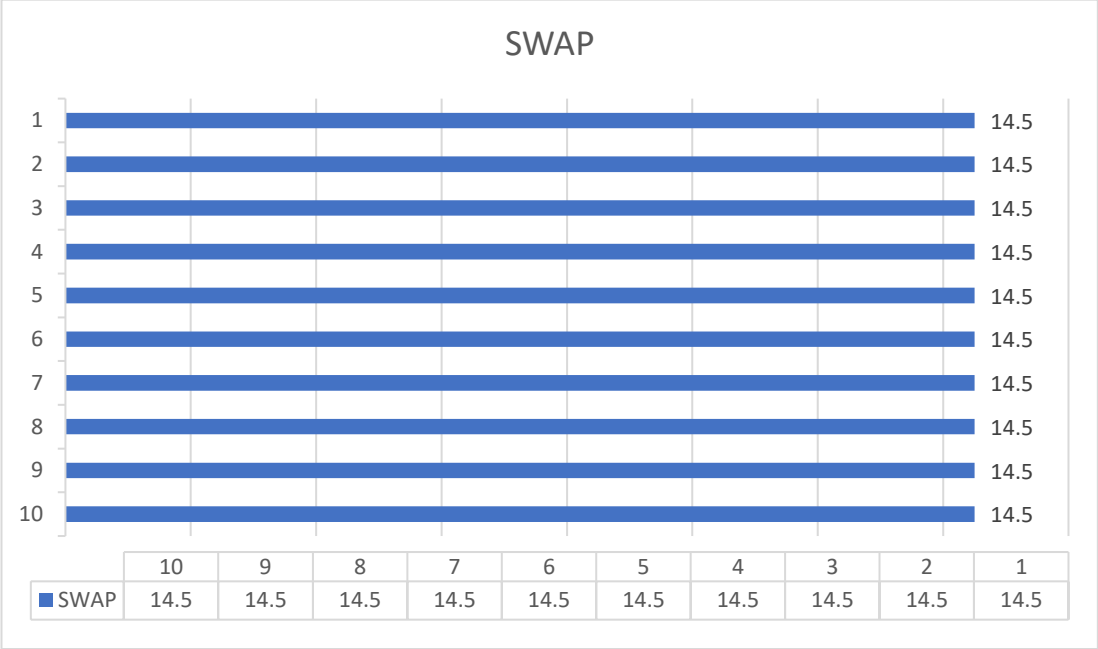
Anexo 3.22 Gráfico porcentaje CPU de stacer en AlmaLinux.

Con base al anexo 3.22 se realizó un promedio el cual dio como resultado que el rendimiento de CPU dentro de la máquina virtual AlmaLinux es de 90%.



Anexo 3.23 Gráfico porcentaje Memoria de stacer en AlmaLinux.

Con base al anexo 3.23 se realizó un promedio el cual dio como resultado que el uso de memoria dentro de la máquina virtual AlmaLinux es de 2.89G.



Anexo 3.24 Gráfico porcentaje SWAP de stacer en AlmaLinux.

Con base al anexo 3.24 se realizó un promedio el cual dio como resultado que el uso de memoria SWAP dentro de la máquina virtual AlmaLinux es de 14.5G.

Tabla 3.9 Comparación Stacer Docker y máquina virtual.

STACER		
	DOCKER	**MÁQUINA VIRTUAL**
CPU	81.3%	90%
MEMORIA	887.34M	2.89G
DISCO	4.3G	14.5G

Elaborado por: El investigador

Con base a la tabla 3.9 se puede evidenciar que en cuanto al uso de cpu, memoria y swap Docker sigue siendo liderando en poco uso de recursos antes que la máquina virtual.

3.2.6 Benchmarking

El tipo de Benchmarking que se realizó fue el comparativo ya que se realizó una comparación profunda entre máquina virtual y Docker para determinar cuál es el mejor para realizar virtualización.

- Planificación

Se planificó medir los tiempos de arranque, uso de memoria, uso de CPU y disco de cada uno de los elementos a comparar, después se planifico medir el tiempo de respuesta de la aplicación a testear. Para esto se utilizaron 3 herramientas de monitoreo.

- Datos

Los datos obtenidos en promedio con base a las herramientas usadas y a la ejecución de la aplicación fueron los mostrados en la tabla 3.10:

Tabla 3.10 Datos Benchmarking rendimiento Docker y máquina virtual

	DOCKER	**MÁQUINA VIRTUAL**
USO DE CPU PROMEDIO	66.63%	88.623%
USO DE MEMORIA (STACER Y HTOP)	861.72M	2.73G
USO DE DISCO (STACER Y HTOP)	2.469G	7.655G
TIEMPO DE ARRANQUE DE LA APLICACIÓN	1 minuto 23 seg 40 milisegundos	1 minuto 27 seg 60 milisegundos
AGREGAR LIBRO	28 seg 43 milisegundos	35 seg 86 milisegundos
ACTUALIZAR LIBRO	7 seg	7 seg 19 milisegundos
BORRAR LIBRO	4 seg 92 milisegundos	4 seg 32 milisegundos

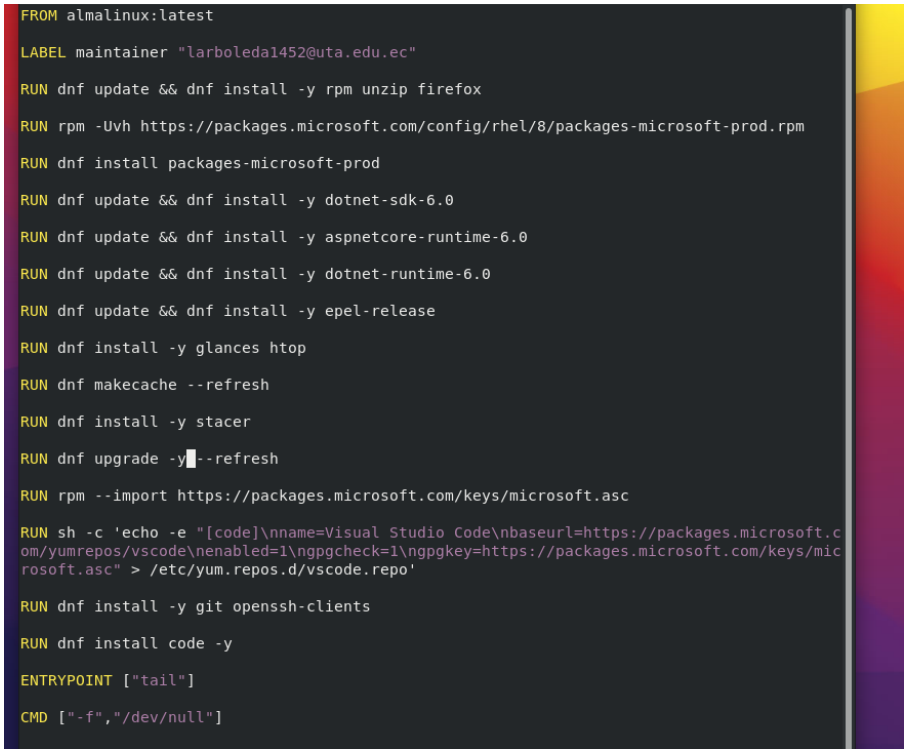
Elaborado por: El investigador

- Análisis

Con base a los datos obtenidos en el benchmarking se puede determinar que un contenedor ayuda a usar menos cpu, memoria y disco debido a que no usa directamente estos recursos de la máquina en la que esta iniciada, esta ventaja ayuda a que cualquier aplicación que se corra dentro de él va a ser más rápida y va a consumir menos recursos de la máquina host, sin embargo depende el tipo de aplicación que se quiera correr, en este caso resulto que el CRUD corrió más rápido en el contenedor pero de acuerdo a las pruebas individuales de procesos en la máquina virtual uso menos recursos por lo que se concluye que depende del tipo de aplicación que se desea correr se debe buscar las especificaciones adecuadas para que el contenedor funcione de la mejor manera.

- Acción

La acción que se realizó fue crear un contenedor con las especificaciones necesarias para que use menos recursos de memoria, cpu y disco, pero también que los procesos individualmente no consuman tanto. Para esto se creó un Dockerfile (Figura3.37) en el cual se inició un AlmaLinux ya que se observó que la máquina virtual AlmaLinux consumía menos recursos en procesos individuales, una vez iniciado el AlmaLinux se procede a actualizarlo e instalar las dependencias necesarias para ejecutar una aplicación C# (dotnet, aspnetCore y Visual Studio Code), aplicaciones necesarias para testear el rendimiento del contenedor (htop, glances y stacer), y un navegador. Este nuevo contenedor tiene un sistema operativo Almalinux 8.6, 4 procesadores, 6G de memoria y 11G de disco.



```
FROM almalinux:latest

LABEL maintainer "larboleda1452@uta.edu.ec"

RUN dnf update && dnf install -y rpm unzip firefox

RUN rpm -Uvh https://packages.microsoft.com/config/rhel/8/packages-microsoft-prod.rpm

RUN dnf install packages-microsoft-prod

RUN dnf update && dnf install -y dotnet-sdk-6.0

RUN dnf update && dnf install -y aspnetcore-runtime-6.0

RUN dnf update && dnf install -y dotnet-runtime-6.0

RUN dnf update && dnf install -y epel-release

RUN dnf install -y glances htop

RUN dnf makecache --refresh

RUN dnf install -y stacer

RUN dnf upgrade -y --refresh

RUN rpm --import https://packages.microsoft.com/keys/microsoft.asc

RUN sh -c 'echo -e "[code]\nname=Visual Studio Code\nbaseurl=https://packages.microsoft.com/yumrepos/vscode\nenabled=1\ngpgcheck=1\ngpgkey=https://packages.microsoft.com/keys/microsoft.asc" > /etc/yum.repos.d/vscode.repo'

RUN dnf install -y git openssh-clients

RUN dnf install code -y

ENTRYPOINT ["tail"]

CMD ["-f","/dev/null"]
```

Figura 3.37 Código Dockerfile.

Para que el nuevo contenedor funcione correctamente se creó un docker-compose.yml (Figura 3.38) que brinda la vinculación de las dependencias necesarias para que funcionen de forma correcta las aplicaciones que se instalaron en el Dockerfile. Se especifica la versión, el servicio, el entorno, puerto y volumen.

```
version: '6.0'
services:
        dotnet:
                build:
                        context: ./
                        #args:
                        # - DISPLAY: ${DISPLAY}
                environment:
                 - DISPLAY=${DISPLAY}
                container_name: contenedor_almalinux
                ports:
                  - 5001:5001
                volumes:
                  #Compartir interfaz grafica
                  - /tmp/.X11-unix/:/tmp/.X11-unix
```

Figura 3.38 Código docker-compose.yml.

Una vez creados los dos archivos se procedió a montar el contenedor para empezar a usarlo y testear la aplicación para medir sus tiempos. El tiempo que tardo en instalarse debido a las instalaciones que debe hacer fue de 25 min.

Dentro del contenedor se procedió a abrir Visual Studio Code e instalar el paquete .NET watch attach por las dependencias que tiene la aplicación que se ha testeado en los anteriores contenedores. Se descargó la aplicación CRUD y se procedió a ejecutarla, el tiempo que tardo en ejecutarse fue de 1 minuto con 10 seg como se observa en la figura 3.39.

Figura 3.39 Tiempo inicio programa en nuevo contenedor.

La figura 3.40 muestra la aplicación ejecutándose en el nuevo contenedor.

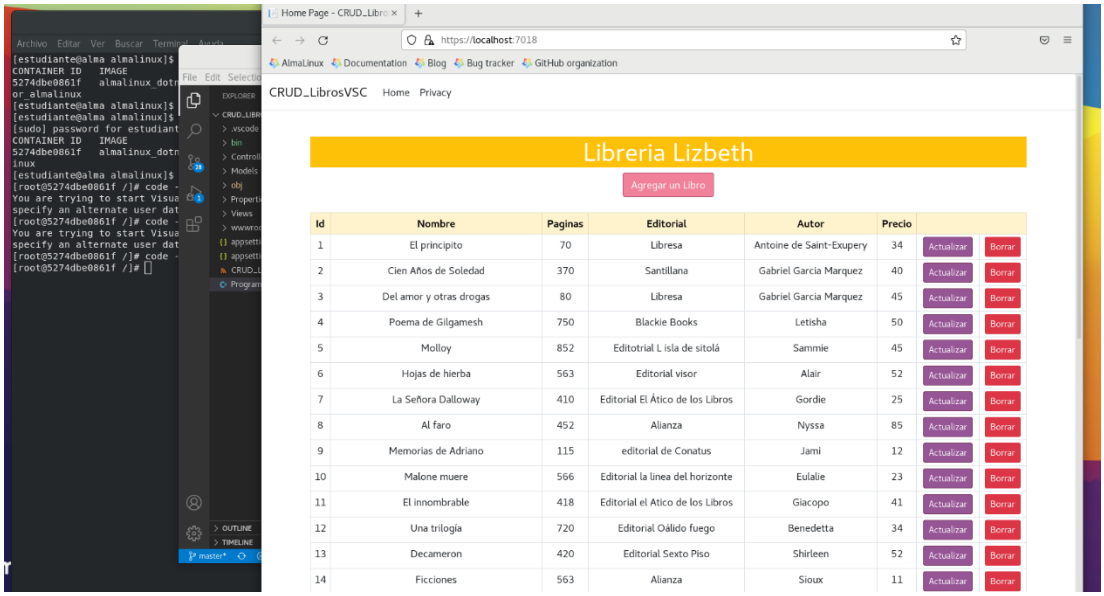


Figura 3.40 Inicio programa en nuevo contenedor.

Se ejecuto toda la aplicación CRUD en el nuevo contenedor, es decir, la aplicación de la librería virtual se ejecutó y se se visualizó, agregó, actualizó. y eliminó un libro.

El tiempo que tardó en agregar un libro fue de 33 seg como se observa en la figura 3.41.

Figura 3.41 Tiempo agregar libro en el programa en el nuevo contenedor.

La figura 3.42 muestra la pantalla de agregar libro dentro del nuevo contenedor.

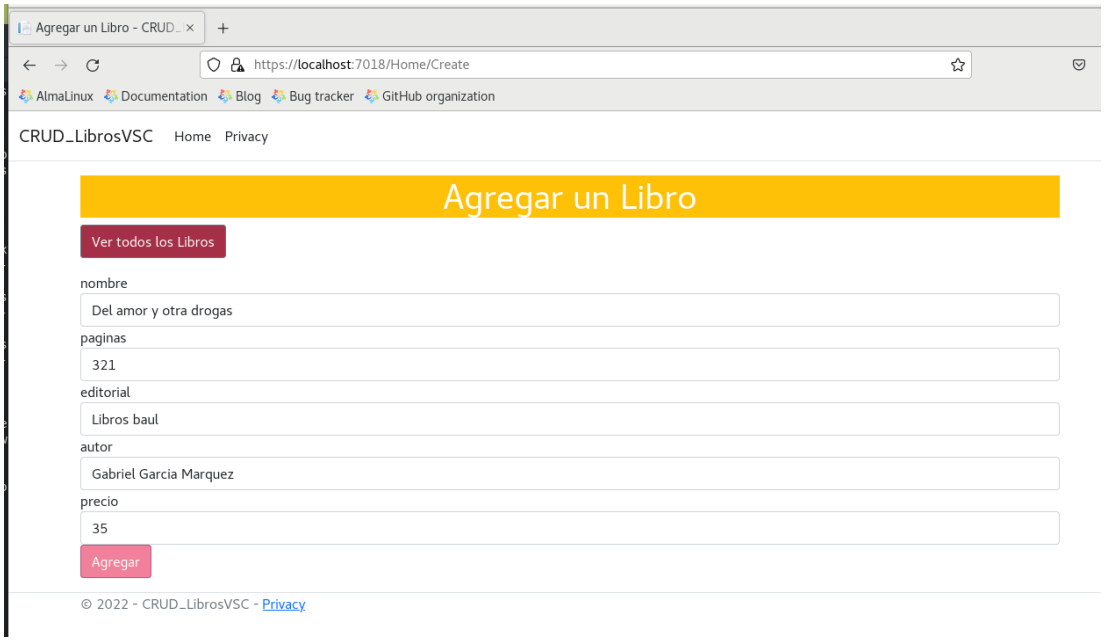


Figura 3.42 Agregar libro en el programa en el nuevo contenedor.

La figura 3.43 muestra el libro agregado en la ejecución de la aplicación en el nuevo contenedor.

29	Tiempos de tormentas	255	Hoja de lata	Boris Iszaguirre	15	Actualizar	Borrar
30	La llamada de la tribu	178	Calambur	Charles Dickens	40	Actualizar	Borrar
91	Del amor y otra drogas	321	Libros baul	Gabriel Garcia Marquez	35	Actualizar	Borrar

Figura 3.43 Libro Agregado en el programa en el nuevo contenedor.

El tiempo que tardó en actualizar un libro fue de 11 seg como lo muestra la figura 3.44.

Figura 3.44 Tiempo en actualizar un libro en el programa en el nuevo contenedor.

La figura 3.45 muestra la pantalla de agregar libro dentro del nuevo contenedor.

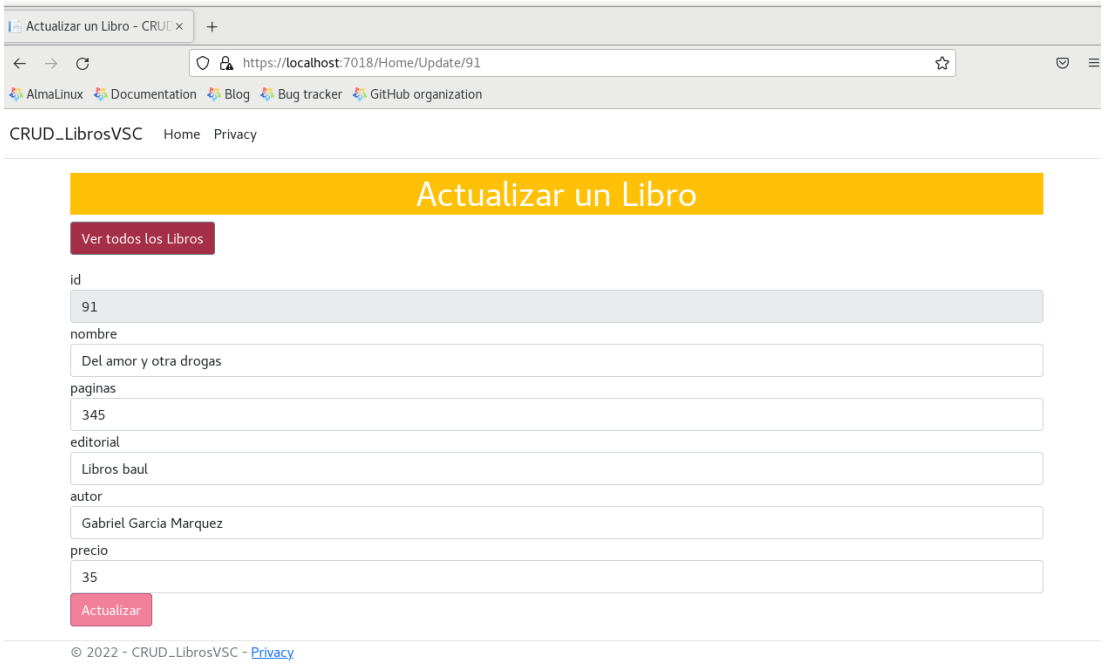


Figura 3.45 Actualizar libro en el programa en el nuevo contenedor.

La figura 3.46 muestra el libro actualizado en la ejecución de la aplicación en el nuevo contenedor.

28	Las almas de Brandon	147	Hoja de lata	Cesar Brandon	16	Actualizar	Borrar
29	Tiempos de tormentas	255	Hoja de lata	Boris Iszaguirre	15	Actualizar	Borrar
30	La llamada de la tribu	178	Calambur	Charles Dickens	40	Actualizar	Borrar
91	Del amor y otra drogas	345	Libros baul	Gabriel Garcia Marquez	35	Actualizar	Borrar

Figura 3.46 Libro Actualizado en el programa en el nuevo contenedor.

El tiempo que tardó en borrar un libro fue de 3 seg como muestra la figura 3.47.

Figura 3.47 Tiempo en borrar un libro en el programa en el nuevo contenedor.

La figura 3.48 muestra el libro borrado en la ejecución de la aplicación en el nuevo contenedor.

29	Tiempos de tormentas	255	Hoja de lata	Boris Iszaguirre	15	Actualizar	Borrar
30	La llamada de la tribu	178	Calambur	Charles Dickens	40	Actualizar	Borrar

Figura 3.48 Libro borrado en el programa en el nuevo contenedor.

Se ejecutaron las aplicaciones para medir el rendimiento dando los siguientes resultados:

En htop (Figura3.49) el porcentaje de uso de cpu es de 41.9% dividido en 4 procesos, el uso de memoria es de 2.52 G de 5.80 G y el uso de SWAP es de 622M de 5.93 G.

En cuanto a la aplicación testeada en Visual Studio Code el uso de cpu es de 5.1, el uso de memoria es de 3.3, el uso de memoria virtual es de 4158M y el uso de memoria residente es de 192M, en cuanto a la aplicación en localhost el uso de cpu es de 1.3, el uso de memoria es de 4.9, el uso de memoria virtual es de 3014M y el uso de memoria residente es de 293M.

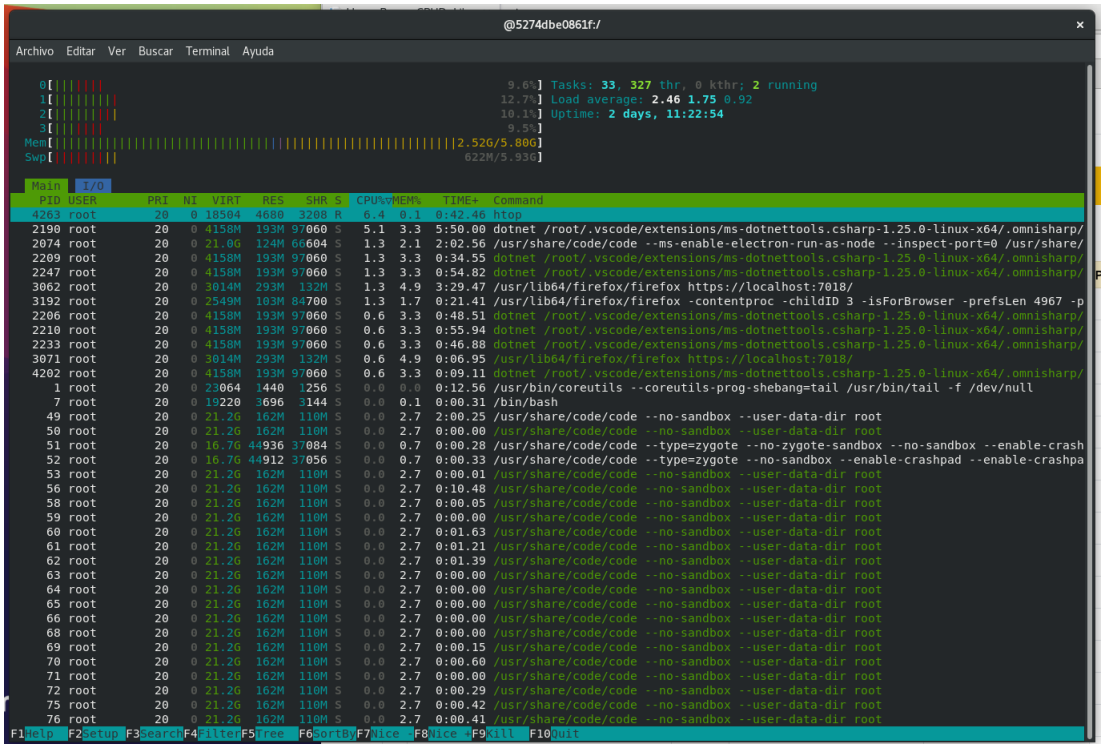


Figura 3.49 Htop en el nuevo contenedor.

Según Glances (Figura 3.50) el uso de cpu es de 10.2% el cual se divide en 6.3% en usuario y 1.1% en sistema. En cuanto a memoria muestra que está en uso un 48.4% dividido en activo 1.80G y que en total en todas las instalaciones previas se ha usado 2.81GB de 5.80 GB. En SWAP está en uso 12.2% especificado que se usa 741M de 5.93GB. En cuanto a la aplicación en Visual Studio Code se puede observar que ocupa el 5.1 de cpu, 3.0 de memoria, 4.07G de memoria virtual, 180M de memoria residente y en cuanto a la aplicación ejecutada en localhost se puede observar que ocupa el 0.5 de cpu, 5.0 de memoria, 2.92G de memoria virtual, 298M de memoria residente.

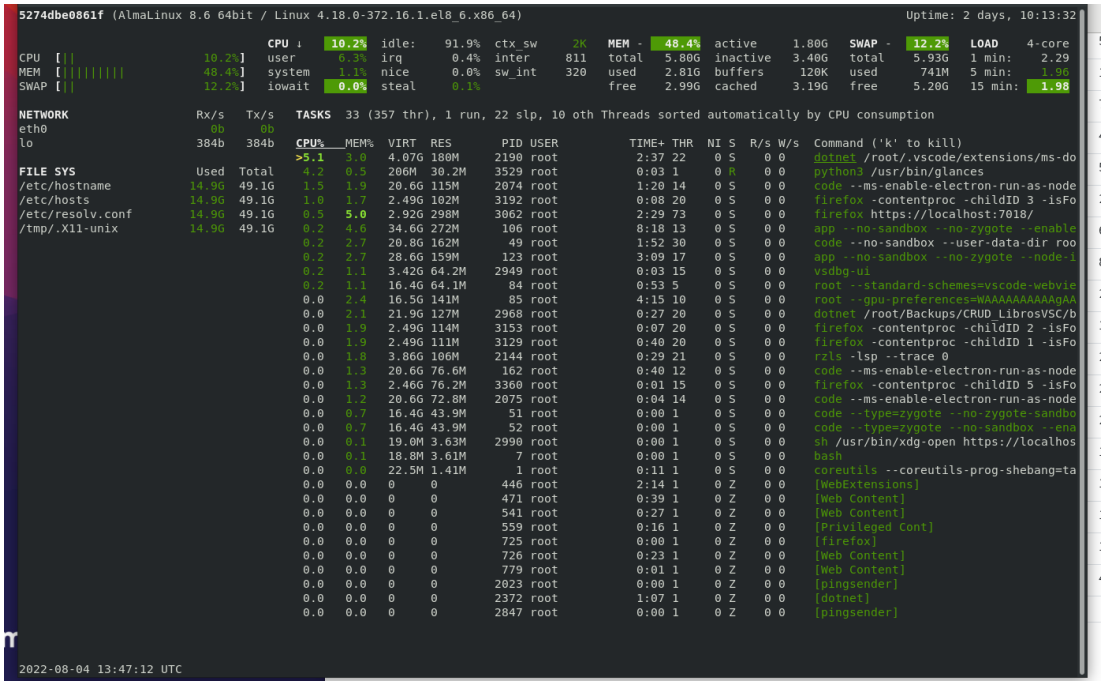


Figura 3.50 Glances en el nuevo contenedor.

Según Stacer (Figuras 3.51 y 3.52) el uso de cpu es de 8%, el uso de memoria es de 2.7 GiB de 5.8 GiB que posee el contenedor, el uso de disco es de 14.9 GiB de 49.1 GiB. En descarga de archivos tiene 0 bytes/s de un total de 200.7 MiB y en subida 0 bytes/s de un total de 2.7 MiB. Con respecto al uso de Visual Studio Code el uso de cpu es de 11.8 y de memoria es de 3, el uso de memoria residente es de 179.6 MiB y con respecto a la ejecución del programa en localhost el uso de cpu es de 13.4, el uso de memoria es de 5 y el uso de memoria residente es de 299.9 MiB.

La figura 3.51 muestra la ventana principal del stacker en el nuevo contenedor.

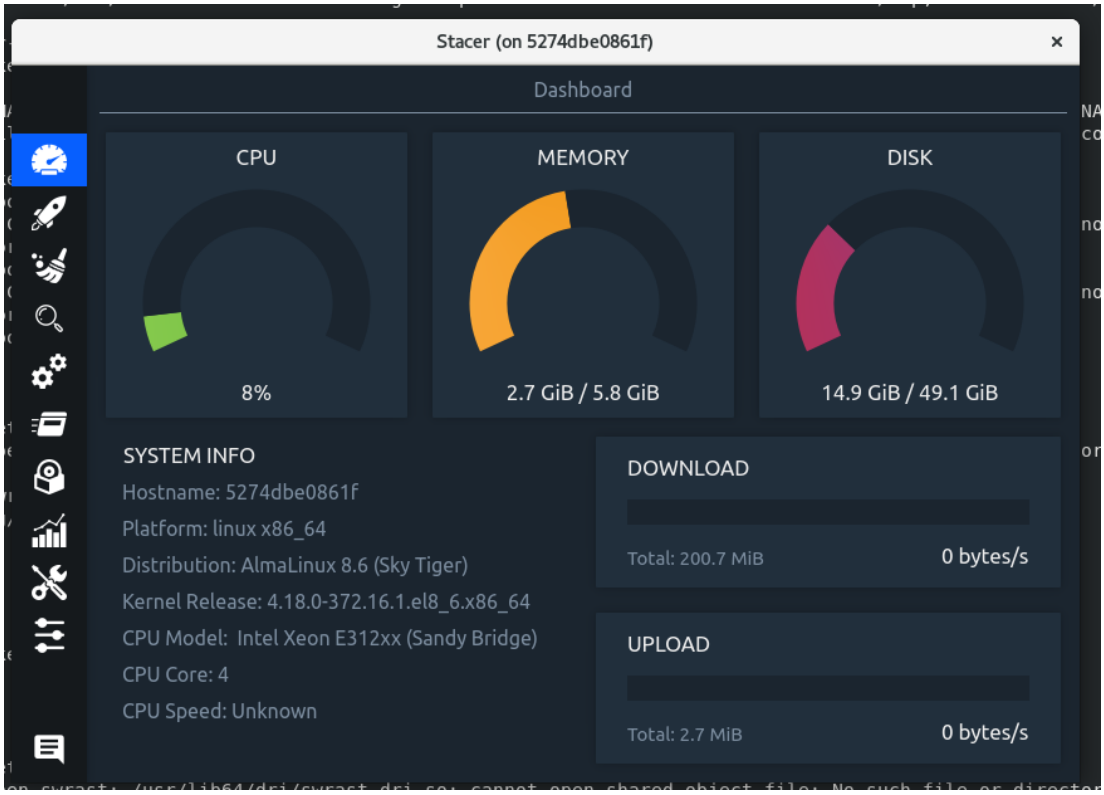


Figura 3.51 Stacer principal en el nuevo contenedor.

La figura 3.52 muestra la ventana de procesos de Stacer ejecutada en el nuevo contenedor.

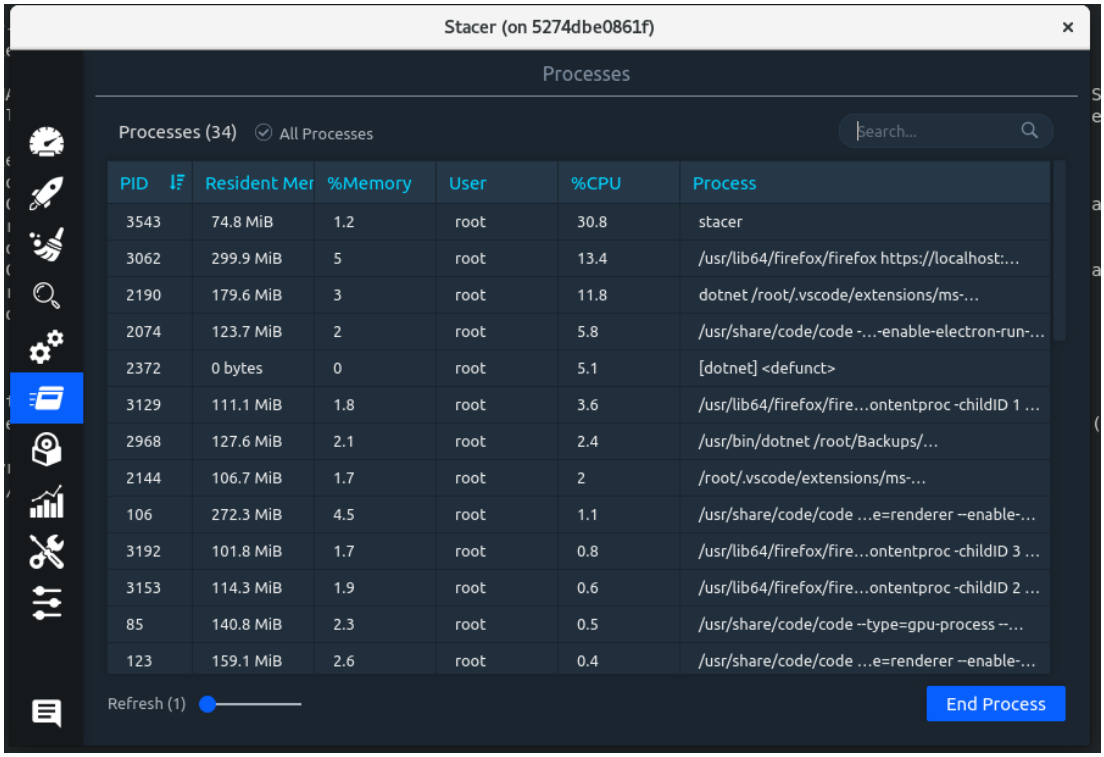


Figura 3.52 Stacer procesos en el nuevo contenedor.

La tabla 3.11 muestra un resumen de los datos obtenidos en el nuevo contenedor.

Tabla 3.11 Datos finales nuevo contenedor.

	NUEVO CONTENEDOR
USO DE CPU	20.03%
USO DE DISCO	16.26 G
USO DE MEMORIA	3.32G
TIEMPO DE ARRANQUE DE LA APLICACIÓN	1 minuto 10 seg 8 milisegundos
AGREGAR LIBRO	33 seg 75 milisegundos
ACTUALIZAR LIBRO	11 seg 81 milisegundos
BORRAR LIBRO	3 seg 49 milisegundos

Elaborado por: El Investigador

A continuación, se muestra un cuadro resumen con las características y los resultados obtenidos en el contendor Docker, máquina virtual AlmaLinux y el nuevo contenedor.

Tabla 3.12 Cuadro Resumen de Docker, máquina virtual y nuevo contenedor.

	DOCKER	**MÁQUINA VIRTUAL**	**NUEVO CONTENEDOR**
SO	Fedora Linux	AlmaLinux 8.6	AlmaLinux 8.6
PROCESADORES	1	4	4
MEMORIA	1.5G	6G	6G
DISCO	28G	87G	11G
USO DE CPU	66.63%	88.623%	20.03%
USO DE DISCO	861.72M	2.73G	16.26 G
USO DE MEMORIA	2.469G	7.655G	3.32G
TIEMPO ARRANQUE APLICACIÓN	1 minuto 23 seg 40 milisegundos	1 minuto 27 seg 60 milisegundos	1 minuto 10 seg 8 milisegundos
AGREGAR LIBRO	28 seg 43 milisegundos	35 seg 86 milisegundos	33 seg 75 milisegundos
ACTUALIZAR LIBRO	7 seg	7 seg 19 milisegundos	11 seg 81 milisegundos
BORRAR LIBRO	4 seg 92 milisegundos	4 seg 32 milisegundos	3 seg 49 milisegundos

Elaborado por: El Investigador

La comparación del cuadro resumen de la figura 3.12 dio como resultado que el nuevo contenedor creado con todas las dependencias necesarias para ejecutar aplicaciones C# utiliza menos cpu, disco y memoria, al momento de ejecutar la aplicación también la ejecuta en menor tiempo y con fluidez comprobando que la creación de este nuevo contenedor fue todo un éxito.

- Seguimiento

Para el seguimiento del contenedor se lo subió a Docker Hub en donde las personas lo pueden descargar y ejecutar. La figura 3.53 muestra la imagen en Docker Hub.

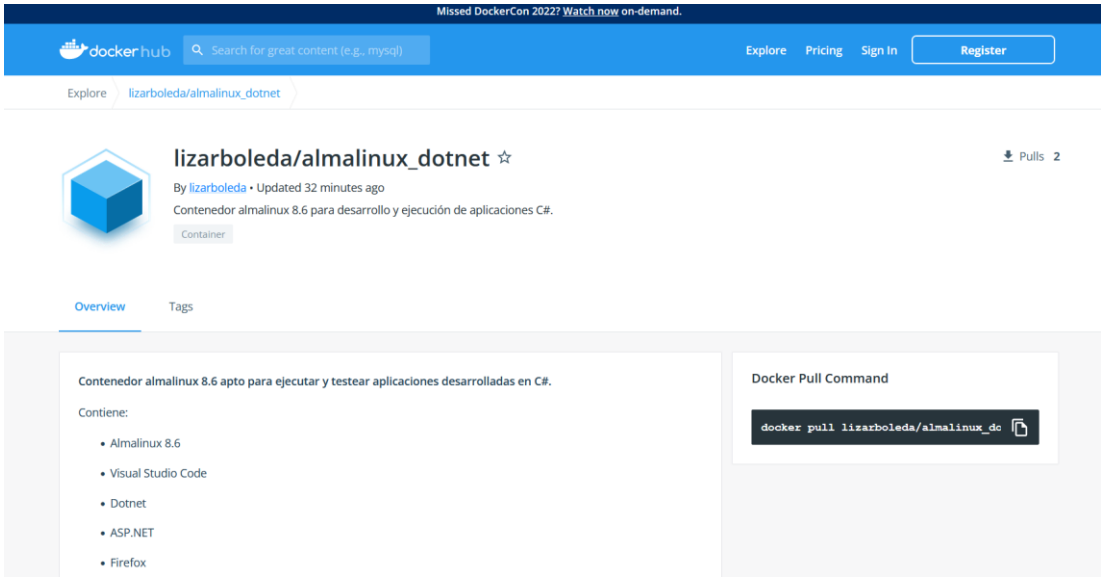


Figura 3.53 Imagen publicada en Docker Hub.

CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- Se ha logrado configurar correctamente una aplicación en C# al igual que un contenedor Docker y una máquina virtual en la cual se puede exponer que gracias a las características propias de los contenedores el rendimiento del mismo es mucho más flexible y no consume muchos recursos al momento de correr la aplicación creada.
- Para el uso de Docker antes se debe preparar la máquina en la que se va a instalar y siempre es preferible tener una aplicación que ayude a administrar los contenedores de manera segura, la más conocida es Docker Desktop.
- Para ejecutar una aplicación C# no importa el sistema operativo que se use siempre se debe instalar el framework .Net ya que este ayuda a que no exista problemas entre versiones y generando compatibilidad para que la aplicación se ejecute en cualquier sistema.
- Gracias a Benchmarking se logró determinar que la mejor manera de virtualización, con base a testeo de aplicaciones desarrolladas en C#, ya que utiliza menos recursos, el tiempo de arranque es mucho menor al igual que la velocidad, y es más fácil y rápido de usar e instalar es el contenedor Docker debido a que contiene un sistema operativo completo pero no utiliza el sistema operativo de la máquina en la que se está ejecutando por lo que optimiza recursos y funciona de manera más flexible.

4.2 Recomendaciones

- Se sugiere dar mayor seguimiento al uso de contenedores Docker en testeo de aplicaciones y en actividades académicas debido a que existe poca documentación e información sobre contenedores Docker y sus beneficios.
- Se recomienda el uso de contenedores Docker al momento de ejecutar tareas específicas ya que consume menos recursos y las ejecuta en menor tiempo.
- Se recomienda usar Benchmarking comparativo para realizar comparaciones entre dos objetos similares ya que brinda un proceso guiado en el cual se puede sacar conclusiones y mejoraras para los dos objetos comparados.

BIBLIOGRAFÍA

[1] M. I. Romero *et al.*, "INTRODUCCIÓN A LA SEGURIDAD INFORMÁTICA Y EL ANÁLISIS DE VULNERABILIDADES."

[2] M. Toapanta and R. Maciel, "Las TICS en el Ecuador," Aug. 2015, [Online]. Available: https://www.researchgate.net/publication/309666080

[3] E. Villar and J. Gómez, "Introducción a la Virtualización," 2011, Accessed: Feb. 01, 2022. [Online]. Available: http://www.adminso.es

[4] Red Hat, "¿Qué es Docker?," Jan. 09, 2018. https://www.redhat.com/es/topics/containers/what-is-docker (accessed Feb. 01, 2022).

[5] A. García, J. Calvo, and O. Bravo, "Linux vs. Windows," Univerdidad Nacional de Colombia, Colombia, 2017. Accessed: Feb. 01, 2022. [Online]. Available: http://www.fce.unal.edu.co/media/files/UIFCE/Otros/Windows_Vs_Linux.pdf

[6] M. Castillo, "Seguridad informática, evaluación, comparación e implementación de equipos de seguridad," 2010, Accessed: Nov. 28, 2021. [Online]. Available: http://dspace.udla.edu.ec/bitstream/33000/3478/1/UDLA-EC-TTRT-2010-02%28S%29.pdf

[7] C. De, P. de Titulación, J. Luis, P. Laje, I. Pedro, and M. Romero, "MICROSERVICIOS Y CONTENEDORES DOCKERS Y UNA ARQUITECTURA TRADICIONAL (MONOLÍTICA) CON COMPROBACIÓN APLICATIVA," 2018.

[8] P. Ponsico, "Tecnología de Contenedores Docker," 2017, Accessed: Nov. 28, 2021. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/113040/Degree_thesis.pdf?sequence=1

[9] R. Cayetano and D. Alonso, "SOFTWARE CONTAINERIZATION WITH DOCKER," 2017.

[10] J.-A. Kabbe and C. A. Boyd, “Security analysis of Docker containers in a production environment,” 2017.

[11] “Serie: Cuadernos de Informática”, Accessed: Nov. 30, 2021. [Online]. Available: www.uigv.edu.pe

[12] C. Belloch, “Las Tecnologías de la Información y Comunicación en el aprendizaje,” 2002, Accessed: Nov. 30, 2021. [Online]. Available: http://www.clubcultura.com/clubliteratura/clubescritores/sampedro/miradas_global.htm

[13] Universidad Tecnologica Nacional, “VIRTUALIZACIÓN DE REDES LINUX”.

[14] Y. Fernández and García Karen, “Virtualización”.

[15] “informático, informática | Definición | Diccionario de la lengua española | RAE - ASALE.” https://dle.rae.es/inform%C3%A1tico (accessed Nov. 30, 2021).

[16] J. Bernardo, V. Gomez, and R. T. Milenio, “Arquitectura de computadoras I.”

[17] “What is a Container? | App Containerization | Docker.” https://www.docker.com/resources/what-container (accessed Nov. 14, 2021).

[18] J. Gallo, “Docker 101”.

[19] L. Chen, N. Li, K. Liang, and S. Schneider, “Computer Security-ESORICS 2020.” [Online]. Available: http://www.springer.com/series/7410

[20] D. Bell and M. Parr, *C# para Estudiantes*, vol. 1. Gran Bretaña: Pearson Education Limited, 2011. Accessed: Jan. 30, 2022. [Online]. Available: https://profesorezequielruizgarcia.files.wordpress.com/2013/08/c-sharp-para-estudiantes.pdf

[21] Microsoft, “A Tour of C# language,” *Microsoft*, Nov. 2021, Accessed: Jan. 30, 2022. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[22] Besoftware, “¿Qué es C# en programación y para que sirve?,” *Soluciones TIC*, 2020. https://bsw.es/que-es-c/ (accessed Jan. 30, 2022).

[23] “Docker Desktop overview | Docker Documentation.” https://docs.docker.com/desktop/ (accessed Jun. 13, 2022).

[24] “What is .NET? | Codecademy.” https://www.codecademy.com/article/what-is-net (accessed Jun. 13, 2022).

[25] “What is ASP.NET? | .NET.” https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet (accessed Jun. 13, 2022).

[26] “What is Visual Studio? - Incredibuild.” https://www.incredibuild.com/integrations/visual-studio (accessed Jun. 13, 2022).

[27] “Visual Studio: IDE y Editor de código para desarrolladores de software y Teams.” https://visualstudio.microsoft.com/es/ (accessed Jun. 13, 2022).

[28] “What is CRUD? | Codecademy.” https://www.codecademy.com/article/what-is-crud (accessed Jun. 13, 2022).

[29] “What is CRUD? Explaining CRUD Operations | Sumo Logic.” https://www.sumologic.com/glossary/crud/ (accessed Jun. 13, 2022).

[30] “Benchmarking Definition - What is Benchmarking.” https://www.shopify.in/encyclopedia/benchmarking (accessed Jun. 13, 2022).

[31] “What is Benchmarking? Technical & Competitive Benchmarking Process | ASQ.” https://asq.org/quality-resources/benchmarking (accessed Jun. 13, 2022).

[32] “Kanban Model and Process for Software Development: Principles, Steps, Methodology.” https://xbsoftware.com/blog/software-development-life-cycle-sdlc-all-about-kanban/ (accessed Jun. 13, 2022).

[33] “¿Qué Es Kanban? Una Descripción General Del Método Kanban.” https://www.digite.com/es/kanban/que-es-kanban/ (accessed Jun. 13, 2022).

[34] “Los mejores software de virtualización: Windows, Mac y Linux - IONOS,” Mar. 09, 2022. https://www.ionos.es/digitalguide/servidores/configuracion/comparativa-de-software-de-virtualizacion/ (accessed Jul. 24, 2022).

[35] “10 distribuciones Linux más usadas | EDteam.” https://ed.team/blog/10-distribuciones-linux-mas-usadas (accessed Jul. 24, 2022).

[36] M. Heyan, “7 Best Gnome system monitor alternative Linux applications,” Jan. 29, 2021. https://www.how2shout.com/linux/7-best-gnome-system-monitor-alternative-linux-applications/ (accessed Jul. 24, 2022).

[37] “Comandos frecuentes de Docker - Preguntas Frecuentes - FAQ - Cerebro Digital.” https://cerebro-digital.com/panel/knowledgebase/63/Comandos-frecuentes-de-Docker.html (accessed Jul. 24, 2022).

[38] M. Franco, “Virtualización con KVM y comandos virsh.,” May 01, 2018. https://manuelfrancoblog.wordpress.com/2018/05/01/virtualizacion-con-kvm-y-comandos-virsh/ (accessed Jul. 24, 2022).

ANEXOS

FICHA BIBLIOGRÁFICA	
TEMA	Docker Containers Versus Virtual Machine-Based Virtualization
TESIS	El autor propone que a los contenedores Docker se los puede conocer como una técnica de virtualización ligera. Brinda aspectos positivos de la virtualización para la integración de aplicaciones como lo es la escalabilidad y flexibilidad. Proporciona un concepto claro de virtualización, máquinas virtuales y contenedores Docker. Brinda los beneficios de más máquinas virtuales y sus limitaciones, funcionalidad de Docker y la necesidad de cambiarse al mismo. Se puede observar la diferencia entre máquinas virtuales y Dockers en todos los aspectos (compatibilidad, seguridad, arranque, portabilidad, etc.), lo muestra de forma detallada y en una tabla comparativa.
PROPÓSITO	Comparar máquinas virtuales con contenedores Docker para ver cual brinda más ventajas de acuerdo con la operabilidad en la nube.
IDEAS CENTRALES	• Docker es un concepto más liviano de máquinas virtuales. • Máquinas virtuales son mejores para hardware mientras que contenedores Docker para software.

	• Docker ayuda a la portabilidad e interoperabilidad usando técnicas de virtualización mejoradas. • Docker simplifica el trabajo de los desarrolladores de aplicaciones. • Docker mejora la seguridad y ayuda a aislar las dependencias de las aplicaciones. • Los contenedores generalmente se ejecutan en el sistema operativo del host y las • máquinas virtuales se ejecutan en el hipervisor.
CONCEPTOS CLAVES	Contiene los conceptos de máquina virtual y Docker.
CONCLUSIONES	Tanto Docker como máquinas virtuales se basan en la virtualización y resuelven problemas específicos. Si el requisito es crear un sistema seguro, entonces se deben preferir las máquinas virtuales. Mientras se trabaja en un entorno heterogéneo, los contenedores Docker se centran en las aplicaciones y las dependencias asociadas con ellas.
APORTE A TEMA ELEGIDO	Este paper ayuda a ver las claras diferencias que existen entre máquinas virtuales y contenedores en todos los aspectos posibles de hardware y software, de igual manera indica las ventajas de usar Docker con respecto a las aplicaciones lo cual ayudará a que el

	objetivo principal del tema elegido se compruebe.

Anexo 4.1 Ficha Bibliográfica 1

FICHA BIBLIOGRÁFICA	
TEMA	Performance Evaluation of Docker Container and Virtual Machine
TESIS	Brinda concepto clave de contenerización, la funcionalidad y las ventajas de trabajar con ella. Describe las partes de Docker y como aporta cada uno a la contenerización. Contiene un cuadro comparativo entre máquina virtual y contenedores Docker analizando el rendimiento, entradas y salidas, velocidad entre otros. Brinda herramientas para realizar una evaluación comparativa entre máquinas virtuales y Docker.
PROPÓSITO	Evaluar el rendimiento de Docker y máquina virtual y compararlo.
IDEAS CENTRALES	• La contenerización es más eficiente porque no hay un sistema operativo invitado. • Los contenedores Docker siempre se ejecutan sobre el kernel del sistema operativo en un espacio aislado. • La plataforma más conocida de contenerización se llama Docker. • Docker posee 4 partes fundamentales. • La tecnología Docker se refiere principalmente a la arquitectura cliente-servidor. • Las máquinas virtuales se vuelven ineficientes cuando los usuarios las

	usan para ejecutar aplicaciones pequeñas.
CONCEPTOS CLAVES	Contiene conceptos de contenerización, Docker y máquinas virtuales. Herramientas: Sysbench, Phorix, Apache benchmark, INTmem, FLOATmen e IOzone.
CONCLUSIONES	Los contenedores Docker funcionan mejor que las máquinas virtuales en las pruebas realizadas debido a que son una tecnología de virtualización ligera emergente.
APORTE A TEMA ELEGIDO	Aporta brindando información más detallada de acuerdo al rendimiento y velocidad del uso de máquinas virtuales y Docker que posteriormente se podrá verificar en la ejecución del tema planteado, de igual forma brinda información más detallada de Docker con la cual se puede entender mejor sus componentes y cuál será el mejor usar para el testeo de aplicaciones, de igual forma brinda aplicaciones que pueden ayudar a medir el rendimiento de Docker que pueden ayudar a cumplir los objetivos específicos.

Anexo 4.2 Ficha Bibliográfica 2

FICHA BIBLIOGRÁFICA	
TEMA	Performance Comparison Between Virtual Machines and Docker Containers

TESIS	Brinda información de investigaciones ya realizadas sobre comparación entre máquinas virtuales y Docker, herramientas para realizar comparación de rendimiento y consumo de recursos, las técnicas para realizar una correcta comparación.
PROPÓSITO	Comparar el rendimiento de una máquina virtual frente a un contenedor Docker.
IDEAS CENTRALES	• Los contenedores surgieron como una nueva • alternativa a las máquinas virtuales. • Los contenedores tienen paquetes e imágenes con sus propias bibliotecas y archivos que aseguran la interoperabilidad entre las distintas solicitudes de la aplicación. • Los contenedores son la sucesión de las máquinas virtuales. • Docker es mejor en entornos de gran escala de tiempo. • Existen 3 técnicas que se deben usar para comparar rendimiento: Medición, modelado, simulación.
CONCEPTOS CLAVES	Contiene conceptos de hipervisor, máquina virtual, virtualización, contenedor y Sysbench.
CONCLUSIONES	Si se quiere realizar una correcta comparación entre máquinas virtuales y Docker se debe tener en cuenta los pasos a seguir y las actividades que se quiere realizar, en este caso se presentó una guía detallada de las comparaciones que se hizo y se puede concluir que Docker

	presento un mejor desempeño ante la máquina virtual.
APORTE A TEMA ELEGIDO	Brinda una idea global de cómo se podría realizar una comparación entre máquinas virtuales y Docker para ver su rendimiento, de igual forma explica las correctas técnicas que se debe implementar si se desea comparar las dos herramientas lo cual aporta al tema de forma que si se sigue esos pasos se podrá conseguir un correcto análisis para la posterior comparación que se va a realizar en base a los objetivos propuestos.

Anexo 4.3 Ficha Bibliográfica 3

FICHA BIBLIOGRÁFICA	
TEMA	Software Containerization with Docker
TESIS	Da a conocer un ligero resumen sobre Docker y lo compara con máquinas virtuales, explica más a fondo sobre lo que es una imagen Docker y explica la separación entre el cliente y el demonio Docker, brinda un ejemplo de cómo se debe usar Docker en diferentes sistemas operativos.
PROPÓSITO	Brindar ideas sobre Docker para la implementación de este en cualquier empresa y lo demuestras con 2 ejemplos específicos en diferentes sistemas operativos.

IDEAS CENTRALES	• Los contenedores no reemplazan la virtualización tradicional. • Docker ha evolucionado significativamente en los últimos años. • Docker proporciona un nivel de aislamiento reducido en comparación con las máquinas virtuales. • La mayoría de las empresas se han ido involucrando con Docker en los últimos años gracias a su flexibilidad y portabilidad. • Docker posee características relevantes al momento de compararlo con máquinas virtuales, tales como: tamaño, flexibilidad, rapidez y reusabilidad.
CONCEPTOS CLAVES	Contiene los conceptos claves de toda la arquitectura de Docker, concepto de Docker y de máquina virtual.
CONCLUSIONES	Los contenedores no fueron creados para reemplazar a las máquinas virtuales sin embargo hoy en día están funcionando como una nueva alternativa de virtualización sin hipervisor. El ecosistema de Docker es muy amplio por explorar, posee apartados específicos para actividades específicas que ayudaran a que los equipos obtengan iteraciones de software más rápidas sin necesidad de gastar en un mejor hardware.
APORTE A TEMA ELEGIDO	Aporta con una idea más clara y practica de cómo se puede implementar Docker en distintos sistemas operativos esto servirá para la práctica cuando se desee

	demostrar el rendimiento de Docker frente a máquinas virtuales y para diferentes testeos que se realizarán.

Anexo 4.4 Ficha Bibliográfica 4

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace CRUD_LibrosVSC.Models
{
    public class Libro
    {
        [Required]
        public int id { get; set; }
        [Required]
        public string nombre { get; set; }
        [Required]
        public int paginas { get; set; }
        [Required]
        public string editorial { get; set; }
        [Required]
        public string autor { get; set; }
        [Required]
        public float precio { get; set; }
    }
}
```

Anexo 4.5 Clase libro CRUD

```
using CRUD_LibrosVSC.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;

namespace CRUD_LibrosVSC.Controllers
{
    public class HomeController : Controller
    {
        LibroDataAccess objlibro = new
LibroDataAccess();

        public IActionResult Index()
        {
            List<Libro> lstLibro = new List<Libro
>();
            lstLibro = objlibro.TraerLibros().
ToList();

            return View(lstLibro);
        }
        [HttpGet]
public IActionResult Create()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create([Bind] Libro libro)
{
    if (ModelState.IsValid)
    {
        objlibro.AgregarLibro(libro);
        return RedirectToAction("Index");
    }
    return View(libro);
}

[HttpGet]
public IActionResult Update(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    Libro libro = objlibro.TraerLibroPorId(id);

    if (libro == null)
    {
        return NotFound();
    }
    return View(libro);
}

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Update(int id, [Bind]Libro
libro)
{
    if (id != libro.id)
    {
        return NotFound();
    }
    if (ModelState.IsValid)
    {
        objlibro.ActualizarLibro(libro);
        return RedirectToAction("Index");
    }
    return View(libro);
}

[HttpGet]
public IActionResult Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    Libro libro = objlibro.TraerLibroPorId(id);

    if (libro == null)
    {
        return NotFound();
    }
    return View(libro);
}

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public IActionResult DeleteConfirmed(int? id)
{
    objlibro.BorrarLibro(id);
    return RedirectToAction("Index");
}
    }
}
```

Anexo 4.6 Controlador CRUD

```
@model Libro

@{
    ViewData["Title"] = "Agregar un Libro";
}

<h1 class="bg-warning text-white" style=
"text-align: center; background-color: #fffc95;">
Agregar un Libro</h1>
<a asp-action="Index" class="btn btn-secondary"
style="background-color: #a52f47;">
Ver todos los Libros</a>

<div asp-validation-summary="All" class=
"text-danger"></div>

<div>
<form method="post" enctype="multipart/form-data">
    <div class="form-group">
        <label asp-for="nombre"></label>
        <input type="text" asp-for="nombre" class=
"form-control" />
        <span asp-validation-for="nombre" class=
"text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="paginas"></label>
        <input type="text" asp-for="paginas" class=
"form-control" />
        <span asp-validation-for="paginas" class=
"text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="editorial"></label>
        <input type="text" asp-for="editorial"
class="form-control" />
        <span asp-validation-for="editorial" class=
"text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="autor"></label>
        <input type="text" asp-for="autor" class=
"form-control" />
        <span asp-validation-for="autor" class=
"text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="precio"></label>
        <input type="text" asp-for="precio" class=
"form-control" />
        <span asp-validation-for="precio" class=
"text-danger"></span>
    </div>
    <button type="submit" class="btn btn-primary"
style=
"text-align: center; background-color: #f1819a; bor
der-color:#b35b7f;"
>Agregar</button>
</form>
</div>
@section Scripts {
    @{await Html.RenderPartialAsync
("_ValidationScriptsPartial");}
}
```

Anexo 4.7 Pantalla Agregar Libro

```
@model List<Libro>
@{
    ViewData["Title"] = "Home Page";
}

<div style="text-align:center;">
    <br />
    <h1 class="bg-warning text-white" style=
"text-align: center; background-color: #fffc95;">
Libreria Lizbeth</h1>
    <a asp-action="Create" class=
"btn btn-secondary" style=
"text-align: center; background-color: #f1819a; bor
der-color:#b35b7f;"
>Agregar un Libro</a>
    <!--
<a asp-action="Masive" class="btn btn-secondary" st
yle="text-align: center; background-color: #a52f47;
border-color: #a52f47; ">Agregar Libros Masivo</a>
-->
    <br />
    <br />
    <table class="table table-sm table-bordered">
        <tr class="table-warning">
            <th>Id</th>
            <th>Nombre</th>
            <th>Paginas</th>
            <th>Editorial</th>
            <th>Autor</th>
            <th>Precio</th>
            <th colspan="2"></th>
        </tr>
        @foreach (Libro l in Model)
        {
            <tr>
                <td>@l.id</td>
                <td>@l.nombre</td>
                <td>@l.paginas</td>
                <td>@l.editorial</td>
                <td>@l.autor</td>
                <td>@l.precio</td>
                <td>
                    <a class="btn btn-sm"
asp-action="Update" asp-route-id="@l.id" style=
"background-color: #965595; border-color: #7a3e61;
 color:white;"
>
                        Actualizar
                    </a>
                </td>
                <td>
                    <form asp-action="Delete"
asp-route-id="@l.id" method="post">
                        <button type="submit" class
="btn btn-sm btn-danger">
                            Borrar
                        </button>
                    </form>
                </td>
            </tr>
        }
    </table>
</div>
```

Anexo 4.8 Pantalla Principal

```
@model Libro

@{
    ViewData["Title"] = "Actualizar un Libro";
}

<h1 class="bg-warning text-white" style=
"text-align: center; background-color: #fffc95;">
Actualizar un Libro</h1>
<a asp-action="Index" class="btn btn-secondary"
style="background-color: #a52f47;">
Ver todos los Libros</a>

<div asp-validation-summary="All" class=
"text-danger"></div>
<div>
    <form method="post" enctype=
"multipart/form-data">
        <div class="form-group">
            <label asp-for="id"></label>
            <input type="text" asp-for="id"
readonly class="form-control" />
        </div>
        <div class="form-group">
            <label asp-for="nombre"></label>
            <input type="text" asp-for="nombre"
class="form-control" />
            <span asp-validation-for="nombre" class
="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="paginas"></label>
            <input type="text" asp-for="paginas"
class="form-control" />
            <span asp-validation-for="paginas"
class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="editorial"></label>
            <input type="text" asp-for="editorial"
class="form-control" />
            <span asp-validation-for="editorial"
class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="autor"></label>
            <input type="text" asp-for="autor"
class="form-control" />
            <span asp-validation-for="autor" class=
"text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="precio"></label>
            <input type="text" asp-for="precio"
class="form-control" />
            <span asp-validation-for="precio" class
="text-danger"></span>
        </div>
        <button type="submit" class=
"btn btn-primary" style=
"text-align: center; background-color: #f1819a; bor
der-color:#b35b7f;"
>Actualizar</button>
    </form>
</div>
@section Scripts {
    @{await Html.RenderPartialAsync
("_ValidationScriptsPartial");}
}
```

Anexo 4.9 Pantalla actualizar libro

```
using CRUD_LibrosVSC.Models;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want
 to change this for production scenarios, see http
s://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern:
"{controller=Home}/{action=Index}/{id?}");

app.Run();

var connectionString = builder.Configuration.
GetConnectionString("DevConnection");

builder.Services.AddDbContext<LibroContext>(options
 =>
   options.UseMySql(connectionString,
                new MySqlServerVersion(new Version
(8, 0, 11))));
builder.Services.AddControllersWithViews();
```

Anexo 4.10 Programa c#