UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

Tema:

GENERACIÓN DE ESTRATEGIAS PARA OPTIMIZAR EL DESARROLLO CON CONTENEDORES CONSTRUYENDO IMÁGENES DOCKER EFICIENTES

Trabajo de titulación modalidad Proyecto de Investigación, presentado previo a la obtención del título de Ingeniero en Tecnologías de la Información.

ÁREA: Hardware y redes
LÍNEA DE INVESTIGACIÓN: Sistemas administradores de recursos
AUTOR: Bryan Anthony López Guerrero
TUTOR: Ing. Mg. Franklin Oswaldo Mayorga Mayorga

Ambato - Ecuador
agosto – 2023

APROBACIÓN DEL TUTOR

En calidad de tutor del trabajo de titulación con el tema: GENERACIÓN DE ESTRATEGIAS PARA OPTIMIZAR EL DESARROLLO CON CONTENEDORES CONSTRUYENDO IMÁGENES DOCKER EFICIENTES, desarrollado bajo la modalidad Proyecto de Investigación por el señor Bryan Anthony López Guerrero, 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 el estudiante ha sido tutorado durante todo el desarrollo del trabajo hasta su conclusión, de acuerdo a lo dispuesto en el Artículo 17 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.3 del instructivo del reglamento referido.

Ambato, agosto 2023.

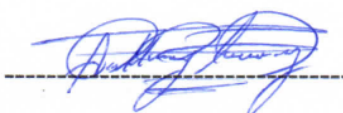
Ing. Mg. Franklin Oswaldo Mayorga Mayorga

TUTOR

AUTORÍA

El presente trabajo de titulación titulado: GENERACIÓN DE ESTRATEGIAS PARA OPTIMIZAR EL DESARROLLO CON CONTENEDORES CONSTRUYENDO IMÁGENES DOCKER EFICIENTES es absolutamente original, auténtico y personal y ha observado los preceptos establecidos en la Disposición General Quinta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato. En tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato, agosto 2023.



Bryan Anthony López Guerrero

C.C. 1804397220

AUTOR

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato para que reproduzca total o parcialmente este trabajo de titulación dentro de las regulaciones legales e institucionales correspondientes. Además, cedo todos mis derechos de autor a favor de la institución con el propósito de su difusión pública, por lo tanto, autorizo su publicación en el repositorio virtual institucional como un documento disponible para la lectura y uso con fines académicos e investigativos de acuerdo con la Disposición General Cuarta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato.

Ambato, agosto 2023.

Bryan Anthony López Guerrero

C.C. 1804397220

AUTOR

APROBACIÓN DEL TRIBUNAL DE GRADO

En calidad de par calificador del informe final del trabajo de titulación presentado por el señor Bryan Anthony López Guerrero, 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 GENERACIÓN DE ESTRATEGIAS PARA OPTIMIZAR EL DESARROLLO CON CONTENEDORES CONSTRUYENDO IMÁGENES DOCKER EFICIENTES, nos permitimos informar que el trabajo ha sido revisado y calificado de acuerdo al Artículo 19 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.4 del instructivo del reglamento referido. Para cuya constancia suscribimos, conjuntamente con la señora Presidente del Tribunal.

Ambato, agosto 2023.

Firmado electrónicamente por:
ELSA PILAR URRUTIA URRUTIA

Ing. Elsa Pilar Urrutia 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

DEDICATORIA

Con este capítulo que llega a su fin, abro las puertas a un nuevo comienzo lleno de posibilidades y sueños por cumplir. En este viaje, he contado con el amor incondicional y el apoyo invaluable de mis queridos padres, Eduardo Gonzalo y Flor María. A lo largo de los años, han sido faros de luz en mis días más oscuros y cómplices de mis momentos de alegría. A ustedes les dedico el fruto de este proyecto, como una pequeña muestra de mi profunda gratitud por ser mis guías, mis héroes y mis mayores motivaciones.

A ti, Alexis, mi hermano menor, te dedico cada logro como un intento constante de ser el ejemplo que mereces tener en tu vida. Tu presencia me inspira a superarme día a día.

Desde el fondo de mi corazón, dedico con amor y cariño este logro a cada uno de ustedes, quienes han dejado una huella imborrable en mi camino.

Bryan Anthony López Guerrero

AGRADECIMIENTO

A mis padres, Eduardo Gonzalo y Flor María. Su constante apoyo y amor incondicional han sido los cimientos sólidos sobre los cuales he construido este logro. Gracias a su ejemplo, he aprendido la importancia del esfuerzo y la perseverancia. Cada logro mío es un reflejo de la dedicación y valores que me han transmitido.

A mi tío, Dr. Homero Vargas. Un sincero reconocimiento por su apoyo constante y por demostrarme que con esfuerzo y trabajo duro se pueden alcanzar las cimas más altas.

A mis amigos, Anthony, David, Felipe, Luis y Paúl, les debo un sincero agradecimiento por su amistad sincera y su apoyo constante en este viaje.

Un agradecimiento especial al Ing. Franklin Mayorga, mi tutor, por su orientación y dedicación en este proyecto. Su guía y apoyo han sido fundamentales para mí.

ÍNDICE GENERAL DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN EJECUTIVO

Las nuevas tecnologías están revolucionando el campo de la informática, aportando soluciones innovadoras para el desarrollo y despliegue de software. Entre estas tecnologías, los contenedores Docker han ganado una atención significativa debido a su versatilidad y eficiencia.

Los contenedores Docker ofrecen numerosas ventajas al sector informático, entre ellas una mayor portabilidad, escalabilidad y utilización de recursos. Al encapsular las aplicaciones y sus dependencias, los contenedores permiten un despliegue coherente en distintos entornos, lo que simplifica el proceso de entrega de software. Además, los contenedores promueven la asignación eficiente de recursos, lo que permite a las organizaciones optimizar los costes de infraestructura y lograr un mejor rendimiento.

El desarrollo eficiente de imágenes de contenedores es crucial para maximizar los beneficios de los contenedores Docker. Las imágenes optimizadas contribuyen a un despliegue más rápido, a reducir los requisitos de almacenamiento y a mejorar el rendimiento general del sistema. Esta investigación se centra en el desarrollo de estrategias para crear imágenes Docker eficientes mediante el análisis de las prácticas existentes, la identificación de técnicas de optimización y la realización de evaluaciones de rendimiento.

La investigación pretende demostrar el impacto del desarrollo eficiente de imágenes en el rendimiento de los contenedores, el consumo de recursos y la escalabilidad. Mediante el análisis comparativo y la evaluación de varias técnicas de optimización, el estudio mostrará cómo las imágenes de contenedores bien diseñadas pueden mejorar el campo de la informática. Los resultados permitirán a los profesionales de la informática optimizar sus flujos de trabajo de imágenes Docker, lo que mejorará el rendimiento del software, ahorrará costes y agilizará los procesos de desarrollo.

Palabras clave: Nuevas tecnologías, contenedores Docker, informática, desarrollo eficiente de imágenes, estrategias de optimización, evaluación del rendimiento.

ABSTRACT

New technologies are revolutionizing the field of informatics, providing innovative solutions for software development and deployment. Among these technologies, Docker containers have gained significant attention due to their versatility and efficiency.

Docker containers offer numerous benefits to the informatics sector, including enhanced portability, scalability, and resource utilization. By encapsulating applications and their dependencies, containers enable consistent deployment across different environments, simplifying the software delivery process. Additionally, containers promote efficient resource allocation, allowing organizations to optimize infrastructure costs and achieve better performance.

Efficient container image development is crucial for maximizing the benefits of Docker containers. Optimized images contribute to faster deployment, reduced storage requirements, and improved overall system performance. This research focuses on developing strategies to create efficient Docker images by analyzing existing practices, identifying optimization techniques, and conducting performance evaluations.

The research aims to demonstrate the impact of efficient image development on container performance, resource consumption, and scalability. By benchmarking and evaluating various optimization techniques, the study will showcase how well-designed container images can enhance the informatics field. The findings will empower informatics professionals to optimize their Docker image workflows, leading to improved software performance, cost savings, and streamlined development processes.

Keywords: New technologies, Docker containers, informatics, efficient image development, optimization strategies, performance evaluation.

CAPÍTULO I.- MARCO TEÓRICO

1.1. Tema de investigación

GENERACIÓN DE ESTRATEGIAS PARA OPTIMIZAR EL DESARROLLO CON CONTENEDORES CONSTRUYENDO IMÁGENES DOCKER EFICIENTES.

1.1.1. Planteamiento del problema

Cada año se gastan cientos de miles de millones de dólares en Tecnologías de la información y la comunicación (TIC), lo que refleja una fuerte creencia mundial en el potencial transformador de estas nuevas tecnologías. Para las empresas multinacionales, sin duda, las TIC se han convertido en esenciales. La globalización exige flujos de información y procesamiento de la información que, sencillamente, no podría tener lugar sin las TIC. Estas nuevas tecnologías facilitan enormemente la adquisición y absorción de conocimientos, ofreciendo oportunidades sin precedentes para mejorar los sistemas informáticos y ampliar el abanico de oportunidades para las pequeñas empresas y microempresas.

A nivel mundial las herramientas de virtualización han ganado mucha importancia gracias a las máquinas virtuales, que no son más que ordenadores de software que otorgan la mayoría de las funcionalidades que las máquinas físicas ofrecen. Estas tienen la capacidad de ejecutarse de forma independiente en el computador ya que se encuentran separadas de forma lógica; además, si de alguna forma una máquina instalada llega a tener fallos estos no afectan en absoluto a las demás, ya que su funcionamiento es a través de un sistema operativo individual completo junto a sus aplicaciones [1].

En años recientes nuevas tecnologías de virtualización han aparecido en el mercado mundial, como Docker, que trata de una herramienta de código abierto, la cual provee al usuario una forma de generar una capa de abstracción para implementar servicios, lo que la convierte en una de las mejores alternativas de virtualización liviana basada en contenedores de aplicaciones, los cuales tienen la virtud de ejecutar procesos de manera aislada [1]. Actualmente esta tecnología es usada activamente por millones de desarrolladores y testers en el mundo. La importancia de Docker se encuentra alojada

en la capacidad de construir imágenes capaces de crear y compartir un entorno de trabajo completo junto a todo el software requerido, configuración detallada y a medida de los recursos proporcionados por el ordenador anfitrión con el fin de probar e implementar aplicaciones rápidamente, ya sea en la nube o localmente [2].

La Internet se ha convertido en un medio fundamental para que la mayoría de las empresas puedan salir adelante, sin embargo, Ecuador aún se encuentra en vías de desarrollo y no posee los avances tecnológicos que disponen otros países, lo que conlleva a que la mayoría de las compañías carezcan de conocimientos sobre nuevas tecnologías y recientes avances tecnológicos que son de suma importancia ya que permitirían optimizar el funcionamiento de su software [3], por ello, es recomendable buscar ingenieros en Tecnologías de la Información que sean capaces de adentrarse aún más en el mundo de nuevas tecnologías e investigar como los países más desarrollados hacen uso de las mismas para cumplir con los objetivos planteados por la empresa. La mayoría de grandes empresas ecuatorianas han decidido cambiar su sistema operativo, pasando de Windows a Linux, debido a los grandes ventajes que este último brinda, como, por ejemplo, mayor rendimiento de Sistema Operativo (S.O.), velocidad de flujo de datos, estabilidad, granularidad en manejo de permisos, más seguridad, etc.

El gran problema de las empresas en la ciudad de Ambato puede radicar en la gran ausencia de conocimiento para poder implementar este S.O. o simplemente en el miedo a implementar algo “desconocido” para la mayoría de las personas. Esto genera que algunas compañías ecuatorianas se priven de grandes mejoras en sus sistemas, las cuáles ayudarían a producir mejores resultados con los mismos recursos que ya fueron asignados previamente [3]. Es de suma importancia que las empresas empiecen a ver la tecnología con la misma importancia que miran sus finanzas, ya que, si estas cuentan con un sistema informático incapaz de cumplir con lo que se espera de este, se podrían generar grandes pérdidas para la compañía, volviendo a esta vulnerable y dejándola mal posicionada en el basto y agresivo mercado.

1.2. Antecedentes investigativos

Una vez realizado el análisis de fuentes de investigación alojadas dentro de los repositorios de distintas universidades del Ecuador, se obtuvieron diversos proyectos de investigación cuya información servirá de apoyo para el presente proyecto.

Según Arboleda Bonilla [3], a través de la implementación de Benchmarking y en base a los datos recolectados se determinó que la mejor estrategia es usar contenedores Docker optimizados, teniendo en cuenta que estos consumen menos recursos, son más veloces, fáciles y rápidos de usar e instalar que una máquina virtual debido a que contienen un sistema operativo completo, pero sin llegar a utilizar el sistema operativo de la máquina en la que se está ejecutando por lo que optimiza recursos y funciona de manera más flexible.

Según Vizcaíno Quiroz [4], a través de la evaluación de eficiencia de desempeño establecida en la ISO/IEC 2510, se determinó que la correcta implementación de la aplicación con Docker con un menor tiempo de ejecución, mejor manejo de recursos y sin conflictos de versiones, demostró la eficiencia y facilidad de trabajo en la creación de contenedores que comparten los mismos recursos del sistema permitiendo el uso de diferentes aplicaciones.

Según Montalvo Ochoa [5], la correcta implementación y uso de Docker como sistema de contenedores de software ha demostrado tener la capacidad de agilizar el proceso de despliegue de aplicaciones y servicios en entornos de prueba y/o producción, además de ofrecer al usuario la posibilidad de crear aplicaciones ligeras y portables las cuales pueden ejecutarse en cualquier otro equipo o servidor que cuente con una instalación de Docker.

Según Antepara Reyes y Villamar Flores [6], a través de un análisis de software especializado en la sintetización de aplicativos informáticos, se determinó que es factible levantar una infraestructura optimizada de software basado en herramientas Docker con diversas funcionalidades para ambientes web mediante el estudio de artículos científicos y bibliografía de alto impacto.

Según Pacheco Laje [7], a través de la implementación de microservicios basados en contenedores Dockers, se obtuvo una mejora considerable en los tiempos de respuesta, con un promedio superior al 20% que la arquitectura tradicional, además, con este prototipo se pudo verificar que el tiempo en nuevos despliegues se reduce del 20% al 0%.

1.3. Fundamentación teórica

Ingeniería de software

La Ingeniería de software es un área de conocimiento que ofrece una variedad de métricas y metodologías que pueden ser empleadas para asegurar el correcto desarrollo del software. Esta disciplina permite gestionar el personal que participa en los diferentes proyectos de software, los ciclos de vida del proyecto, los costos asociados al mismo y otros factores administrativos relevantes. Las metodologías utilizadas también son conocidas como modelos de proceso de software, y proveen guías para construir software de alta calidad [8].

Desarrollo de software

El Desarrollo de software es un conjunto de actividades informáticas que están dedicadas al proceso de creación, diseño, despliegue y compatibilidad de software [9]. Programadores, ingenieros de software y desarrolladores de software son las personas que principalmente llevan a cabo el desarrollo de software. Los roles mencionados interactúan entre sí y se superponen, y la dinámica entre ellos va variando mucho entre los departamentos y comunidades de desarrollo.

Programadores o codificadores

Se encargan de escribir el código fuente para realizar tareas en específico como fusionar bases de datos, enrutar comunicaciones o mostrar textos y gráficos. Estas personas interpretan las instrucciones que los desarrolladores e ingenieros de software les proporcionan y utilizan lenguajes de programación como C# o Python para llevarlas a cabo [10].

Ingenieros de software

Los profesionales de la ingeniería de software aplican los principios de la ingeniería para diseñar software y sistemas que sean útiles para abordar una amplia gama de problemas. Utilizan lenguajes de modelado y otras herramientas para idear soluciones generales para la mayoría de los problemas, en lugar de solo generar soluciones para un problema específico. Las soluciones propuestas por un ingeniero de software deben seguir el método científico y deben ser efectivas en el mundo real, de la misma manera en que un puente o un ascensor lo sería [8].

Desarrolladores de software

Los Desarrolladores de software tienen un rol menos formal que los ingenieros y estos pueden participar en áreas específicas del proyecto, en dónde está incluida la escritura de código. Al mismo tiempo, se encargan de impulsar el ciclo de vida general del desarrollo de software mediante el trabajo colaborativo funcional con el fin de transformar los requisitos en funciones, realización de pruebas y mantenimiento de software [11].

Metodologías de desarrollo de software

Una Metodología de desarrollo de software se define como un marco de trabajo que se usa para controlar, estructurar y planificar el proceso de desarrollo de sistemas de información [12]. Una metodología para el desarrollo de sistemas no necesariamente tiene que ser la más adecuada para usarla en cualquier tipo de proyecto. Cada una de las metodologías existentes son más o menos adecuadas para tipos específicos de proyectos, teniendo en cuenta varias consideraciones técnicas, organizacionales, de proyecto y de equipo.

Existen diversas propuestas metodológicas que repercuten directamente en el desarrollo del software, las cuales son:

- **Metodologías tradicionales**

  Estas se centran en el control del proceso, caracterizadas por la rigurosidad de las actividades, el producto final que se obtiene y, las herramientas que se van

a emplear. Estas metodologías pueden ser efectivas y necesarias en una gran cantidad de proyectos, sin embargo, muestran graves falencias en proyectos con requerimientos que van cambiando con el tiempo [12].

- **Metodologías de desarrollo ágil**

  Estas están centradas en el factor humano, es decir, la comunicación, colaboración y relación con el cliente, también se encuentran enfocadas en el desarrollo incremental con iteraciones cortas. Estas metodologías son ideales para aquellos proyectos cuyos requerimientos son cambiantes y cuyo tiempo de desarrollo tiene la tendencia a acortarse, lo cual contrasta en gran medida con las metodologías ágiles [13].

Generación de estrategias de optimización

La optimización del rendimiento de los programas y el software es el proceso de modificación de un sistema de software para que funcione de forma más eficiente y se ejecute más rápidamente. La optimización del rendimiento es clave para tener una aplicación que funcione de forma eficiente y se lleva a cabo mediante la supervisión y el análisis del rendimiento de una aplicación y la identificación de formas de mejorarlo [4].

La optimización del rendimiento suele centrarse en mejorar sólo uno o dos aspectos del rendimiento del sistema, por ejemplo, el tiempo de ejecución, el uso de la memoria, el espacio en disco, el ancho de banda, etc. Esto suele requerir una compensación en la que un aspecto se implementa a expensas de otros. Por ejemplo, aumentar el tamaño de la caché mejora el rendimiento en tiempo de ejecución, pero también aumenta el consumo de memoria [4].

Open Source

También llamado “código abierto”, es un término utilizado para describir un determinado tipo de software que se distribuye bajo una licencia que permite al usuario final, siempre y cuando tenga los conocimientos necesarios, utilizar el código fuente del programa para estudiarlo, modificarlo y mejorarlo, e incluso redistribuirlo.

Este tipo de software ofrece una serie de características y ventajas únicas, ya que los programadores, al tener una vía de acceso directa al código fuente de un determinado programa, pueden leerlo e incluso modificarlo, y así mejorarlo, añadiendo una serie de opciones y corrigiendo los posibles problemas que puedan encontrarse, de manera que el programa una vez sea compilado estará mucho mejor diseñado que cuando salió del ordenador de su programador original [14].

Virtualización basada en contenedores

La virtualización de contenedores (a menudo denominada virtualización de sistema operativo) es algo más que un tipo diferente de hipervisor. Los contenedores utilizan el sistema operativo del host como base, y no el hipervisor. En lugar de virtualizar el hardware (que requiere imágenes completas del sistema operativo virtualizado para cada huésped), los contenedores virtualizan el propio sistema operativo, compartiendo el núcleo del sistema operativo del host y sus recursos tanto con el host como con otros contenedores [15].

Los contenedores proporcionan los elementos esenciales necesarios para que cualquier aplicación se ejecute en un sistema operativo anfitrión. Se podría pensar en ellos como máquinas virtuales reducidas que ejecutan el software suficiente para desplegar una aplicación. Muchas aplicaciones (como los servidores de bases de datos, que funcionan mejor utilizando almacenamiento en bloque) requieren un acceso directo a los recursos de hardware. Conseguir ese acceso a los discos y dispositivos de red a través de la emulación de hardware de un hipervisor suele afectar negativamente a el rendimiento. La virtualización basada en contenedores ayuda simplemente evitando la capa de emulación, lo que proporciona un mejor manejo de recursos y evita pérdida de rendimiento [15].

Software basado en microservicios

Una arquitectura de microservicios es una forma de estructurar aplicaciones. Con el auge de los contenedores, los desarrolladores han empezado a dividir los monolitos en microservicios. La idea central se basa en construir una aplicación como un conjunto de servicios libremente acoplados que pueden ser actualizados y escalados por separado bajo la infraestructura de un contenedor [16].

Los microservicios permiten utilizar la "herramienta adecuada para la tarea adecuada", lo que significa que las aplicaciones pueden desarrollarse y entregarse mediante la tecnología que mejor se adapte a la tarea, en lugar de quedar atrapadas en una única tecnología, tiempo de ejecución o marco de trabajo. Gracias a esto se puede escalar el desarrollo y entrega de aplicaciones grandes y complejas mediante su descomposición, lo que permite que los componentes individuales evolucionen de forma independiente [16].

Modelado de imágenes Docker

Una imagen Docker es una plantilla de sólo lectura que contiene un conjunto de instrucciones para crear un contenedor que puede ejecutarse en la plataforma Docker. Proporciona una forma cómoda de empaquetar aplicaciones y entornos de servidor preconfigurados, que puede utilizar para su uso privado o compartir públicamente con otros usuarios de Docker. Las imágenes Docker son también el punto de partida para cualquier persona que utilice Docker por primera vez [17].

Una imagen Docker se compone de una colección de archivos que reúnen todos los elementos esenciales (como instalaciones, código de aplicación y dependencias) necesarios para configurar un entorno de contenedores totalmente operativo. Se puede crear una imagen Docker utilizando uno de los siguientes métodos:

- **Interactivo**: Ejecutando un contenedor desde una imagen Docker existente, cambiando manualmente ese entorno de contenedores a través de una serie de pasos en vivo, y guardando el estado resultante como una nueva imagen [19].
- **Dockerfile**: Construyendo un Dockerfile, que es un archivo de texto plano que proporciona las especificaciones para crear una imagen Docker [17].

Desarrollo con contenedores

El Desarrollo de contenedores (Dev Container) permite generar y utilizar un contenedor como un entorno de desarrollo, el cual se puede utilizar para ejecutar una aplicación, para separar herramientas, bibliotecas o los tiempos de ejecución necesarios para trabajar con un código base, y para ayudar en la integración y las

pruebas continuas. Los contenedores de desarrollo pueden ejecutarse de forma local o remota, en una nube privada o pública [18].

La especificación de contenedores de desarrollo pretende encontrar formas de enriquecer los formatos existentes con ajustes, herramientas y configuraciones comunes específicas para el desarrollo, sin dejar de ofrecer una opción de contenedor único simplificada y no orquestada, de modo que puedan utilizarse como entornos de codificación o para la integración y las pruebas continuas. Más allá de los metadatos centrales de la especificación, la especificación también permite a los desarrolladores compartir y reutilizar rápidamente los pasos de configuración del contenedor a través de las características y plantillas del contenedor de desarrollo [18].

Metodologías de desarrollo

Las metodologías de desarrollo de software son una serie de enfoques y técnicas que se utilizan para planificar, diseñar, desarrollar, probar y mantener sistemas de software de alta calidad. Estas metodologías pueden ser vistas como un conjunto de prácticas recomendadas que ayudan a los equipos de desarrollo de software a crear sistemas más efectivos y eficientes.

El objetivo principal de una metodología de desarrollo de software es proporcionar un marco de trabajo estructurado para el proceso de desarrollo de software. Esto ayuda a los equipos de desarrollo de software a entender el alcance del proyecto, definir las actividades necesarias, identificar los roles y responsabilidades de los miembros del equipo, y establecer un conjunto de metas y objetivos claros para el proyecto.

Estas metodologías de desarrollo se utilizan para gestionar el proceso de desarrollo de software de principio a fin, desde la planificación y el diseño hasta la implementación y el mantenimiento. Al adoptar una metodología de desarrollo de software, las organizaciones pueden garantizar que los proyectos de software se entreguen a tiempo, dentro del presupuesto y con la calidad requerida.

Benchmarking

El benchmarking es una técnica que se utiliza para medir y comparar el rendimiento de un producto o servicio con los estándares de la industria. Es una herramienta útil para las empresas que desean mejorar su desempeño y ganar una ventaja competitiva en el mercado. El proceso implica la identificación de los productos o servicios líderes en la industria, el análisis de sus fortalezas y debilidades, y la evaluación de cómo se comparan con los propios productos o servicios de la empresa.

El objetivo principal del benchmarking es mejorar la calidad y la eficiencia de los productos o servicios de la empresa, y reducir los costos. Al comparar el rendimiento de la empresa con los líderes de la industria, se pueden identificar las áreas de mejora y desarrollar estrategias para mejorar la calidad y eficiencia de los procesos de la empresa. Además, el benchmarking también puede ayudar a la empresa a identificar nuevas oportunidades de mercado y a desarrollar productos o servicios innovadores que satisfagan mejor las necesidades de los clientes.

Distribuciones GNU/Linux

Las distribuciones GNU/Linux, comúnmente conocidas como "distros", son sistemas operativos basados en el kernel Linux y en herramientas de software libre y de código abierto. Las distros pueden incluir diferentes componentes, como el entorno de escritorio, el gestor de paquetes y las aplicaciones preinstaladas. Las distros pueden ser personalizadas para satisfacer las necesidades específicas de los usuarios y organizaciones.

Las distros de GNU/Linux son ampliamente utilizadas en todo el mundo para una variedad de propósitos, desde servidores web y de bases de datos hasta sistemas de escritorio y dispositivos embebidos. Debido a su naturaleza de código abierto, las distros de GNU/Linux permiten a los usuarios y desarrolladores acceder al código fuente y modificarlo según sea necesario. Esto hace que sea más fácil adaptar la distro para satisfacer las necesidades específicas de una organización o usuario, lo que puede ser particularmente útil para entornos empresariales.

Herramientas de benchmarking

Las herramientas de benchmarking para Linux son programas diseñados para evaluar el rendimiento de un sistema operativo Linux, ya sea en términos de velocidad, capacidad de procesamiento, consumo de recursos, entre otros. Estas herramientas permiten a los usuarios medir y comparar el desempeño de diferentes componentes del sistema, como el procesador, la memoria RAM, el disco duro y la tarjeta gráfica, entre otros.

El benchmarking es una práctica común en la industria de la informática y se utiliza para medir el rendimiento de hardware y software en diferentes sistemas. En el caso de Linux, estas herramientas son particularmente útiles para los administradores de sistemas y los desarrolladores de software que necesitan asegurarse de que sus sistemas operativos y aplicaciones estén funcionando de manera eficiente y sin problemas.

Docker Compose

Docker Compose es una herramienta que permite definir y gestionar múltiples contenedores de manera coordinada [19]. Su uso facilita la configuración y orquestación de contenedores interrelacionados, lo que contribuye a la optimización del entorno en varios aspectos:

- Gestión simplificada de múltiples contenedores: Docker Compose permite definir y gestionar varios contenedores como un conjunto único. Esto simplifica la administración y coordinación de contenedores relacionados, como aquellos que dependen unos de otros o que trabajan en conjunto para formar una aplicación completa.
- Configuración centralizada: Con Docker Compose, se puede definir la configuración de los contenedores en un archivo YAML centralizado. Esto facilita la gestión y mantenimiento de la configuración, ya que se encuentran en un solo lugar y se pueden realizar cambios y ajustes de forma más eficiente.
- Interconexión de contenedores: Docker Compose permite establecer dependencias y enlaces entre los contenedores definidos en el archivo. Esto garantiza que los contenedores se inicien y ejecuten en el orden adecuado,

asegurando que los servicios y aplicaciones dependientes estén disponibles y listos para su uso.

- Comunicación eficiente entre contenedores: Docker Compose crea una red interna para los contenedores definidos, lo que facilita la comunicación y el intercambio de datos entre ellos. Esto mejora la eficiencia y el rendimiento de las interacciones entre contenedores, evitando la necesidad de exponer puertos innecesarios o realizar configuraciones adicionales.

1.4. Objetivos

1.4.1. Objetivo general

Proponer estrategias para optimizar el desarrollo con contenedores construyendo imágenes Docker eficientes.

1.4.2. Objetivos específicos

- Analizar la forma en la que una imagen Docker se ha construido (capas, operaciones, configuración, etc.)
- Implementar benchmarking para probar el rendimiento que ofrece un contenedor generado en base a microservicios.
- Elaborar estrategias de optimización sobre la construcción de imágenes Docker para desarrollar contenedores más eficientes.

CAPÍTULO II.- METODOLOGÍA

2.1. Materiales

Debido a la naturaleza del proyecto de investigación se utilizó la revisión bibliográfica como técnica de investigación, que consiste en recopilar información relevante de fuentes bibliográficas, como libros, artículos científicos y documentos relacionados. El instrumento utilizado en esta técnica de investigación fue la ficha bibliográfica. Las fichas bibliográficas son herramientas estructuradas que permiten registrar de manera organizada la información relevante extraída de las fuentes bibliográficas consultadas.

Tabla 1. Estructura ficha bibliográfica

FICHA BIBLIOGRÁFICA	
TEMA	
RESUMEN DEL ARTÍCULO CIENTÍFICO	
PROPÓSITO	
IDEAS CENTRALES	
CONCEPTOS CLAVES	
CONCLUSIONES	
APORTE A TEMA ELEGIDO	

Autor: Anthony López

2.2. Métodos

2.2.1. Modalidad de investigación

- **Investigación de campo**

  El proyecto de investigación fue de campo porque se recopilaron datos de fuentes primarias, como lo son investigaciones previas realizadas sobre el tema planteado.

- **Investigación bibliográfica – documental**

  El proyecto de investigación fue bibliográfico – documental porque se documentó la recolección de datos de investigaciones previas del tema planteado, en artículos científicos, libros y tesis que hagan referencia a las variables planteadas para poder desarrollar investigación.

2.2.2. Población y muestra

Debido a la naturaleza del problema no se requiere de población y muestra ya que la investigación es netamente de revisión bibliográfica.

2.2.3. Recolección de información

Las fichas bibliográficas que se presentan a continuación realizan comparaciones entre las diferentes estructuras que una imagen Docker puede llegar a tener y específicamente buscan la mejor forma de modelar un contenedor en base a una imagen Docker modificada. Muestran varias herramientas de benchmarking que proveen datos y gráficas para comparar el rendimiento entre contenedores con cambios estructurales en las diferentes capas de las que está compuesta una imagen Docker.

Tabla 2. Ficha bibliográfica 1

FICHA BIBLIOGRÁFICA	
TEMA	Performance Evaluation of Docker Container and Virtual Machine
RESUMEN DEL ARTÍCULO CIENTÍFICO	El autor propone que los contenedores ofrecen la ventaja de ser portátiles y escalables, lo que significa que pueden ser desplegados fácilmente en diferentes entornos sin la necesidad de configuraciones complejas. Docker es una herramienta popular para la contenerización y consta de varias partes clave, incluyendo el demonio de Docker, la API de Docker y la interfaz de línea de comandos. Cada uno de estos componentes juega un papel importante en la creación, gestión y ejecución de contenedores Docker.
PROPÓSITO	Evaluar y comparar del rendimiento entre un contenedor Docker y una la máquina virtual.
IDEAS CENTRALES	• Los tres componentes principales de Docker son: demonio de Docker, la API de Docker y la interfaz de línea de comandos. • Para medir el rendimiento de Docker, existen varias herramientas disponibles, como Sysbench y BenchmarkDotNet.

	• En comparación con las máquinas virtuales, los contenedores Docker tienen un rendimiento superior, ya que comparten el sistema operativo subyacente con el host y no requieren un hipervisor.
CONCEPTOS CLAVES	Contiene conceptos clave sobre virtualización, contenedores Docker, máquinas virtuales y herramientas de benchmarking.
CONCLUSIONES	Los contenedores Docker rinden mejor que las máquinas virtuales, ya que la presencia de la capa QEMU en la máquina virtual la hace menos eficiente que los contenedores Docker. En términos de rendimiento de CPU, rendimiento de memoria, E/S de disco, prueba de carga y medición de la velocidad de operación, un contenedor Docker sobresale mucho más que una máquina virtual.
APORTE AL TEMA ELEGIDO	Aporta proporcionando información detallada sobre el rendimiento y velocidad del uso de máquinas virtuales y Docker es esencial para comprender cómo estas tecnologías pueden ayudar a las aplicaciones de prueba. En comparación con las máquinas virtuales, Docker es una opción más ligera y rápida. Como los contenedores Docker comparten el sistema operativo

	subyacente con el host, no necesitan un hipervisor, lo que significa que tienen una sobrecarga mucho menor y un mejor rendimiento en términos de velocidad de inicio y entrada y salida de datos.

Autor: Anthony López

Tabla 3. Ficha bibliográfica 2

FICHA BIBLIOGRÁFICA	
TEMA	A performance comparison of linux containers and virtual machines using Docker and KVM.
RESUMEN DEL ARTÍCULO CIENTÍFICO	El autor analiza formas de mejorar el rendimiento de la computación en la nube. Compara el rendimiento de Docker y de una máquina virtual basada en el núcleo KVM. Brinda tres tipos de comparaciones, con las cuales demuestra que Docker es más rápido que KVM.
PROPÓSITO	Comparar el rendimiento de contenedores Linux y máquinas virtuales utilizando Docker y KVM.
IDEAS CENTRALES	• Comparación del uso de CPU y memoria del sistema operativo anfitrión entre Docker y KVM. • Medición en reposo del uso de CPU y memoria y del rendimiento de E/S

	mediante la copia de archivos de gran tamaño. • Uso de JMeter como herramienta de carga para comparar el rendimiento del servidor Web. • Las comparaciones de rendimiento muestran que Docker es más rápido que KVM.
CONCEPTOS CLAVES	Contiene conceptos de Docker, KVM, contenedores, máquinas virtuales, y benchmarking. Herramientas: JMeter.
CONCLUSIONES	La comparación del rendimiento muestra que Docker utiliza los recursos de hardware como CPU, HDD, RAM de forma más rápida y eficiente que KVM.
APORTE AL TEMA ELEGIDO	Aporta brindando información detallada acerca del rendimiento que ofrece tanto Docker como KVM. Proporciona tres tipos de comparaciones con sus respectivas gráficas en dónde se puede ver de una forma resumida los puntos fuertes y débiles de cada tecnología, esto brindará un nuevo punto de vista que puede ser utilizado para elegir entre una u otra tecnología. De igual forma da a conocer JMeter, una herramienta que sirve como prueba de carga para analizar

	y medir el rendimiento de una variedad de servicios.

Autor: Anthony López

Tabla 4. Ficha bibliográfica 3

FICHA BIBLIOGRÁFICA	
TEMA	Performance analysis of multi services on container Docker, LXC, and LXD
RESUMEN DEL ARTÍCULO CIENTÍFICO	Esta investigación evalúa el impacto de diferentes plataformas de contenedores (Docker, LXC y LXD) en la ejecución de diferentes servicios TCP y también mide el rendimiento del sistema de cada contenedor en comparación con el sistema nativo sin ninguna solución de contenedor basada en métricas de rendimiento general. Esta se centra en las tres plataformas como servicio (PaaS) más utilizados: Servidor FTP, servidor web y servidor de correo.
PROPÓSITO	Analizar, comparar y obtener métricas del rendimiento de multiservicios en contenedores Docker, LXC y LXD.
IDEAS CENTRALES	• Los tres PaaS más utilizados: Servidor FTP, servidor web y servidor de correo. • Uso de un servidor que se virtualizará mediante un contenedor para ejecutar un determinado servidor de

	aplicaciones (servidor FTP, servidor web y servidor de correo). • Obtener métricas de rendimiento general, que incluyen el rendimiento de la CPU, la velocidad de la RAM y el sistema de archivos IOzone como parámetros. • Medir la capacidad de Apache2 como servidor web en un contenedor para servir peticiones HTTP utilizando Apache Benchmark (ab). • Medir el tiempo total, la latencia y la velocidad de transferencia de la transacción FTP entre el servidor y el cliente. • Evaluación del servidor de correo utilizando postal como herramienta de referencia.
CONCEPTOS CLAVES	Contiene conceptos clave sobre contenedores, Docker, LXC, LXD, PaaS y benchmarking.
CONCLUSIONES	Existe una diferencia notable en el rendimiento del sistema entre los contenedores y el nativo, debido a la sobrecarga que la virtualización genera. Sin embargo, la sobrecarga de rendimiento en los contenedores se considera pequeña y en varios casos casi no existe.

APORTE AL TEMA ELEGIDO	Este artículo científico aporta una gran cantidad de información acerca de temas importantes, como contenedores, Docker, LXC y LXD. A su vez, gracias a las métricas que se obtuvieron, se puede llegar a una conclusión más detallada sobre cuál es el mejor método por elegir para virtualización.

Autor: Anthony López

Tabla 5. Ficha bibliográfica 4

FICHA BIBLIOGRÁFICA	
TEMA	Modelado Funcional de Contenedores Virtuales Docker
RESUMEN DE LA TESIS	Esta tesis se centra en proponer un método para facilitar la manipulación de contenedores Docker. El procedimiento que se propone consta de tres etapas: modelado, representación y consumo. Proporciona un método interesante para modelar una imagen Docker en base a la funcionalidad y mejores tiempos de respuesta. A su vez, expone conceptos interesantes sobre las capas que una imagen Docker maneja y todos los archivos que esta contiene.
PROPÓSITO	Crear una estrategia para modelar correctamente contenedores virtuales utilizando Docker.

IDEAS CENTRALES	• Tareas como la migración, monitoreo, clasificación, entre otras, permiten la manipulación de contenedores para obtener un mejor desempeño de este. • Implementar una API REST permite representar el consumo y las variaciones entre las respuestas en tiempo real del servidor. • Establecer memoria compartida como tipo de comunicación, evitando usar la red y mejorar el rendimiento del contenedor.
CONCEPTOS CLAVES	Contiene conceptos clave sobre la arquitectura de WoT, modelado y manipulación de contenedores Docker y estrategias de evaluación.
CONCLUSIONES	Docker facilita el desarrollo con contenedores volviéndolo rápido y ágil. Estos contenedores están compuestos por piezas de software, las cuáles, mediante una estrategia de modelado pueden ser asignadas correctamente con el fin de ofrecer un empaquetado lógico aún más eficiente.
APORTE AL TEMA ELEGIDO	Esta tesis aporta un nuevo punto de vista sobre los contenedores y su esquema interno. La información que esta investigación aporta es fundamental porque deja en claro que tener una estrategia de modelado para construir

	contenedores es fundamental para trabajar de una forma más eficiente.

Autor: Anthony López

Tabla 6. Ficha bibliográfica 5

FICHA BIBLIOGRÁFICA	
TEMA	Estudio comparativo entre una arquitectura con microservicios y contenedores dockers y una arquitectura tradicional (monolítica) con comprobación aplicativa
RESUMEN DE LA TESIS	Brinda una serie de ventajas y desventajas entre la aplicación de arquitecturas monolíticas y arquitecturas basadas en microservicios en contenedores Docker. Muestra estadísticas recogidas durante la utilización de contenedores Docker con cada una de las arquitecturas mencionadas anteriormente. Contiene gráficas que muestran el desempeño de las distintas aplicaciones utilizando las diferentes arquitecturas aplicadas en la construcción de los contenedores.
PROPÓSITO	Comparar arquitecturas de microservicios en contenedores Docker con una arquitectura tradicional (monolítica).
IDEAS CENTRALES	• La implementación de una arquitectura basada en microservicios

	en contenedores Docker ofrece mejores tiempos de respuesta. • Una arquitectura en base a microservicios en tolerante a fallos. • Implementar microservicios puede ser un punto de unión para utilizar metodologías de desarrollo ágil como Scrum o Extreme.
CONCEPTOS CLAVES	Contiene conceptos clave sobre microservicios, Docker, contenedores, imágenes Docker y metodologías de desarrollo ágil.
CONCLUSIONES	Aplicar una arquitectura que está basada en microservicios ayuda a mejorar el tiempo de respuesta que un contenedor ofrece, además de que esta es tolerante a fallos, lo cual permite que los datos que son solicitados retornen de forma exitosa.
APORTE AL TEMA ELEGIDO	Esta tesis aporta una gran cantidad de información sobre la arquitectura de una imagen Docker, lo que la convierte en un excelente aporte teórico que será de gran utilidad para desarrollar uno de los objetivos específicos del tema elegido.

Tabla 7. Ficha bibliográfica 6

FICHA BIBLIOGRÁFICA	
TEMA	Implementación de un servidor con herramientas DevOps: Implementación de un servidor NGINX en Docker con vhost
RESUMEN DE LA TESIS	Esta tesis proporciona información valiosa sobre cómo y con qué construir una imagen Docker personalizada para implementar un servidor NGINX con VHOST de la forma más optimizada posible. Herramientas como Docker Engine son de gran de utilidad para modelar estas imágenes y por otro lado están los Dockerfiles, que, a partir de imágenes ya existentes, se pueden modificar las mismas y adaptarlas según los requerimientos solicitados.
PROPÓSITO	Implementar un servidor NGINX en Docker con VHOST.
IDEAS CENTRALES	• Docker Engine permite configurar, construir y ejecutar contenedores sobre cualquier sistema operativo. • Docker Engine brinda mecanismos interesantes y fundamentales para la construcción de imágenes en base a Dockerfiles.

	• Existen diferentes métodos detallados para la construcción de imágenes Docker.
CONCEPTOS CLAVES	Contiene conceptos clave sobre DevOps, Docker, virtualización, NGINX, alojamiento virtual Vhost, contenedores e imágenes Docker.
CONCLUSIONES	El uso de un archivo Dockerfile es fundamental para construir imágenes personalizadas en base a imágenes que ya existen, generando adaptabilidad en base a las necesidades y requerimientos del servicio que se desea implantar, logrando obtener los resultados esperados.
APORTE AL TEMA ELEGIDO	Aporta con ideas frescas y muy interesantes sobre cómo se pueden construir imágenes Docker personalizadas. La información detallada y las estadísticas recogidas en esta investigación muestran que crear, modificar y personalizar todas y cada una de las capas de una imagen Docker desemboca en un mejor rendimiento del contenedor.

Autor: Anthony López

Tabla 8. Ficha bibliográfica 7

FICHA BIBLIOGRÁFICA	
TEMA	Entorno de contenedores con emuladores de sistemas embebidos STM32
RESUMEN DEL ARTÍCULO CIENTÍFICO	Esta tesis proporciona información de mucha utilidad sobre las capas que contiene una imagen Docker y como se pueden optimizar estas según los requerimientos.
PROPÓSITO	Crear un entorno de contenedores con emuladores de sistemas embebidos STM32
IDEAS CENTRALES	• Uso de archivos Dockerfile como método de construcción para imágenes Docker. • Proceso de instalación y configuración rápido y sencillo gracias al uso de archivos Dockerfile. • Usar los workflows de Github para que la creación de la imagen sea realizada en los servidores de Github y no en la máquina local.
CONCEPTOS CLAVES	Contiene conceptos claves sobre Docker, emulación, internet de las cosas, Qemu, sistemas embebidos, STM32.
CONCLUSIONES	Los archivos Dockerfile permiten generar imágenes personalizadas en

	donde se especifican los comandos que este utilizará para construir un contenedor optimizado y especializado según los requerimientos especificados. Crear una buena estrategia para la construcción de imágenes Docker es necesario para usar la menor cantidad de capas resultantes que conforman la imagen, lo que se considera como una buena práctica.
APORTE AL TEMA ELEGIDO	Aporta con una idea clara y sólida de cómo utilizar archivos Dockerfile para crear imágenes optimizadas para construir contenedores específicos según las necesidades que se propongan.

Autor: Anthony López

2.2.4. Procesamiento y análisis de datos

Mediante el análisis exhaustivo de la información recolectada de temas similares al planteado, se puede concluir que:

- En la actualidad, Docker ofrece numerosas ventajas sobre las máquinas virtuales en el campo de la virtualización.
- Una investigación exhaustiva sobre la construcción de una imagen Docker proporciona un conocimiento detallado de cómo configurar y optimizar cada una de las capas de la imagen para obtener el máximo rendimiento posible del contenedor.
- Realizar una investigación exhaustiva sobre el archivo Dockerfile para poder encontrar y comprender cómo modificarlo de manera eficiente. Al modificar este archivo, se pueden crear imágenes personalizadas que satisfagan

necesidades específicas, como el aumento del rendimiento o la optimización de recursos.

- Cuando se busca comparar imágenes Docker, se debe considerar que este proceso puede requerir una cantidad significativa de recursos de hardware y software para llevarse a cabo de manera efectiva.
- Es fundamental buscar y utilizar software especializado que permita ejecutar contenedores de manera eficiente y confiable, asegurando que los resultados de las comparaciones sean precisos y consistentes. Esto puede incluir herramientas de automatización de contenedores, plataformas de gestión de clústeres y monitoreo de recursos.
- Tener en cuenta que para comparar el rendimiento entre imágenes Docker de manera precisa y objetiva, se deben realizar pruebas de benchmarking.
- Para llevar a cabo la implementación del proceso de benchmarking, es necesario realizar una selección rigurosa entre diversas herramientas y metodologías disponibles, con el fin de determinar cuál es la más adecuada para cumplir con los objetivos y necesidades específicas planteadas en el proceso.
- Implementar la imagen Docker optimizada y realizar pruebas exhaustivas para demostrar que se han cumplido los objetivos definidos.

CAPÍTULO III.- RESULTADOS Y DISCUSIÓN

3.1. Análisis y discusión

3.1.1. Análisis de la construcción de una imagen Docker

La construcción de una imagen Docker es un proceso fundamental en el desarrollo y despliegue de aplicaciones basadas en contenedores. En este contexto, se analiza detalladamente el proceso de construcción de imágenes Docker, el cual consiste en la definición y configuración de un entorno virtualizado que encapsula todas las dependencias y configuraciones necesarias para ejecutar una aplicación de manera independiente.

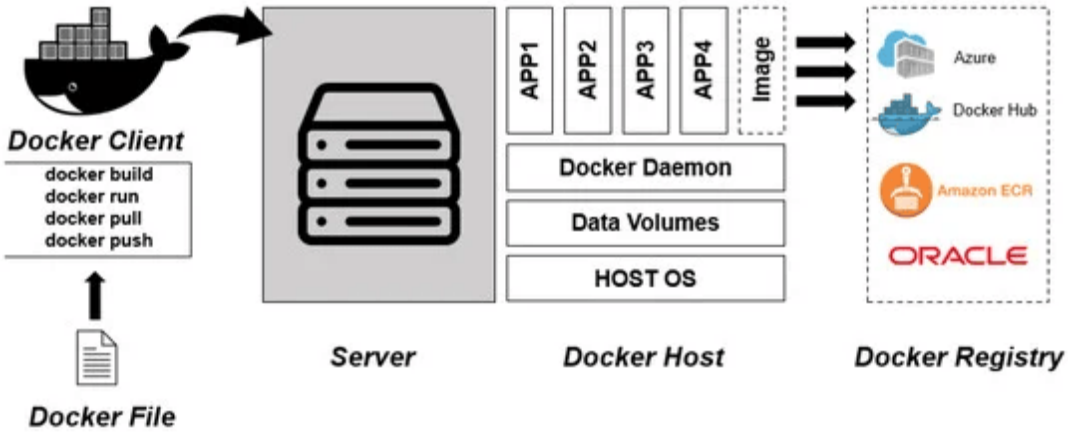


Figura 1. Arquitectura Docker [20]

Tabla 9. Análisis de la estructura de una imagen Docker

ESTRUCTURA DE UNA IMAGEN DOCKER	
Una imagen Docker es una instantánea de un sistema de archivos y consta de varias capas, que se combinan para formar una imagen completa. Cada capa representa un cambio en el sistema de archivos y es inmutable. Esto significa que una vez que se crea una capa, no se puede modificar.	
Metadatos	Los metadatos de una imagen Docker incluyen información sobre la imagen en sí, como su identificador único (ID), nombre, etiquetas, descripción y autor. Estos metadatos se almacenan en un archivo llamado manifest.json, que se encuentra en el directorio /var/lib/docker/image/ en el sistema de archivos del host de Docker.
Capas	Las capas son la parte principal de una imagen Docker y se componen de un conjunto de archivos y directorios que se combinan para formar el sistema de archivos de la imagen. Cada capa es inmutable y se almacena en el sistema de archivos del host de Docker en el directorio /var/lib/docker/overlay2/.
Cada capa en una imagen Docker se identifica por su ID de capa, que es un valor hash generado por una función criptográfica a partir del contenido de la capa. Esto significa que, si dos capas tienen el mismo contenido, tendrán el mismo ID de capa. Cada capa en una imagen Docker es creada por una instrucción en el Dockerfile. Por ejemplo, si la instrucción RUN se utiliza para ejecutar un comando en la imagen, se creará una nueva capa para esa instrucción.	

DIFF	Una carpeta llamada "diff" que contiene los archivos y directorios que se agregan o modifican en la capa.
JSON	Un archivo llamado "json" que contiene información sobre la capa, como su ID de capa y los ID de las capas padre.
VERSION	Un archivo llamado "VERSION" que indica la versión de la estructura de capas que se utiliza.

Autor: Anthony López

Tabla 10. Análisis de la arquitectura Docker

ARQUITECTURA DOCKER	
Docker es una plataforma de contenedores de software que permite la creación, el despliegue y la ejecución de aplicaciones en contenedores. Docker se basa en una arquitectura cliente-servidor que consta de varios componentes clave.	
Cliente Docker	El cliente Docker es la interfaz de línea de comandos (CLI) que se utiliza para interactuar con el demonio de Docker y ejecutar comandos en el sistema de host. El cliente Docker se conecta al demonio de Docker a través de una API RESTful.
Demonio Docker	El demonio Docker es el componente central de la arquitectura de Docker. Es responsable de gestionar los contenedores, las imágenes, las redes y los volúmenes de almacenamiento. El demonio Docker se ejecuta en el sistema

	de host y se comunica con el cliente Docker a través de una API RESTful.
Imágenes Docker	Una imagen Docker es un paquete de software ligero y portátil que contiene todo lo necesario para ejecutar una aplicación, incluyendo el código fuente, las bibliotecas y las dependencias. Las imágenes Docker se construyen a partir de un archivo llamado Dockerfile que contiene las instrucciones necesarias para crear la imagen.
Contenedores Docker	Un contenedor Docker es una instancia en ejecución de una imagen Docker. Los contenedores Docker son aislados del sistema host y de otros contenedores mediante el uso de namespaces y cgroups del kernel de Linux. Los contenedores Docker proporcionan un entorno seguro y consistente para la ejecución de aplicaciones.
Redes Docker	Las redes Docker proporcionan una forma de conectar los contenedores y permiten la comunicación entre ellos. Docker ofrece varios tipos de redes, como la red bridge, la red host y la red overlay, cada una de las cuales tiene sus propias características y casos de uso.
Volúmenes Docker	Los volúmenes Docker proporcionan un método para persistir los datos del contenedor. Los volúmenes son independientes del ciclo de vida del

	contenedor y pueden ser compartidos entre varios contenedores. Los volúmenes se pueden utilizar para almacenar datos de aplicaciones, bases de datos y otros archivos importantes.
Docker Hub	Docker Hub es un repositorio centralizado de imágenes Docker públicas y privadas. Los usuarios pueden buscar, descargar y compartir imágenes Docker en Docker Hub. Docker Hub también ofrece herramientas para la gestión de imágenes y la integración con sistemas de automatización de construcción y despliegue.

Autor: Anthony López

Tabla 11. Análisis de las capas de una imagen Docker

CAPAS DE UNA IMAGEN DOCKER	
Una imagen Docker está compuesta por una serie de capas, cada una de las cuales es una instantánea de un sistema de archivos. Cada capa representa un cambio en el sistema de archivos y se combina con las capas anteriores para formar la imagen completa. Las capas de una imagen Docker se crean a través de las instrucciones de Dockerfile, que se utilizan para construir la imagen. Cada instrucción crea una capa nueva que se agrega a las capas existentes. Las capas son inmutables, lo que significa que una vez que se crea una capa, no se puede modificar. Si se necesita realizar cambios en una capa existente, se debe crear una nueva capa que incluya esos cambios.	
Carpeta "diff"	La carpeta "diff" es la parte principal de una capa de imagen Docker y contiene los archivos y directorios que se agregan

	o modifican en la capa. Los archivos y directorios de la capa anterior se heredan automáticamente. La carpeta "diff" es la que se utiliza para crear la imagen de contenedor.
Archivo "json"	El archivo "json" es un archivo de metadatos que contiene información sobre la capa, como su ID de capa, el ID de las capas anteriores y la información de configuración. Este archivo se utiliza para rastrear las relaciones entre las capas y para recuperar información sobre una capa en particular.
Archivo "VERSION"	El archivo "VERSION" indica la versión de la estructura de capas que se utiliza. La versión actual es la 2.0, que se utiliza para todas las versiones de Docker a partir de la versión 1.10.
Cada capa se identifica por su ID de capa, que es un valor hash generado por una función criptográfica a partir del contenido de la capa. Si dos capas tienen el mismo contenido, tendrán el mismo ID de capa. Esto significa que las capas se pueden compartir entre imágenes, lo que reduce el tamaño total de la imagen.	
La construcción de una imagen Docker implica crear varias capas. Por ejemplo, una instrucción RUN para instalar un paquete de software creará una nueva capa que incluye ese paquete instalado en el sistema de archivos. Una instrucción COPY para copiar archivos en la imagen creará otra capa que incluya esos archivos.	
Las capas son una parte fundamental de la arquitectura de Docker y permiten la construcción eficiente de imágenes. Al utilizar capas, se pueden compartir capas comunes entre imágenes, lo que reduce el tamaño total de la imagen. Además, las	

capas inmutables hacen que las imágenes sean más seguras y fáciles de administrar y actualizar.

Autor: Anthony López

Tabla 12. Análisis del archivo Dockerfile

DOCKERFILE	
El Dockerfile es un archivo de texto que contiene una lista de instrucciones que Docker utiliza para construir una imagen Docker. El Dockerfile describe cómo debe construirse la imagen y qué debe incluir. A continuación, se describen las principales instrucciones que se utilizan en un Dockerfile:	
FROM	La instrucción FROM especifica la imagen base que se utilizará como punto de partida para la construcción de la nueva imagen. Por ejemplo, FROM ubuntu indica que se utilizará la imagen oficial de Ubuntu como base.
RUN	La instrucción RUN ejecuta comandos en el sistema de archivos de la imagen durante el proceso de construcción. Por ejemplo, RUN apt-get update && apt-get install -y apache2 instala el servidor web Apache en la imagen.
COPY	La instrucción COPY copia archivos desde el sistema de archivos del host a la imagen. Por ejemplo, COPY ./app /app copia el directorio "app" del host a la imagen.
WORKDIR	La instrucción WORKDIR especifica el directorio de trabajo para las

	instrucciones RUN, CMD, ENTRYPOINT, COPY y ADD. Por ejemplo, WORKDIR /app establece el directorio de trabajo en "/app".
CMD	La instrucción CMD especifica el comando que se ejecutará cuando se inicie un contenedor basado en la imagen. Si se especifican varios comandos, sólo se ejecutará el último. Por ejemplo, CMD ["python", "app.py"] especifica que se debe ejecutar el script "app.py" con Python cuando se inicie el contenedor.
EXPOSE	La instrucción EXPOSE especifica los puertos en los que se escuchará cuando se inicie un contenedor basado en la imagen. Por ejemplo, EXPOSE 80 expone el puerto 80 de la imagen.
ENV	La instrucción ENV establece variables de entorno en la imagen. Por ejemplo, ENV MY_VAR="value" establece la variable de entorno "MY_VAR" en "value".
ARG	La instrucción ARG define argumentos que se pueden pasar a la imagen durante el proceso de construcción. Por ejemplo, ARG version=latest define el argumento "version" con un valor predeterminado de "latest".
En resumen, el Dockerfile es un archivo de texto que describe cómo debe construirse una imagen Docker utilizando una lista de instrucciones, que incluyen la imagen	

base, las operaciones a realizar durante la construcción, los comandos a ejecutar y las variables de entorno y argumentos que se deben establecer. Con el uso del Dockerfile, la construcción de imágenes Docker se puede automatizar y replicar fácilmente en diferentes entornos de implementación.

Autor: Anthony López

3.1.2. Metodología de Desarrollo

Es fundamental adoptar una metodología de desarrollo durante el proceso de desarrollo de un sistema, ya que brinda estructura y eficiencia al equipo. Además, facilita la definición de objetivos claros y el enfoque en su consecución. Además, contribuye a garantizar la calidad del software y prevenir errores costosos y problemas de mantenimiento a largo plazo.

Existen muchas metodologías de desarrollo de software disponibles, cada una con sus propias características y ventajas. Algunas de las metodologías más populares incluyen Extreme Programming (XP), Scrum, Kanban, Lean Development y Agile Unified Process (AUP). Cada una de estas metodologías tiene un enfoque diferente en cuanto a la planificación, diseño, implementación, pruebas y mantenimiento de sistemas de software.

Extreme Programming

Es una metodología ágil de desarrollo de software que se centra en la calidad del producto y la satisfacción del cliente. XP se enfoca en la programación en parejas, pruebas continuas y entrega frecuente de software funcional. Uno de los aspectos más distintivos de XP es la comunicación y colaboración constante entre los miembros del equipo, incluyendo al cliente. XP también hace hincapié en la simplicidad en la programación y en el proceso de desarrollo en sí, minimizando la cantidad de documentación y procesos innecesarios.

Otro elemento clave de XP es la adaptabilidad, lo que significa que la metodología se ajusta al proyecto y al equipo en lugar de ser una metodología prescriptiva. Los miembros del equipo trabajan juntos en ciclos cortos de desarrollo llamados

iteraciones, y la retroalimentación es constante. La metodología XP también incluye prácticas como la refactorización de código, la integración continua y la planificación de la versión, que ayudan a mejorar la calidad del software y a mantener el enfoque en los objetivos del proyecto.

Scrum

Scrum es una metodología ágil de gestión de proyectos de software que se centra en la entrega iterativa e incremental del producto final. Se divide en ciclos llamados sprints, que suelen durar entre una y cuatro semanas, y al final de cada sprint se entrega un incremento del producto. El proceso se divide en tres roles: el dueño del producto, el equipo de desarrollo y el Scrum Master. El dueño del producto es responsable de definir y priorizar las funcionalidades del producto, el equipo de desarrollo es responsable de desarrollar el software y el Scrum Master es responsable de facilitar el proceso y asegurar que se sigan las prácticas de Scrum.

En Scrum, se utilizan diferentes artefactos para gestionar el proyecto: el backlog del producto, que es una lista priorizada de todas las funcionalidades que deben ser desarrolladas; el backlog del sprint, que es una lista de las funcionalidades que se van a desarrollar durante el sprint actual; y el incremento del producto, que es el resultado del sprint actual y contiene las funcionalidades completadas. Además, se utilizan diferentes eventos para facilitar la gestión del proyecto, como la planificación del sprint, la reunión diaria de Scrum, la revisión del sprint y la retrospectiva del sprint.

Scrum tiene varios beneficios, como la capacidad de adaptarse a los cambios en los requisitos del proyecto y la mejora en la colaboración entre los miembros del equipo. Al enfocarse en la entrega de funcionalidades completas en cada sprint, Scrum también permite a los clientes y dueños del producto ver rápidamente el progreso del proyecto y proporcionar retroalimentación en consecuencia.

Kanban

La metodología Kanban se centra en la gestión visual del flujo de trabajo para mejorar la eficiencia y la productividad. Se utiliza un tablero Kanban que se divide en columnas que representan diferentes etapas del proceso y se mueven las tareas de una columna

a otra según su estado. El objetivo es limitar el trabajo en progreso y optimizar el flujo de trabajo para que el equipo pueda trabajar de manera más efectiva. Kanban también se enfoca en la mejora continua a través de la retroalimentación del equipo y el análisis de los datos de rendimiento.

Una de las características clave de Kanban es su adaptabilidad a diferentes situaciones y proyectos. No tiene roles predefinidos como Scrum, por lo que el equipo puede adaptar el proceso según sus necesidades y responsabilidades. Además, Kanban se enfoca en el trabajo que está en curso y no en el trabajo futuro, lo que lo hace útil para equipos que trabajan en proyectos de larga duración o proyectos que cambian frecuentemente.

Lean Development

La metodología Lean Development, también conocida como desarrollo ágil lean, busca eliminar los desperdicios en el proceso de desarrollo de software y aumentar el valor entregado al cliente. Esta metodología se enfoca en la colaboración entre el equipo de desarrollo y el cliente, y en la entrega de pequeñas mejoras de manera rápida y constante.

Para lograr esto, se utilizan principios como la optimización del flujo de trabajo, la mejora continua y la eliminación de actividades que no agregan valor. Se promueve la automatización de tareas repetitivas, la visualización del proceso de trabajo y la identificación temprana de problemas para solucionarlos de forma rápida.

Agile Unified Process (AUP)

Agile Unified Process (AUP) es una metodología de desarrollo de software ágil y escalable, que combina principios de las metodologías ágiles como Scrum y XP con técnicas y conceptos del Proceso Unificado de Rational (RUP). El enfoque de AUP es el desarrollo iterativo e incremental, la colaboración entre equipos y la entrega continua de software de alta calidad. La metodología se enfoca en la creación de software que satisfaga las necesidades de los usuarios, a través de una comunicación constante y efectiva con los stakeholders.

AUP se basa en la idea de que cada equipo de desarrollo debe adaptar la metodología a las necesidades específicas del proyecto, permitiendo una mayor flexibilidad y capacidad de respuesta a los cambios y requerimientos del cliente. La metodología AUP se compone de cuatro fases principales: Inicio, Elaboración, Construcción y Transición, cada una de las cuales se enfoca en objetivos específicos del proyecto. AUP también hace hincapié en la gestión del riesgo y la gestión del cambio, fomentando una cultura de mejora continua en el proceso de desarrollo de software.

Evaluación de las metodologías

Tabla 13. Comparación entre metodologías

Metodología	Características Técnicas	Enfoque	Principales Prácticas
Extreme Programming (XP)	Desarrollo basado en pruebas, integración continua, refactoring.	Calidad del software.	Desarrollo basado en pruebas, refactoring, integración continua, programación en pareja.
Scrum	Desarrollo iterativo e incremental, roles definidos.	Entrega de valor.	Sprints, reuniones diarias, product backlog, sprint backlog, incremento de producto.
Kanban	Flujo de trabajo visual, límites de trabajo en progreso.	Eficiencia y fluidez.	Gestión visual del flujo de trabajo, límites de trabajo en progreso, mejora continua.
Lean Development	Eliminación de desperdicio, entrega rápida de valor.	Reducción del desperdicio.	Supresión de desperdicio, entrega rápida de valor, mejora continua.
Agile Unified Process	Ciclos de vida iterativos, desarrollo	Adaptabilidad y calidad.	Ciclos de vida iterativos, desarrollo guiado por

	guiado por casos de uso.		casos de uso, modelado ágil.

Autor: Anthony López

Después de un análisis y evaluación de las diferentes metodologías disponibles, se determinó que Agile Unified Process (AUP) era la opción más adecuada para el desarrollo de este proyecto de investigación. Esta metodología fue seleccionada debido a su enfoque en ciclos de vida iterativos, desarrollo guiado por casos de uso y su capacidad para adaptarse a los cambios y garantizar la calidad del producto final. Además, prácticas como el modelado ágil y la integración continua, ofrecen una sólida base para el desarrollo eficiente y efectivo del proyecto. La elección de AUP refleja la importancia de adaptabilidad, calidad y entrega de valor en el contexto de esta investigación, lo que permitirá alcanzar los objetivos de manera eficaz y satisfactoria.

a. Implementación de Agile Unified Process (AUP)

Fase de inicio:

- Definir los objetivos generales y específicos del proyecto de investigación.
- Identificar los roles necesarios para llevar a cabo el proyecto de investigación, como el investigador, el supervisor y otros colaboradores.
- Establecer una visión clara del alcance de la investigación y los entregables esperados.

Fase de elaboración:

- Realizar un análisis del tema de investigación, incluyendo una revisión bibliográfica y la identificación de las áreas clave de estudio.
- Definir el marco teórico y conceptual de la investigación, identificando las metodologías y herramientas que se utilizarán.
- Establecer un plan detallado de trabajo, definiendo las actividades, tareas y plazos.

Fase de construcción:

- Implementar las estrategias y metodologías seleccionadas para optimizar el desarrollo con contenedores y construir imágenes Docker eficientes.
- Realizar pruebas y experimentos para evaluar el rendimiento de los contenedores generados, utilizando técnicas de benchmarking y recolectando datos cuantitativos.
- Aplicar análisis y técnicas de optimización sobre la construcción de imágenes Docker, con el objetivo de mejorar la eficiencia y rendimiento de los contenedores.

Fase de transición:

- Documentar los hallazgos obtenidos durante la investigación, incluyendo los resultados de las pruebas de rendimiento, las estrategias de optimización aplicadas y las conclusiones obtenidas.

3.1.3. Método de benchmarking

El proceso de benchmarking se puede dividir en varias etapas, que incluyen la identificación de los líderes de la industria, la recolección de datos, la evaluación del rendimiento, la identificación de las áreas de mejora, el desarrollo de estrategias de mejora y la implementación de cambios. Para garantizar que el proceso sea efectivo, es importante asegurarse de que se estén midiendo las mismas métricas que los líderes de la industria y que se estén utilizando los mismos métodos de recolección de datos.

Tabla 14. Comparación entre tipos de benchmarking

Tipos de benchmarking	
Benchmarking interno	Se enfoca en comparar procesos o resultados dentro de una misma organización. Se utiliza para identificar oportunidades de mejora y buenas

	prácticas que puedan ser replicadas en otros departamentos o áreas.
Benchmarking competitivo	Se enfoca en comparar los procesos o resultados de la organización con los de su competencia directa. Este tipo de benchmarking puede ayudar a la organización a identificar áreas en las que su competencia es más fuerte, y a encontrar oportunidades para mejorar y ser más competitivos.
Benchmarking funcional	Se enfoca en comparar procesos o resultados de una función específica dentro de una organización, como, por ejemplo, el departamento de recursos humanos o el área de finanzas. Este tipo de benchmarking puede ayudar a identificar oportunidades de mejora y buenas prácticas dentro de la función en cuestión.
Benchmarking externo	Se enfoca en comparar procesos o resultados de una organización con los de otra organización en un sector diferente. Este tipo de benchmarking puede ayudar a la organización a encontrar nuevas ideas y oportunidades de mejora, y a identificar áreas donde se puede ser más innovador.
Benchmarking informático	Se enfoca en comparar el rendimiento, velocidad, capacidad y otros aspectos técnicos de los sistemas informáticos de la organización con los de otras

	organizaciones similares. Este tipo de benchmarking puede ayudar a la organización a identificar oportunidades de mejora en su infraestructura tecnológica y a optimizar su rendimiento.

Autor: Anthony López

Con la información que se obtuvo a partir de los benchmarking señalados anteriormente se ha decidido utilizar benchmarking informático debido a que es una metodología que permite comparar el desempeño y eficiencia de distintas soluciones tecnológicas. El benchmarking informático se enfoca en la medición y comparación de los resultados de un conjunto de pruebas realizadas a distintas soluciones tecnológicas, y puede ayudar a identificar las mejores prácticas y estrategias para mejorar el desempeño y eficiencia.

3.1.4. Herramienta de benchmarking

Algunas de las características que se pueden medir con las herramientas de benchmarking para Linux incluyen el tiempo de arranque del sistema, la velocidad de procesamiento de datos, la tasa de transferencia de datos, la utilización de la memoria RAM y la eficiencia del sistema de archivos. Los resultados obtenidos mediante estas herramientas pueden ser útiles para optimizar el rendimiento del sistema, identificar cuellos de botella y realizar comparaciones con otros sistemas similares.

Tabla 15. Comparación herramientas de benchmarking

Herramienta	Características Técnicas	Compatibilidad
Phoronix Test Suite	Realiza pruebas de rendimiento y benchmarking en diversas áreas del sistema.	Ubuntu, Debian, Fedora, CentOS, Arch Linux, openSUSE, etc.
Stacer	Monitorea el rendimiento del sistema, incluyendo el uso de CPU, memoria y almacenamiento.	Ubuntu, Debian, Fedora, openSUSE, Arch Linux, etc.
Stress-ng	Realiza pruebas de estrés en el sistema, generando cargas de trabajo intensivas.	Ubuntu, Debian, Fedora, CentOS, Arch Linux, openSUSE, etc.
Siege	- Realiza pruebas de carga en servidores web para evaluar su rendimiento.	Ubuntu, Debian, Fedora, CentOS, Arch Linux, openSUSE, etc.
Bonnie++	Realiza pruebas de rendimiento del sistema de archivos.	Ubuntu, Debian, Fedora, CentOS, Arch Linux, openSUSE, etc.
Netperf	Mide el rendimiento de la red, incluyendo latencia y ancho de banda.	Ubuntu, Debian, Fedora, CentOS, Arch Linux, openSUSE, etc.
Docker Stats	Monitorea estadísticas de rendimiento de contenedores Docker.	Ubuntu, Debian, Fedora, CentOS, Arch Linux, openSUSE, etc.

Autor: Anthony López

En base a los datos recolectados en la tabla 16, se tomó la decisión de utilizar tres herramientas específicas. En primer lugar, Stacer ha sido seleccionada debido a su capacidad para medir una amplia gama de recursos del sistema operativo, lo que

permitirá obtener una visión completa del rendimiento general. En segundo lugar, Siege se ha elegido por su eficacia en la evaluación del rendimiento de servidores web, lo que resulta crucial para garantizar un funcionamiento óptimo. Por último, Docker Stats se utilizará para medir las estadísticas de rendimiento de los contenedores, lo que proporcionará información valiosa sobre su desempeño. Estas herramientas combinadas permitirán obtener una visión integral del sistema y tomar medidas adecuadas para mejorar su rendimiento.

3.1.5. Distribución GNU/Linux

Las distros de GNU/Linux también son conocidas por su seguridad y estabilidad, lo que las hace populares entre los usuarios que valoran la privacidad y la seguridad. Además, las distros de GNU/Linux suelen ser gratuitas y no requieren licencias costosas, lo que las hace atractivas para los usuarios que buscan alternativas económicas a los sistemas operativos propietarios.

Tabla 16. Comparación entre distros Linux

Distro	Base	Kernel	Arquitectura	Gestor paquetes	Licencia
Ubuntu	Debian	Linux	x86, x86_64, ARM	APT	GPL
Debian	-	Linux	x86, x86_64, ARM	APT	GPL
Fedora	Red Hat	Linux	x86, x86_64, ARM	DNF	GPL
CentOS	Red Hat	Linux	x86, x86_64, ARM	YUM	GPL
Arch Linux	Independiente	Linux	x86, x86_64, ARM	Pacman	GPL
AlmaLinux	Red Hat	Linux	x86, x86_64	YUM	GPL

Autor: Anthony López

Después de analizar los datos recolectados en la tabla comparativa entre las distros de GNU/Linux, se decidió utilizar Fedora como la distro a utilizar. Esto se debe a que Fedora es una distro intuitiva y fácil de usar, que ofrece una gran cantidad de

documentación y una comunidad muy sólida, lo que hace que sea una excelente opción para aquellos que están empezando a utilizar Linux. Además, Fedora cuenta con una amplia variedad de herramientas y aplicaciones, y su gestor de paquetes es muy eficiente y fácil de usar. Otras características que hacen que Fedora sea una buena elección son su enfoque en la innovación y la actualización constante de su software, así como su gran soporte para arquitecturas de 64 bits.

3.1.6. Instalación de Docker en Linux

Docker es una plataforma de contenedores que permite empaquetar, distribuir y ejecutar aplicaciones en un entorno aislado. Es sencillo de instalar en la mayoría de las distribuciones de Linux, incluyendo Fedora.

A continuación, se proporciona una guía detallada para instalar Docker y Docker Compose en Fedora:

1) Actualizar los paquetes del sistema:

   sudo dnf update

2) Instalar los paquetes necesarios para agregar los repositorios de Docker:

   sudo dnf install dnf-plugins-core

3) Agregar el repositorio de Docker:

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

4) Instalar Docker:

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

5) Iniciar el servicio de Docker:

   sudo systemctl start Docker

6) Verificar que Docker se ha instalado correctamente y está en ejecución:

sudo docker run hello-world

Una vez completada la instalación exitosa de Docker, el usuario podrá observar un mensaje de bienvenida en la pantalla. Este mensaje confirma que Docker se ha instalado correctamente y está listo para ser utilizado.

7) Instalar Docker Compose:

sudo dnf install docker-compose

8) Verificar que Docker Compose se ha instalado correctamente:

docker-compose versión

Al ejecutar el comando correspondiente, se mostrará información detallada sobre la versión de Docker Compose instalada en el sistema, lo que permitirá al usuario verificar y confirmar la correcta instalación del software.

Con los pasos mencionados, el usuario logrará instalar Docker y Docker Compose en Fedora de manera exitosa, obteniendo así un entorno de contenedores listo para su uso.

3.1.7. Entorno de producción para pruebas de contenedores

Para crear el entorno de producción, se han seleccionado dos contenedores Docker clave: bitnami/moodle y node.js.

Se seleccionó bitnami/moodle para la creación de un entorno de servidor específico destinado a realizar pruebas de estrés en la página principal de Moodle. Esta selección se basa en la relevancia de Moodle como plataforma de aprendizaje y su amplia adopción en entornos educativos. Al utilizar el contenedor bitnami/moodle, se aprovecha una imagen preconfigurada y optimizada para ejecutar Moodle de manera eficiente y fiable. Esto permite concentrarse en la evaluación del rendimiento y la capacidad de respuesta de Moodle bajo diferentes cargas de trabajo, obteniendo así información valiosa para la optimización de la plataforma.

Se escogió node.js para crear un entorno de desarrollo en el cual se instalará la herramienta Nativefier. La elección de este contenedor se basa en la popularidad y

versatilidad de node.js como entorno de ejecución para aplicaciones web. La herramienta Nativefier permitirá convertir una aplicación web en una aplicación de escritorio, lo que brinda la posibilidad de evaluar el rendimiento y la eficiencia del desarrollo de aplicaciones utilizando contenedores.

3.1.8. Instalación y configuración bitnami/moodle

Bitnami Moodle es una solución lista para usar que simplifica la instalación y configuración de Moodle. Proporciona una instalación preconfigurada de Moodle con todos los componentes necesarios, lo que facilita su implementación y permite a los educadores centrarse en la creación y gestión de cursos en línea.

A continuación, se proporciona una pequeña guía detallada para instalar y usar bitnami/moodle:

1) Obtener la imagen

   La forma recomendada de obtener la imagen Docker de Bitnami para Moodle es extraer la imagen precompilada del Docker Hub Registry.

   docker pull bitnami/moodle:latest

2) Ejecutar la aplicación utilizando Docker Compose

   curl -sSL https://raw.githubusercontent.com/bitnami/containers/main/bitnami/moodle/docker-compose.yml > docker-compose.yml

   docker-compose up -d

3) Acceder a la aplicación

   http://localhost/

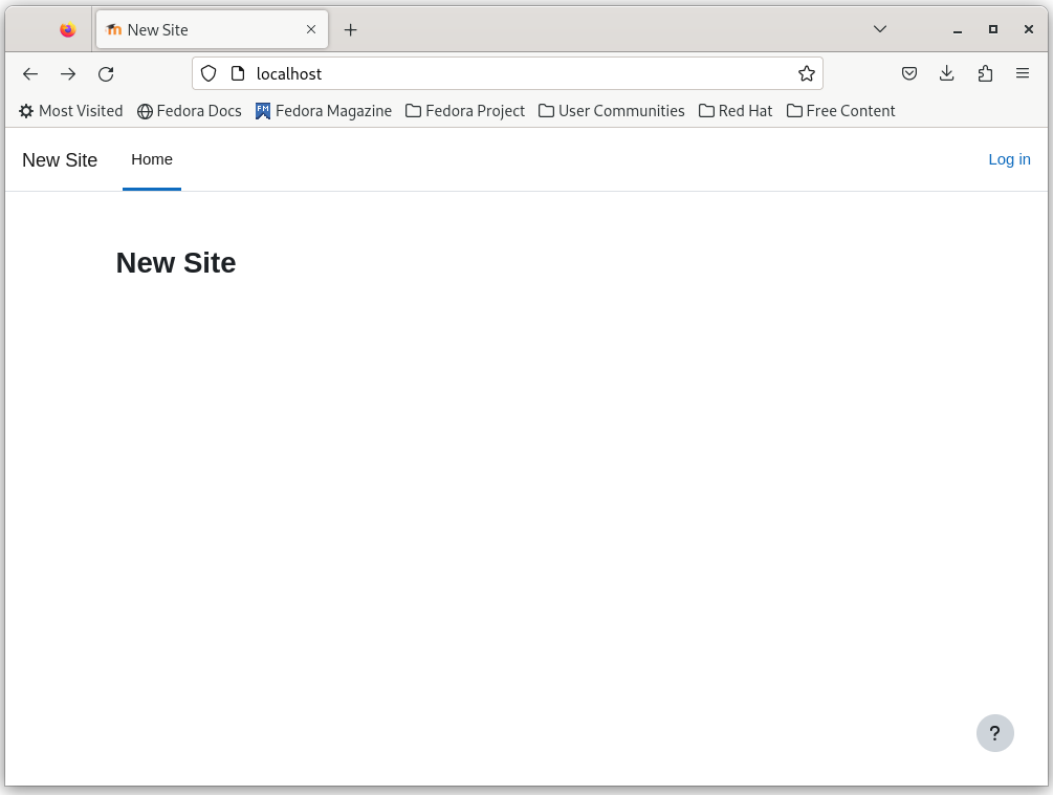


Figura 2. Contenedor bitnami/moodle iniciado

3.1.9. Implementación de benchmarking sobre bitnami/moodle

En la sección de Implementación de benchmarking en bitnami/moodle, se utilizan varias herramientas para evaluar el rendimiento del contenedor y obtener estadísticas relevantes. Una de estas herramientas es Siege, que se utiliza para realizar pruebas de carga simulando un alto número de usuarios en el contenedor de Moodle. Esto permite medir la capacidad de respuesta y el rendimiento del contenedor bajo condiciones de estrés.

Además, se utiliza Docker Stats, una utilidad de Docker que proporciona información en tiempo real sobre el uso de recursos del contenedor. Con Docker Stats, se obtienen estadísticas detalladas sobre el consumo de CPU, memoria, uso de disco y red del contenedor mientras se llevan a cabo las pruebas con Siege. Esto ayuda a identificar posibles cuellos de botella o limitaciones de recursos durante la ejecución de Moodle.

Por último, se emplea Stacer, una herramienta de monitoreo y optimización de sistemas, para evaluar el rendimiento de Fedora mientras el contenedor de Moodle está en funcionamiento. Stacer brinda información detallada sobre el uso de recursos del sistema, incluyendo CPU, memoria, disco y red. De esta manera, se puede evaluar el impacto del contenedor en el rendimiento general del sistema operativo y realizar ajustes si es necesario.

A través del uso combinado de estas herramientas, se obtiene una visión completa del rendimiento del contenedor de Moodle, tanto en términos de la carga de usuarios simulada con Siege, como en el uso de recursos del contenedor y del sistema operativo mediante docker stats y Stacer, respectivamente. Esta evaluación permite identificar posibles mejoras y optimizaciones en el entorno de implementación de bitnami/moodle.

a. Monitoreo del sistema con Stacer

En esta sección, se lleva a cabo el monitoreo del sistema utilizando la herramienta Stacer mientras el contenedor de bitnami/moodle se encuentra en funcionamiento. El objetivo principal es obtener una visión detallada del rendimiento y la utilización de recursos del sistema en tiempo real, sin realizar aún pruebas de estrés específicas en el contenedor.

El monitoreo continuo del sistema con Stacer durante la operación normal del contenedor proporcionará una base sólida de datos para evaluar el rendimiento del sistema y detectar posibles problemas antes de realizar las pruebas de estrés. Esto permitirá tomar medidas preventivas o de optimización, si es necesario, para garantizar un entorno estable y eficiente para el despliegue de bitnami/moodle.

A continuación, se proporciona una pequeña guía detallada para instalar Stacer:

Paso 1: Descargar el paquete RPM de Stacer

1) Abrir un navegador web e ir al sitio oficial de Stacer en GitHub: https://github.com/oguzhaninan/Stacer/releases.
2) En la sección de "Downloads", buscar la última versión de Stacer para Fedora.

3) Hacer clic en el enlace del archivo RPM para descargarlo a su sistema.

Paso 2: Instalar Stacer

1) Abrir una terminal en Fedora.
2) Navegar hasta la ubicación donde se descargó el archivo RPM de Stacer. Por ejemplo, si se encuentra en la carpeta de Descargas, ejecutar el siguiente comando:

   cd ~/Descargas

3) Una vez en la ubicación correcta, utilizar el comando dnf para instalar Stacer. Ejecutar el siguiente comando:

   sudo dnf install ./stacer-<versión>.rpm

   Reemplazar <versión> con el número de versión del archivo RPM que se descargó.

4) El administrador de paquetes dnf instalará Stacer y sus dependencias automáticamente en su sistema.

Paso 3: Ejecutar Stacer

1) Una vez que la instalación se haya completado correctamente, puede iniciar Stacer desde la terminal ejecutando el siguiente comando:

   Stacer

   Stacer se abrirá en una nueva ventana con una interfaz gráfica intuitiva.

Figura 3. Herramienta de monitoreo "Stacer" en ejecución

b. Monitoreo de bitnami/moodle con docker stats

En esta sección, se lleva a cabo el monitoreo del contenedor de bitnami/moodle utilizando la herramienta "docker stats" mientras el contenedor se encuentra en ejecución, sin realizar aún pruebas de estrés específicas. El objetivo principal es obtener información actualizada sobre las estadísticas del contenedor y su rendimiento en términos de uso de CPU, memoria, red y otros recursos.

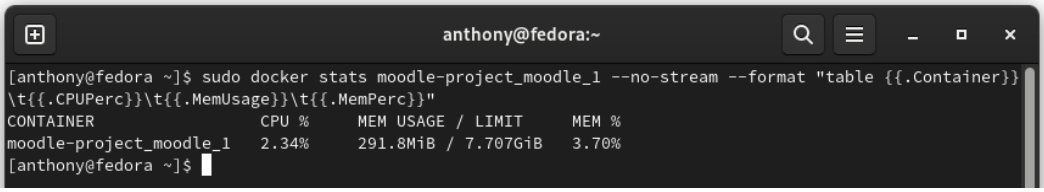


Figura 4. Monitoreo del contenedor utilizando "docker stats"

El código mostrado en la figura 4 realiza lo siguiente:

- *sudo*: Este comando se utiliza para ejecutar la siguiente instrucción con privilegios de superusuario. Permite acceder y realizar operaciones en el entorno Docker.
- *docker stats*: Es un comando de Docker que se utiliza para obtener estadísticas en tiempo real del uso de recursos de los contenedores en ejecución.
- *moodle-project_moodle_1*: Es el nombre específico del contenedor Docker del cual se desea obtener información. Es importante destacar que este nombre puede variar según el contexto y la configuración del entorno.
- *--no-stream*: Es una opción del comando docker stats que desactiva la actualización en tiempo real de las estadísticas. En este caso, se obtiene un único conjunto de datos sin la actualización continua.
- *--format*: Es otra opción del comando docker stats que permite personalizar el formato de salida de las estadísticas.
- *"table {{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.MemPerc}}"*: En este caso, se utiliza un formato de tabla con campos específicos. Los campos incluidos son:
    a. *{{.Container}}*: Representa el nombre o ID del contenedor.
    b. *{{.CPUPerc}}*: Indica el porcentaje de uso de la CPU por parte del contenedor.
    c. *{{.MemUsage}}*: Muestra la cantidad de memoria utilizada por el contenedor.
    d. *{{.MemPerc}}*: Indica el porcentaje de uso de memoria por parte del contenedor.

Los resultados se interpretan de la siguiente manera:

- CONTAINER: Esta columna muestra el nombre o identificador único del contenedor Docker. En este caso, el nombre del contenedor es moodle-project_moodle_1.
- CPU %: Esta columna muestra el porcentaje de uso de CPU del contenedor. En el ejemplo proporcionado, el contenedor está utilizando aproximadamente el 2.34% de la capacidad total de CPU.
- MEM USAGE / LIMIT: Esta columna muestra la cantidad de memoria utilizada por el contenedor y el límite de memoria asignado. En el ejemplo, se muestra que el contenedor está utilizando 291.8MiB (megabytes) de memoria de un límite total de 7.707GiB (gigabytes). Esta información indica cuánta memoria está consumiendo el contenedor en relación con la memoria total asignada.
- MEM %: Esta columna muestra el porcentaje de uso de memoria del contenedor en relación con su límite. En el ejemplo, se muestra que el contenedor está utilizando aproximadamente el 3.70% de la memoria asignada.

c. Pruebas de carga y estrés con Siege

En esta sección, se llevan a cabo pruebas de estrés en el contenedor de bitnami/moodle utilizando la herramienta "siege". Estas pruebas permitirán evaluar el rendimiento y la capacidad de respuesta del contenedor frente a una carga simulada. Se generarán solicitudes intensivas para verificar el comportamiento del contenedor bajo condiciones de alto tráfico y determinar su capacidad de mantener un rendimiento óptimo.

Una vez ejecutadas las pruebas de estrés con siege, se utilizará la herramienta "stacer" para evaluar el rendimiento del sistema operativo Fedora mientras el contenedor de bitnami/moodle está en funcionamiento. Además, durante todo el proceso de pruebas de estrés y monitoreo, se utilizará "docker stats" para obtener estadísticas en tiempo real del contenedor de bitnami/moodle.

A continuación, se proporciona una pequeña guía detallada para instalar y usar siege:

1) Actualizar los repositorios de paquetes usando el siguiente comando:

   sudo dnf update

2) Instalar Siege con el siguiente comando:

   sudo dnf install siege

3) Una vez que la instalación se complete, verificar que la versión de Siege esté instalada correctamente con el siguiente comando:

   siege -V

   La salida del comando debe mostrar la versión de Siege instalada en su sistema.

En esta etapa de la investigación, se llevarán a cabo una prueba de estrés en el contenedor de bitnami/moodle para evaluar su rendimiento bajo un escenario de carga. La prueba consistirá en someter al servidor local a una carga general, simulando un alto número de solicitudes concurrentes. Esto permitirá evaluar la capacidad del contenedor y del servidor para manejar una carga significativa y mantener un rendimiento óptimo.

Esta prueba se repetirá un total de 5 veces para recolectar una cantidad significativa de datos. Esto permitirá obtener resultados más precisos y consistentes, lo cual es esencial para un análisis exhaustivo del rendimiento del sistema. Los datos recolectados serán utilizados para realizar un análisis detallado y obtener conclusiones sólidas sobre el comportamiento del contenedor de bitnami/moodle bajo diferentes cargas de trabajo.

Pruebas de estrés al servidor local

En esta sección se realiza la implementación de la herramienta Siege con el propósito de evaluar el rendimiento y la capacidad de respuesta del servidor mediante pruebas de carga y estrés.

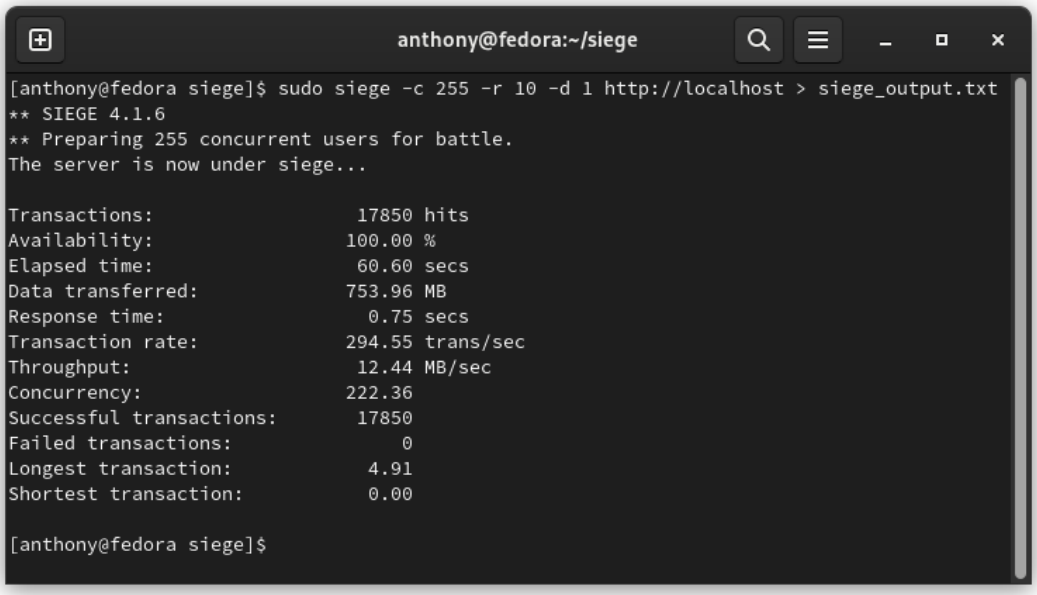


Figura 5. Primera prueba de estrés al servidor local

El código mostrado en la figura 5 realiza lo siguiente:

- *sudo*: Es un comando utilizado en sistemas Linux para ejecutar comandos con privilegios de superusuario. Se utiliza aquí para asegurarse de que el comando se ejecute con los permisos necesarios.
- *siege*: Es la herramienta de prueba de carga utilizada. Se utiliza para simular múltiples usuarios accediendo a un servidor al mismo tiempo y medir el rendimiento y la capacidad de respuesta del servidor.
- *-c 255*: Es la opción que indica el número de usuarios concurrentes que se simularán. En este caso, se simularán 255 usuarios accediendo simultáneamente al servidor.
- *-r 10*: Es la opción que indica el número de repeticiones del escenario de prueba. En este caso, el escenario de prueba se repetirá 10 veces.
- *-d 1*: Es la opción que indica el tiempo de retardo entre las solicitudes realizadas por cada usuario. En este caso, se establece un retardo de 1 segundo entre cada solicitud.

- *http://localhost*: Es la URL del servidor al que se realizarán las solicitudes. En este caso, se realiza una prueba en el servidor local en el puerto 80.
- *> siege_output.txt*: Es la redirección de salida que indica que la salida generada por Siege se debe guardar en un archivo llamado "siege_output.txt" en lugar de mostrarla en la pantalla.

En resumen, el código ejecuta una prueba de carga simulando 255 usuarios accediendo al servidor local en el puerto 80, repitiendo el escenario de prueba 10 veces, con un retardo de 1 segundo entre las solicitudes. La salida de la prueba se guarda en un archivo llamado "siege_output.txt". Esto permite analizar posteriormente los resultados de la prueba y evaluar el rendimiento del servidor bajo carga simulada.

Interpretación de los resultados de la prueba de carga proporcionados por Siege:

- Transactions: Indica el número total de solicitudes realizadas durante la prueba. En este caso, se realizaron 17850 solicitudes.
- Availability: Muestra el porcentaje de disponibilidad del servidor durante la prueba. Un valor del 100% significa que todas las solicitudes fueron exitosas. En este caso, todas las 17850 solicitudes fueron exitosas, lo que indica una disponibilidad del 100%.
- Elapsed time: Es el tiempo total transcurrido durante la prueba, medido en segundos. En este caso, la prueba duró 60.60 segundos.
- Data transferred: Muestra la cantidad total de datos transferidos durante la prueba. En este caso, se transfirieron 753.96 megabytes (MB) de datos.
- Response time: Indica el tiempo promedio de respuesta del servidor a cada solicitud, medido en segundos. En este caso, el tiempo promedio de respuesta fue de 0.75 segundos.
- Transaction rate: Es la tasa de transacciones por segundo, es decir, la cantidad promedio de solicitudes realizadas por segundo durante la prueba. En este caso, se realizaron aproximadamente 294.55 transacciones por segundo.

- Throughput: Es la cantidad promedio de datos transferidos por segundo durante la prueba, medido en megabytes por segundo (MB/sec). En este caso, el rendimiento promedio fue de 12.44 MB por segundo.
- Concurrency: Indica el número promedio de usuarios concurrentes durante la prueba. En este caso, se tuvo una concurrencia promedio de 222.36 usuarios simultáneos.
- Successful transactions: Muestra el número total de solicitudes exitosas durante la prueba. En este caso, todas las 17850 solicitudes fueron exitosas.
- Failed transactions: Indica el número de solicitudes que fallaron durante la prueba. En este caso, no se registraron solicitudes fallidas, lo que indica un 0% de solicitudes fallidas.
- Longest transaction: Muestra el tiempo de respuesta más largo registrado durante la prueba, medido en segundos. En este caso, la solicitud más lenta tomó 4.91 segundos en ser procesada.
- Shortest transaction: Indica el tiempo de respuesta más corto registrado durante la prueba, medido en segundos. En este caso, la solicitud más rápida fue procesada instantáneamente con un tiempo de respuesta de 0 segundos.

Durante el desarrollo de la investigación, se ejecutó el proceso mencionado previamente en cinco ocasiones independientes. Cada repetición del proceso generó un conjunto de datos y resultados específicos. Estos resultados se han representado gráficamente en las figuras 5, 6, 7, 8 y 9, las cuales proporcionan una visualización clara y concisa de los datos obtenidos en cada repetición.

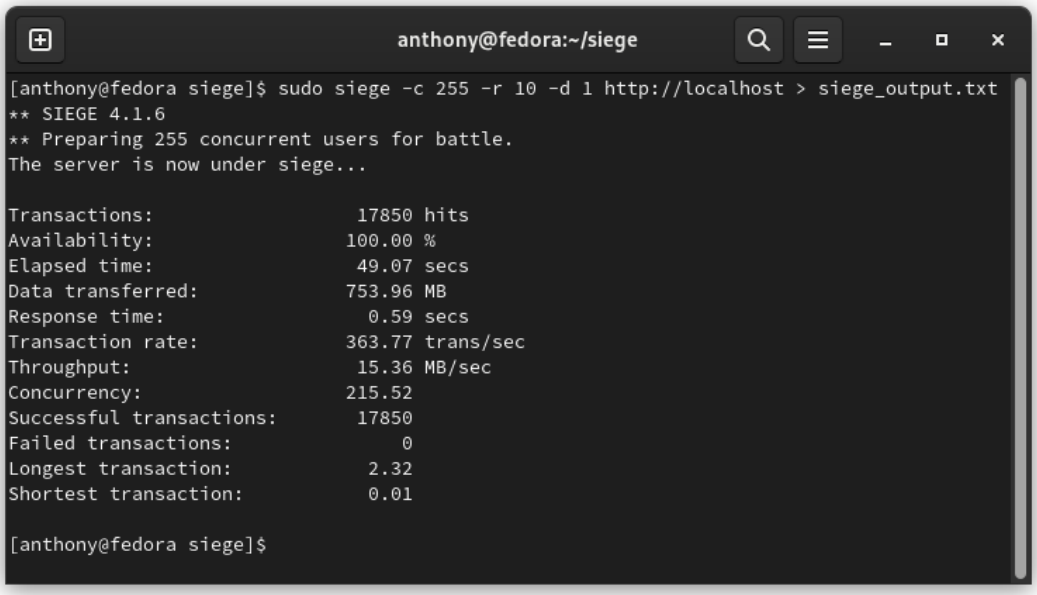


Figura 6. Segunda prueba de estrés al servidor local

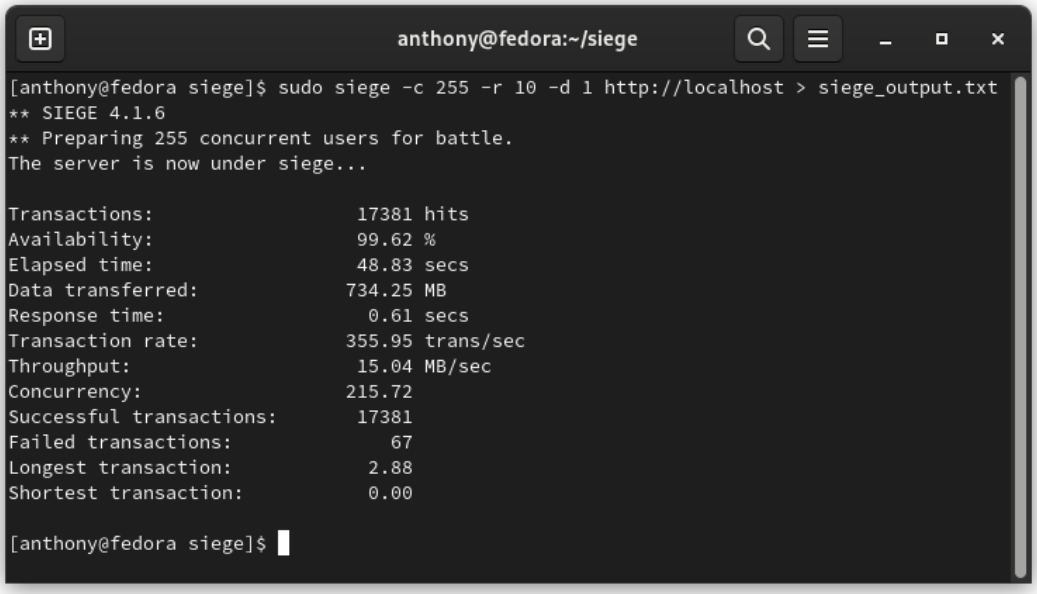


Figura 7. Tercera prueba de estrés al servidor local

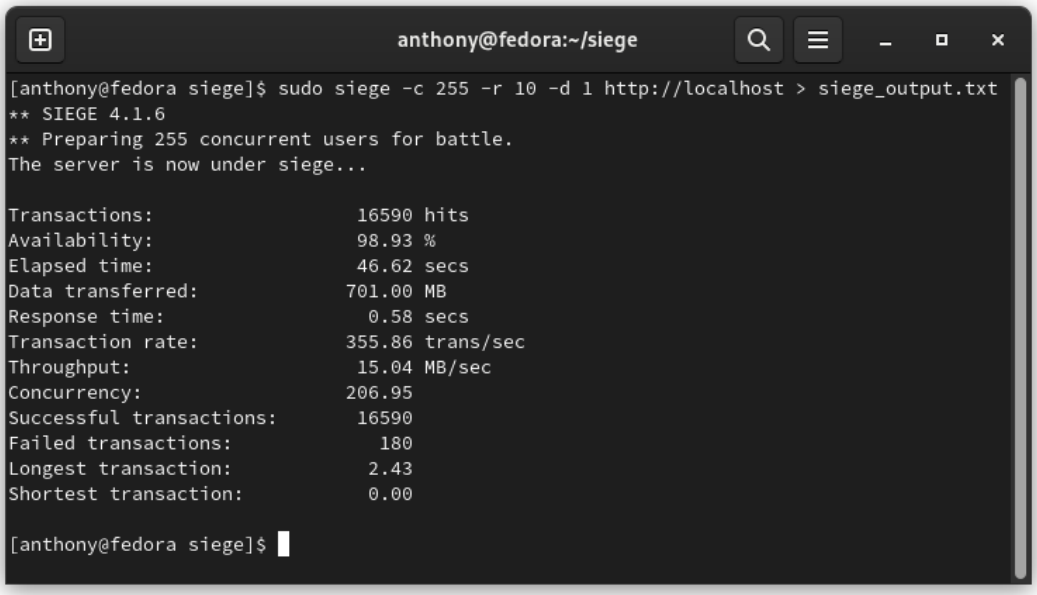


Figura 8. Cuarta prueba de estrés al servidor local

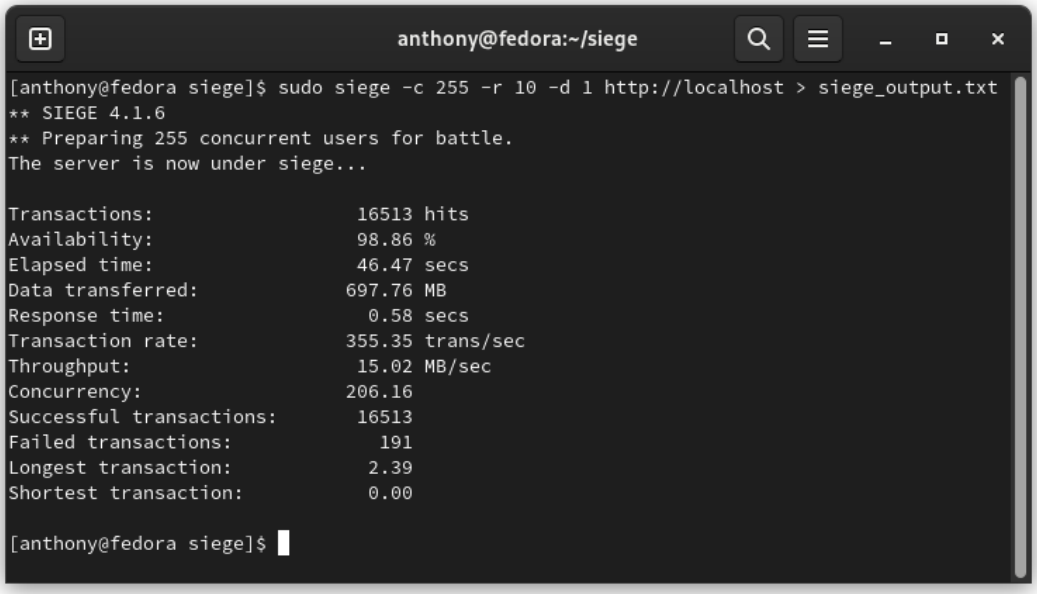


Figura 9. Quinta prueba de estrés al servidor local

Selección de datos para hacer comparativas

Entre los datos proporcionados, hay varios que son relevantes para hacer comparativas con otros contenedores:

- Availability: La disponibilidad del 100.00% indica que todas las solicitudes fueron exitosas, lo cual es un indicador importante de la estabilidad y el buen funcionamiento del contenedor.
- Elapsed time: El tiempo transcurrido de 60.60 segundos indica la duración total de la prueba de carga. Este dato es importante para comprender el rendimiento sostenido del contenedor a lo largo del tiempo.
- Response time: El tiempo de respuesta de 0.75 segundos representa la rapidez con la que el contenedor procesa las solicitudes. Un tiempo de respuesta bajo es deseable, ya que indica una alta capacidad de respuesta y eficiencia.
- Transaction rate: La tasa de transacciones de 294.55 transacciones por segundo muestra la cantidad de solicitudes que el contenedor puede manejar en un período de tiempo determinado. Una tasa de transacciones alta puede indicar un alto rendimiento y capacidad de procesamiento.
- Throughput: El rendimiento de 12.44 MB por segundo indica la cantidad de datos que el contenedor puede transferir en un intervalo de tiempo dado. Un mayor rendimiento indica una mayor capacidad para manejar grandes volúmenes de datos.
- Concurrency: La concurrencia de 222.36 representa el número de usuarios simultáneos que el contenedor puede manejar de manera efectiva. Una alta concurrencia indica una buena capacidad de escalabilidad y la capacidad de manejar múltiples solicitudes al mismo tiempo.

Estos datos son importantes para comparar el rendimiento y la eficiencia de diferentes contenedores, ya que proporcionan información sobre la disponibilidad, la duración de la prueba de carga, el tiempo de respuesta, la capacidad de procesamiento y la capacidad de transferencia de datos.

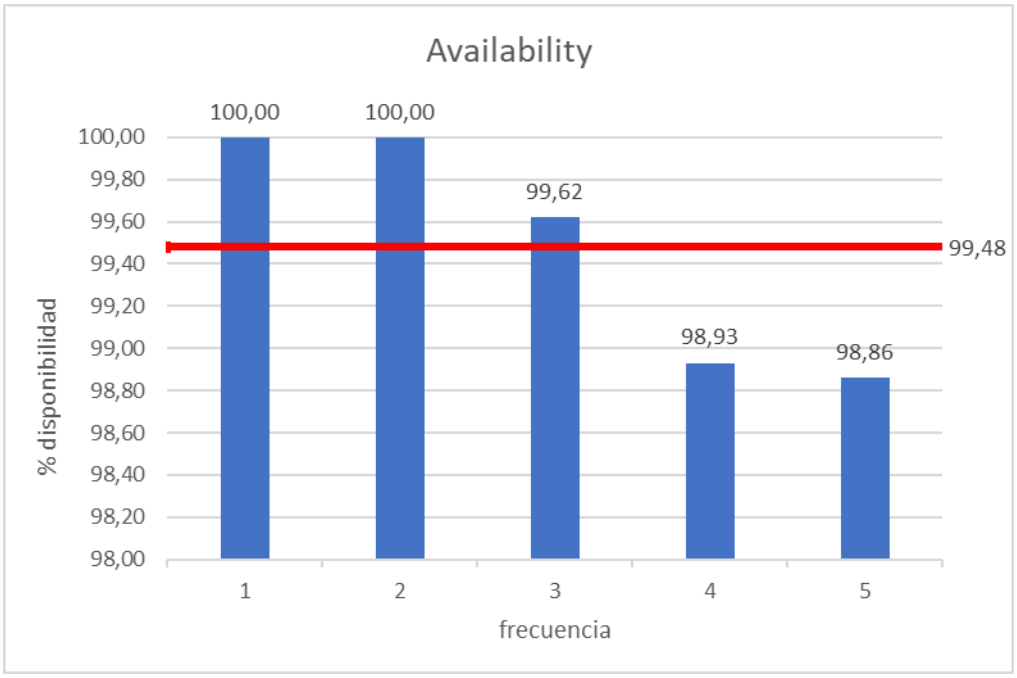


Figura 10. Gráfico con las estadísticas recolectadas de "Availability"

En base a la figura 10, se concluye que, durante las cinco ejecuciones de la prueba de estrés en el contenedor, se logró un promedio del 99,48% en el indicador de "disponibilidad" o "availability".

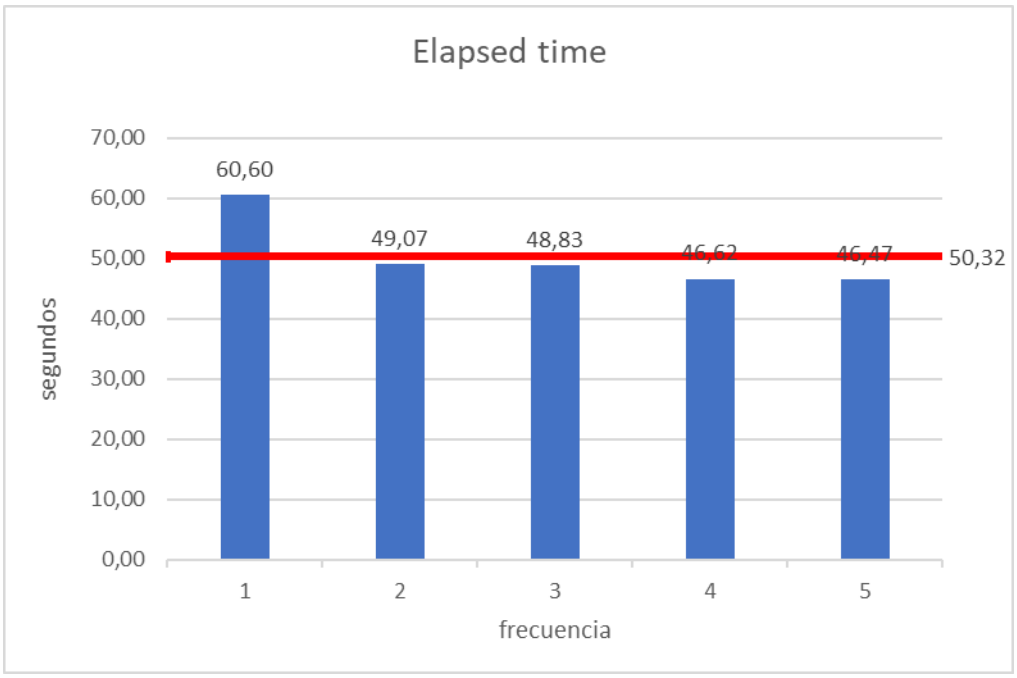


Figura 11. Gráfico con las estadísticas recolectadas de "Elapsed time"

En base a la figura 11, se pudo determinar que, durante las cinco ejecuciones de la prueba de estrés en el contenedor, se obtuvo un promedio de 50,32 segundos para el indicador de "elapsed time" o "tiempo transcurrido".

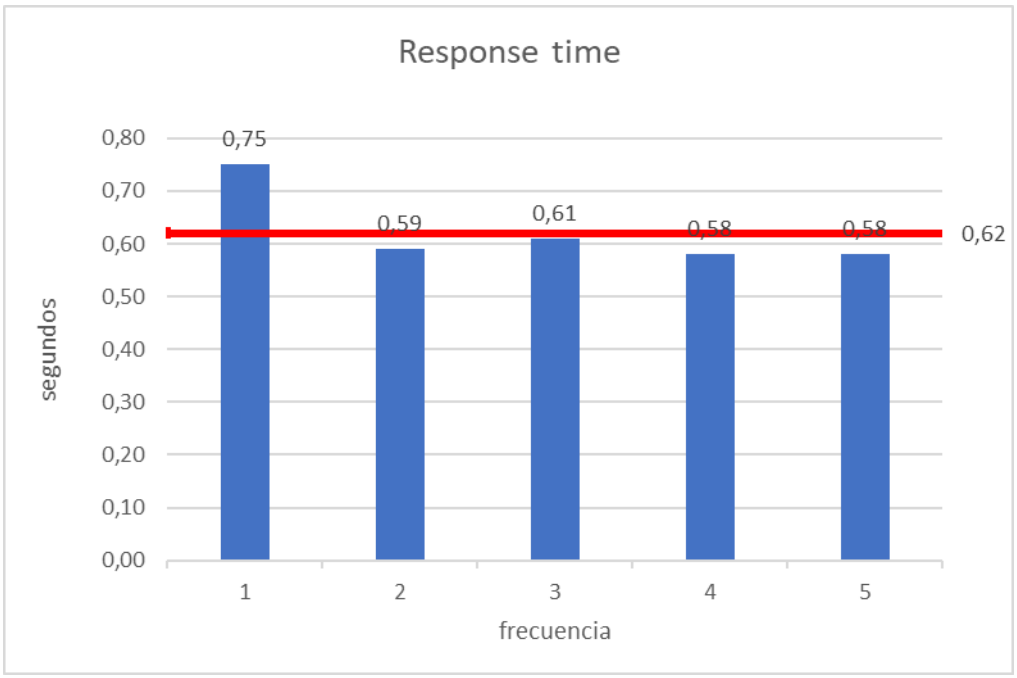


Figura 12. Gráfico con las estadísticas recolectadas de "Response time"

En base a la figura 12, se determinó que, durante las cinco ejecuciones de la prueba de estrés en el contenedor, se obtuvo un promedio de 0,62 segundos para el indicador de "response time" o "tiempo de respuesta".

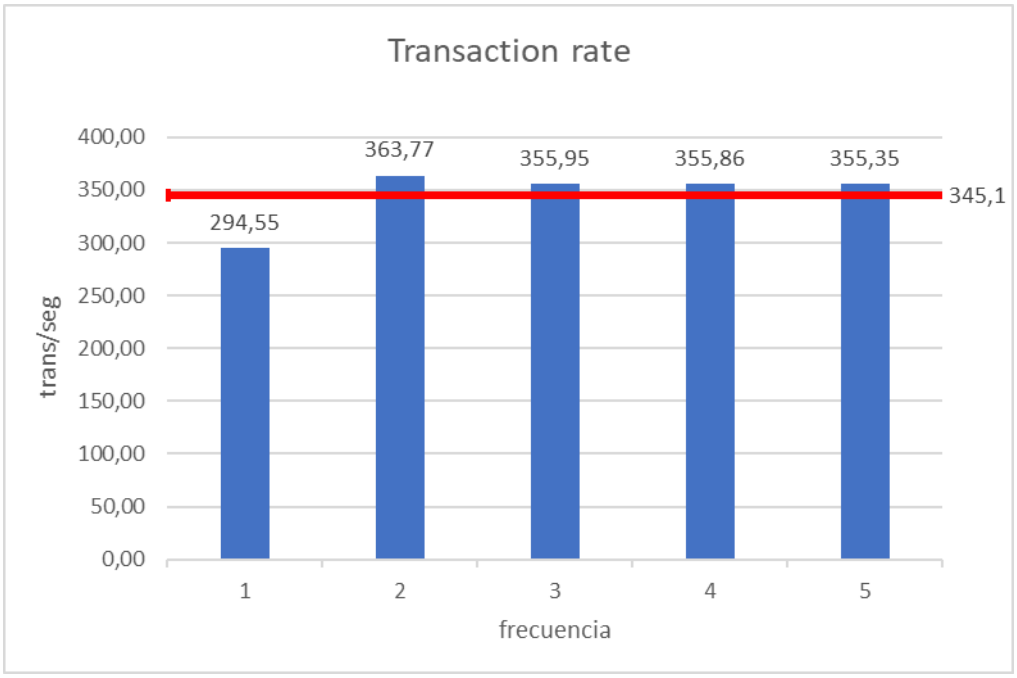


Figura 13. Gráfico con las estadísticas recolectadas de "Transaction rate"

En base a la figura 13, se determinó que, durante las cinco ejecuciones de la prueba de estrés en el contenedor, se obtuvo un promedio de 345,1 transacciones por segundo para el indicador de "transaction rate" o "tasa de transacciones".

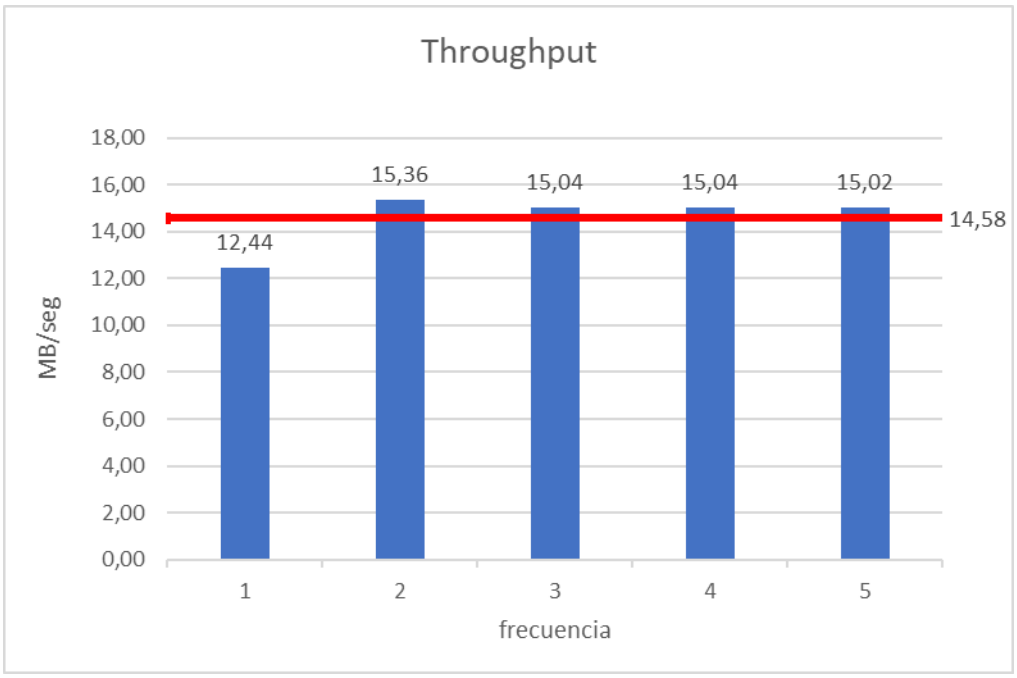


Figura 14. Gráfico con las estadísticas recolectadas de "Throughput"

En base a la figura 14, se determinó que, durante las cinco ejecuciones de la prueba de estrés en el contenedor, se obtuvo un promedio de 14,58 megabytes por segundo (MB/seg) para el indicador de "throughput" o "rendimiento de transferencia".

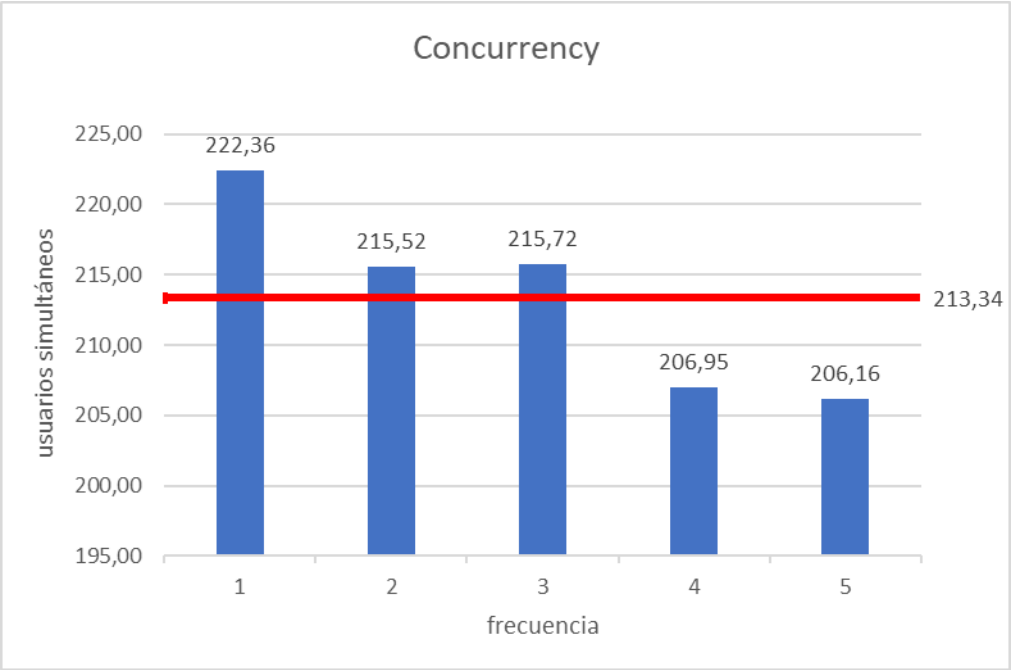


Figura 15. Gráfico con las estadísticas recolectadas de "Concurrency"

En base a la figura 15, se determinó que, durante las cinco ejecuciones de la prueba de estrés en el contenedor, se obtuvo un promedio de 213,34 usuarios simultáneos para el indicador de "concurrency" o "concurrencia".

Monitoreo del sistema con Stacer mientras se ejecuta Siege

En esta sección se lleva a cabo la implementación de la herramienta Stacer con el fin de realizar el monitoreo del sistema durante la ejecución de las pruebas con Siege.

Figura 16. Monitoreo del sistema durante las pruebas de estrés

La figura 16 ilustra el incremento significativo del uso de CPU, memoria y disco durante la ejecución de las pruebas de estrés en el servidor local. Es importante destacar que el consumo de estos recursos se mantiene constante a lo largo de las cinco pruebas, lo cual es esperado dado que se busca alcanzar el máximo consumo de recursos durante las pruebas de estrés. Por consiguiente, es esperable que exista una estabilidad en el consumo de estos recursos y que se mantengan en niveles altos sin variaciones significativas. Esto refuerza la efectividad de las pruebas de estrés al exigir al servidor y al contenedor un alto rendimiento y capacidad de respuesta constante.

Monitoreo del contenedor con docker stats mientras se ejecuta Siege

En esta sección se realiza la implementación de la herramienta docker stats con el propósito de monitorear el sistema durante la ejecución de las pruebas con Siege. Esta información es fundamental para evaluar el desempeño y la estabilidad del entorno en un contexto de contenerización.

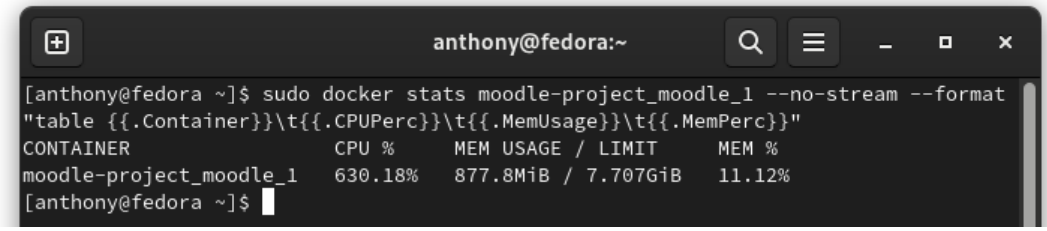


Figura 17. Monitoreo del contenedor en la 1ra prueba de estrés al servidor local

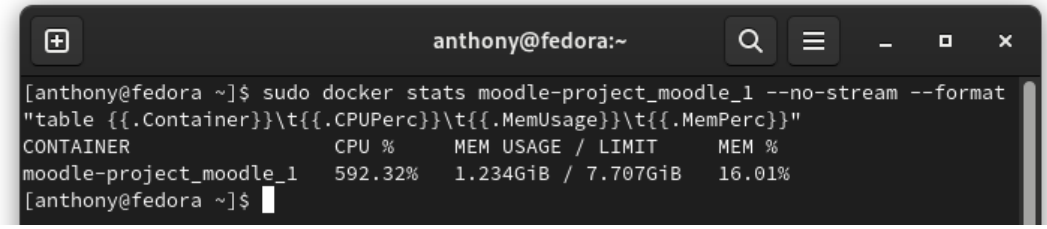


Figura 18. Monitoreo del contenedor en la 2da prueba de estrés al servidor local

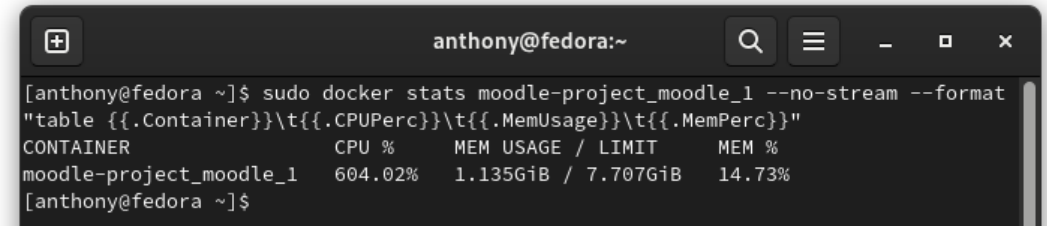


Figura 19. Monitoreo del contenedor en la 3ra prueba de estrés al servidor local

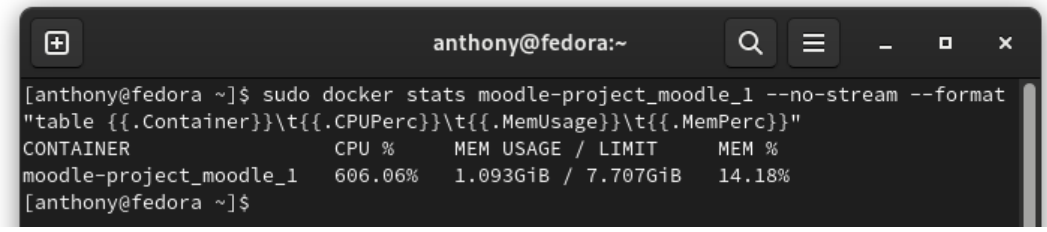


Figura 20. Monitoreo del contenedor en la 4ta prueba de estrés al servidor local

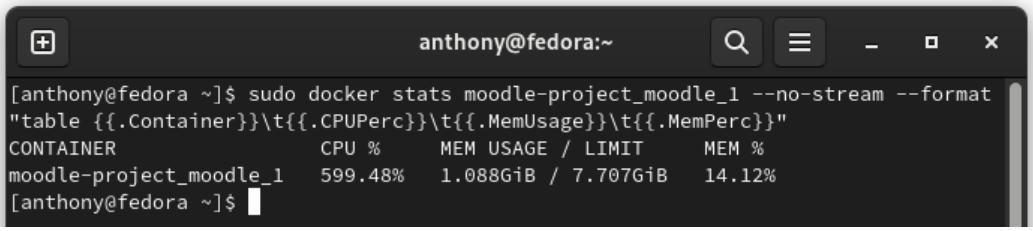


Figura 21. Monitoreo del contenedor en la 5ta prueba de estrés al servidor local

En las Figuras 17, 18, 19, 20 y 21 se observa que el porcentaje de uso de CPU supera el 100%. Esto se debe a la forma en que se calcula y representa este indicador. Al ejecutar un contenedor Docker, es posible configurarlo para utilizar más capacidad de CPU de la disponible en el sistema. Esto se logra mediante la asignación de límites y cuotas de CPU.

Los límites y cuotas de CPU permiten asignar al contenedor una porción específica de la capacidad total de la CPU del sistema. Si un contenedor tiene una cuota de CPU mayor que la capacidad física total de la CPU o si se ejecuta en un entorno con múltiples núcleos de CPU, es posible que su porcentaje de uso de CPU supere el 100.

Es relevante destacar que este porcentaje no indica necesariamente una sobrecarga o un mal funcionamiento del sistema. Depende de la configuración y los límites establecidos para el contenedor. En este caso, es una estrategia válida para aprovechar al máximo los recursos de la CPU y garantizar un rendimiento óptimo de la aplicación que se ejecuta dentro del contenedor.

Datos para hacer comparativas

Los datos de CPU %, MEM USAGE / LIMIT y MEM % son importantes para realizar comparaciones entre contenedores debido a las siguientes razones:

- CPU %: Este dato es crucial para evaluar la carga de trabajo y la eficiencia de un contenedor en particular. Al comparar el uso de CPU entre diferentes contenedores, se puede identificar aquellos que consumen más recursos de CPU y aquellos que son más eficientes en términos de utilización de recursos.

- MEM USAGE / LIMIT: Al comparar los valores de uso de memoria entre contenedores, se puede determinar cuánta memoria está siendo utilizada por cada uno. Esto es relevante para identificar contenedores que requieren una cantidad significativa de memoria en comparación con otros, lo que podría indicar posibles problemas de consumo excesivo de memoria o la necesidad de ajustar la asignación de recursos.
- MEM %: Comparar este valor entre contenedores permite identificar aquellos que están utilizando más memoria en relación con su límite. Esto puede ser importante para evaluar el rendimiento y la eficiencia en el uso de la memoria de los contenedores.

Estos datos proporcionan una visión detallada del uso de CPU y memoria de los contenedores, lo que nos permite comparar su rendimiento y eficiencia en relación con los recursos asignados. Estas métricas son valiosas para tomar decisiones informadas sobre la configuración y optimización de los contenedores, así como para identificar posibles problemas de consumo excesivo de recursos.

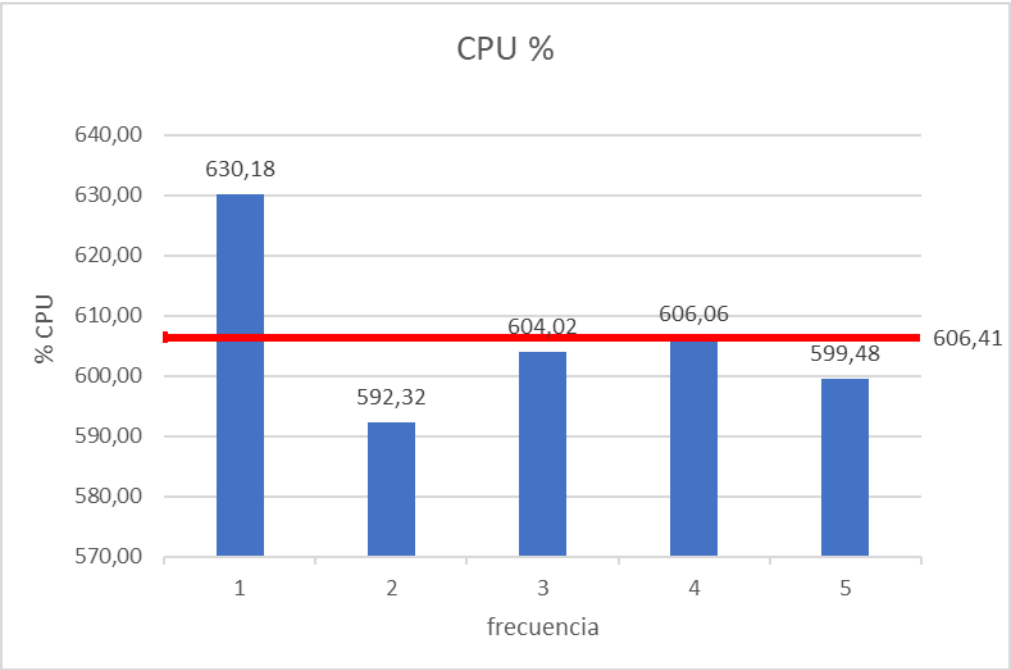


Figura 22. Gráfico con las estadísticas recolectadas de "CPU %"

En base a la figura 22, se determinó que durante las cinco ocasiones en que se ejecutó la prueba de estrés en el contenedor, se obtuvo un promedio de 606,41% de uso de CPU para el apartado "CPU %".

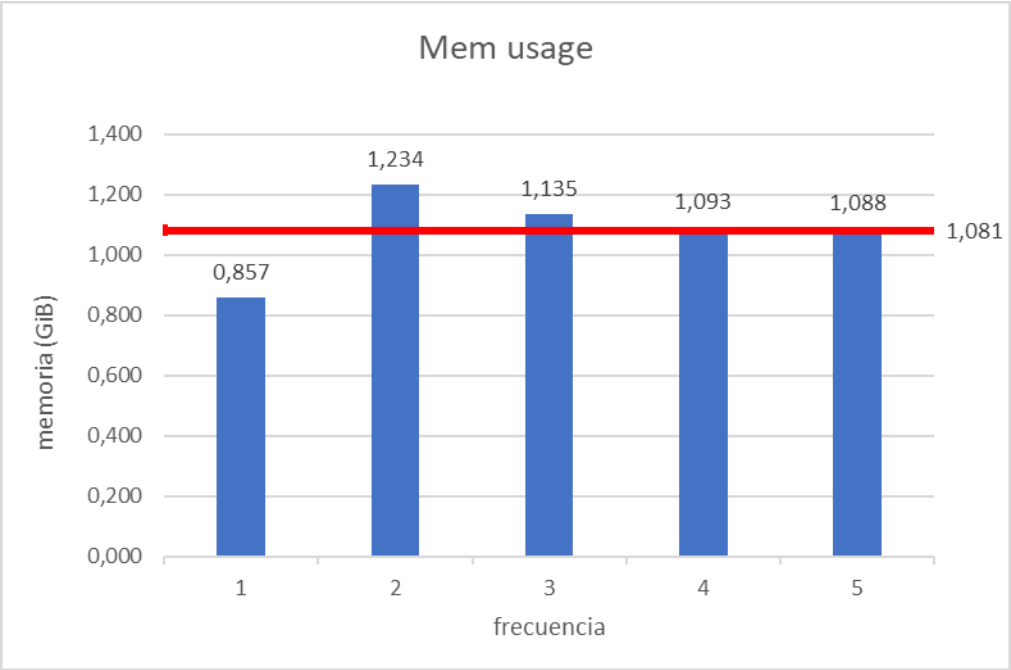


Figura 23. Gráfico con las estadísticas recolectadas de "Mem usage"

En base a la figura 23, se determinó que durante las cinco ocasiones en que se ejecutó la prueba de estrés en el contenedor, se obtuvo un promedio de 1,081 GiB de memoria utilizada por el contenedor para el apartado "mem usage".

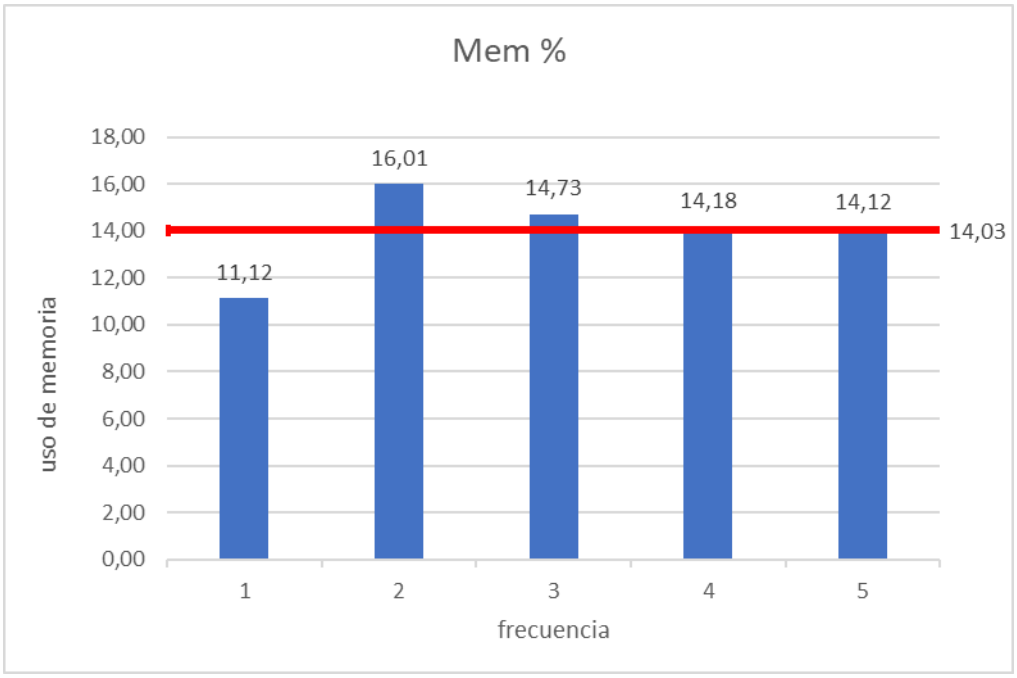


Figura 24. Gráfico con las estadísticas recolectadas de "Mem %"

En base a la figura 24, se determinó que durante las cinco ocasiones en que se ejecutó la prueba de estrés en el contenedor, se obtuvo un promedio de 14,03% de uso de memoria para el apartado "mem %".

3.1.10. Aplicación para testear el desarrollo con contenedores

Para el desarrollo con contenedores, se desarrolló una página web que contiene una aplicación To-Do List, diseñada con una variedad de tecnologías web. La estructura de la página fue creada con HTML, mientras que CSS proporcionó la estética visual. La funcionalidad e interactividad se manejó mediante JavaScript. Bootstrap contribuyó con estilos preestablecidos y componentes reutilizables, y la representación de mensajes y diálogos interactivos se logró a través de SweetAlert2.

La aplicación To-Do List es una herramienta práctica y eficiente que permite al usuario crear, administrar y completar tareas de manera sencilla. El usuario puede agregar nuevas tareas, marcarlas como completadas, editar su contenido y eliminarlas si es necesario. La interfaz intuitiva y fácil de usar facilita la gestión de las tareas diarias, ayudando al usuario a mantenerse organizado y productivo. Además, se utilizan alertas personalizadas para brindar una experiencia interactiva y visualmente agradable.

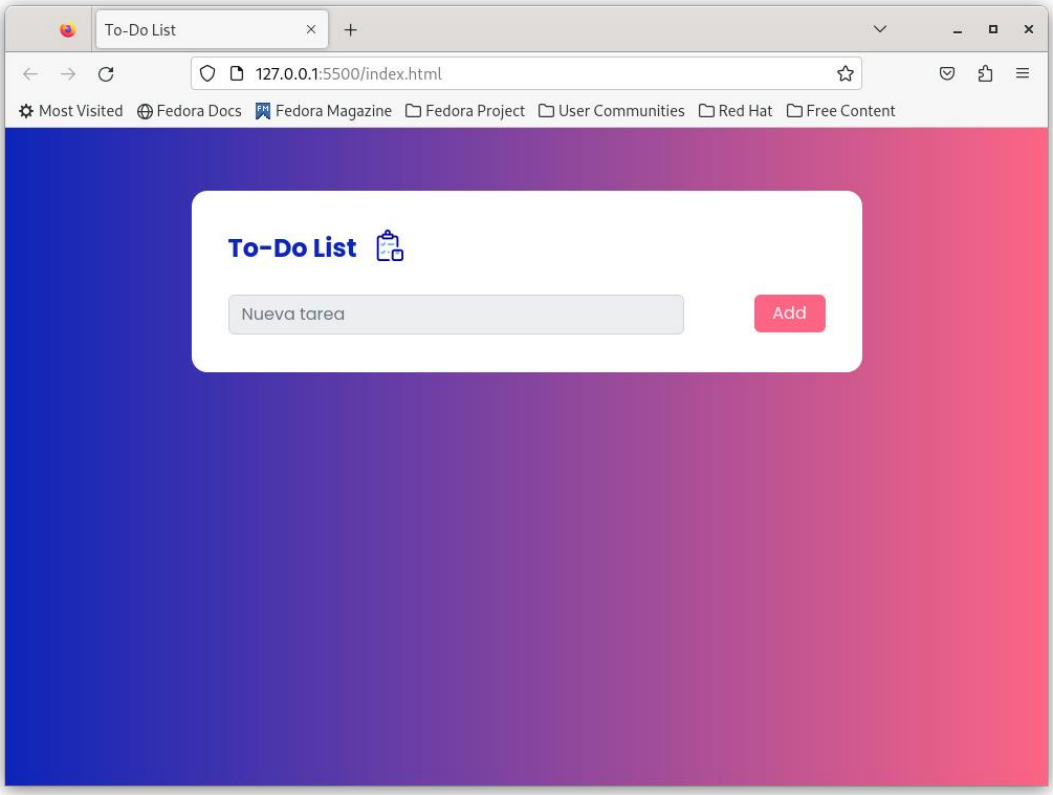


Figura 25. Página web con la aplicación To-Do List

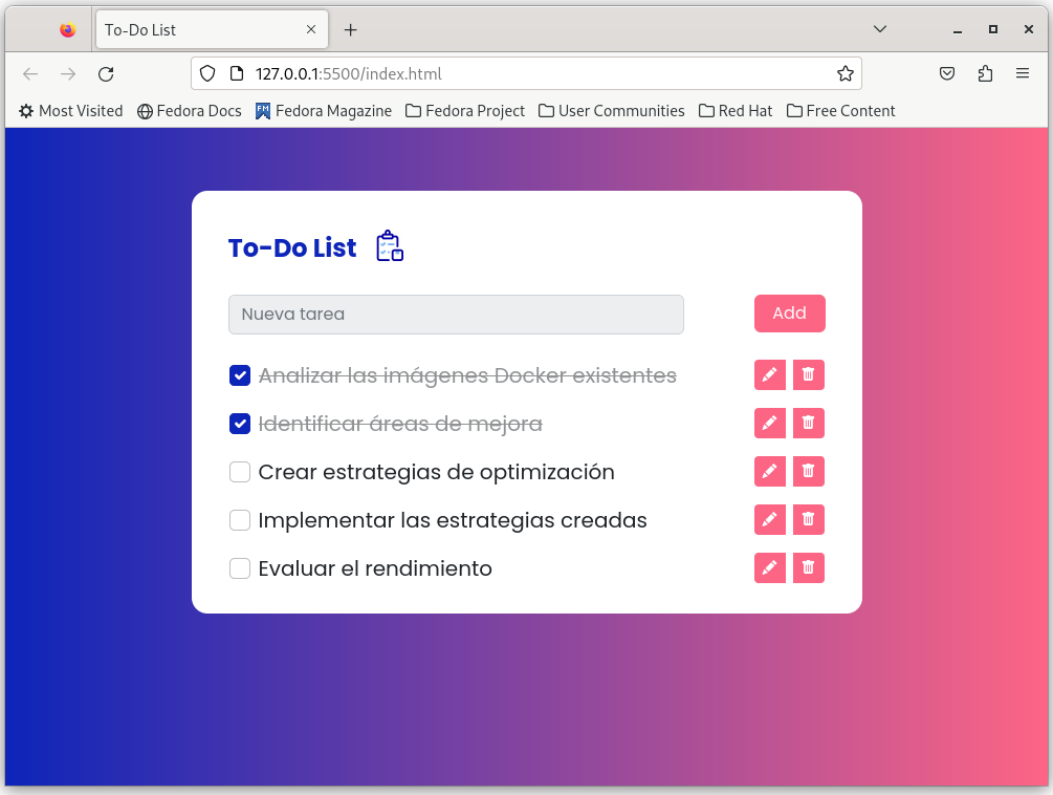


Figura 26. Funcionalidad de la aplicación To-Do List

Esta página web será convertida en una aplicación de escritorio utilizando la herramienta Nativefier. Esta conversión permitirá cuantificar el tiempo requerido para tal transformación, pasando de página web a aplicación de escritorio. El tiempo medido en este proceso será de utilidad para establecer comparaciones con los resultados obtenidos mediante otros contenedores.

El uso de Nativefier requiere la instalación de Node.js, dado que esta herramienta se fundamenta en dicha plataforma. Sin embargo, en el marco de este trabajo y con el objetivo de preservar un ambiente de desarrollo limpio y reproducible, se evitará la instalación directa de Node.js en el sistema operativo anfitrión. En lugar de ello, se recurrirá a la imagen oficial de Node.js disponible en Docker Hub.

3.1.11. Instalación y configuración del contenedor de node.js

Node es un contenedor preempaquetado y preconfigurado que contiene el entorno de ejecución de Node.js. Este contenedor permite a los desarrolladores implementar y

ejecutar sus aplicaciones Node.js en cualquier sistema que soporte Docker, garantizando la consistencia y la portabilidad.

A continuación, se proporciona una pequeña guía detallada para instalar y usar node:

1) Descargar la imagen de Node.js

   Para descargar la versión más reciente, se utiliza el siguiente comando:

   docker pull node

2) Ejecutar un contenedor con la imagen de Node.js

   Una vez se haya descargado la imagen de Node.js, se puede ejecutar un contenedor con ella.

   docker run -it node bash

3.1.12. Implementación de nativefier

Nativefier es una herramienta de código abierto que permite convertir cualquier página web en una aplicación de escritorio independiente. Utiliza la plataforma Electron para generar aplicaciones compatibles con Windows, MacOS y Linux a partir de un sitio web existente.

A continuación, se proporciona una pequeña guía detallada para instalar y usar nativefier en un contenedor Docker con Node.js:

1) Entrar al contenedor

   Primero, es necesario entrar al contenedor Docker que ya tiene Node.js.

   docker run -it --name my-node-app node /bin/bash

2) Instalar nativefier

   Una vez en el contenedor, se puede instalar nativefier globalmente usando npm (el manejador de paquetes de Node.js):

npm install -g nativefier

3) Usar Nativefier

   Ahora se puede usar nativefier para convertir cualquier página web en una aplicación de escritorio. Aquí un ejemplo:

   nativefier "http://ejemplo.com"

   Este comando creará una nueva aplicación de escritorio para la página web "http://ejemplo.com". La aplicación se generará en un directorio llamado "ejemplo.com-linux-x64" en el directorio actual.

a. Conversión de la página web en una aplicación de escritorio

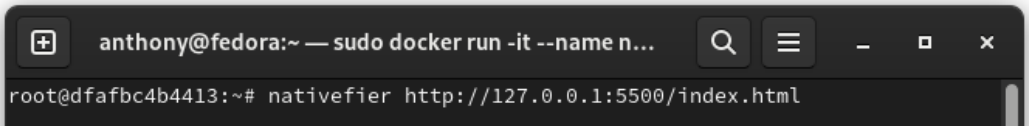


Figura 27. Conversión de la página web en aplicación de escritorio

El código mostrado en la figura 27 realiza lo siguiente:

- *nativefier*: es el comando principal que invoca la herramienta nativefier. Internamente, utiliza el framework Electron para empaquetar el sitio web en una aplicación de escritorio.
- *http://127.0.0.1:5500/index.html*: “127.0.0.1” es la dirección IP de localhost, lo que indica que el sitio web se está ejecutando localmente en la máquina. “5500” es el número de puerto en el que el servidor web está escuchando, y “/index.html” es la ruta al archivo HTML que se desea ver.

A continuación, se explica detalladamente lo que hace el comando en su conjunto:

1) Invocar la herramienta nativefier desde la línea de comandos.
2) Se le dice a nativefier que cree una aplicación de escritorio para la página web en la URL http://127.0.0.1:5500/index.html.

3) Nativefier toma la página web y utiliza Electron para empaquetarla en una aplicación de escritorio.

4) La aplicación resultante se guarda en el directorio de trabajo actual. Tendrá un nombre basado en el título de la página web y estará empaquetada para ejecutarse como una aplicación nativa en el sistema operativo.

Benchmarking de rendimiento de herramientas de desarrollo

Con el propósito de evaluar el rendimiento de la herramienta de desarrollo "nativefier", se llevó a cabo la medición del tiempo requerido para convertir una aplicación web en una aplicación de escritorio.

La medición del tiempo de conversión constituye un indicador clave para evaluar la eficiencia y la velocidad de la herramienta, proporcionando información valiosa sobre su desempeño en el proceso de transformación de aplicaciones web en aplicaciones de escritorio.

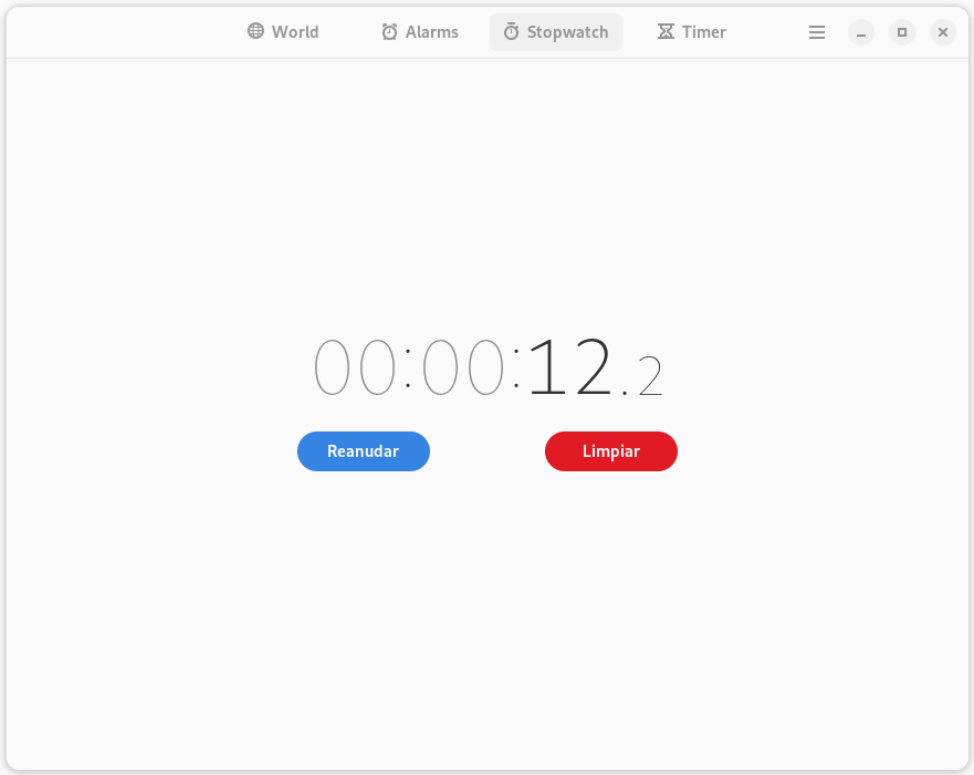


Figura 28. Tiempo de conversión de la página web a aplicación de escritorio

En la figura 28 se presenta el intervalo de tiempo que nativefier requirió para transformar la página web en una aplicación de escritorio, registrándose un lapso aproximado de 12.2 segundos desde la ejecución del comando correspondiente.

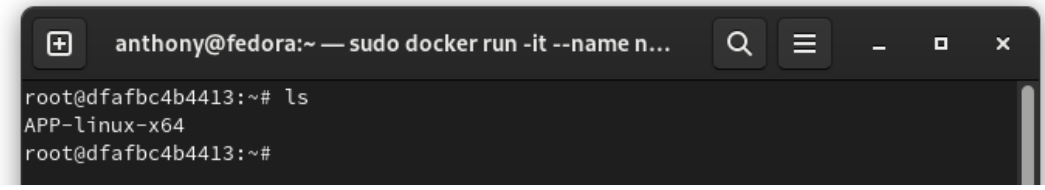


Figura 29. Aplicación de escritorio generada por nativefier

La figura 29 evidencia la creación de un directorio denominado "APP-linux-64" dentro del contenedor de Node.js, correspondiente a la carpeta que alberga la aplicación de escritorio generada mediante el uso de nativefier.

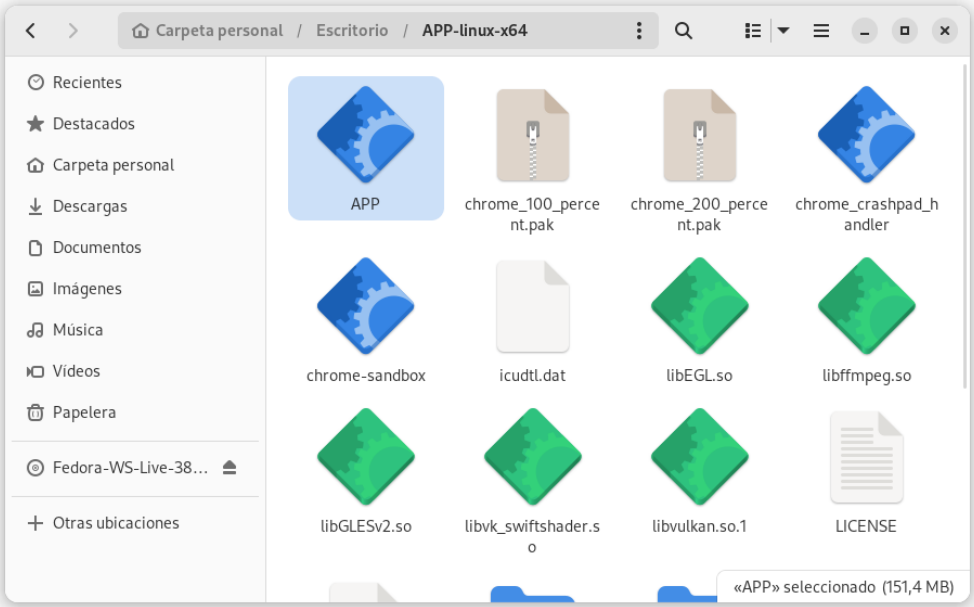


Figura 30. Vistazo a la aplicación de escritorio en el gestor de archivos de Fedora

La figura 30 proporciona una visión a través del gestor de archivos de Fedora hacia el directorio que resguarda la aplicación de escritorio, que, en este escenario particular, recibe la denominación de "APP".

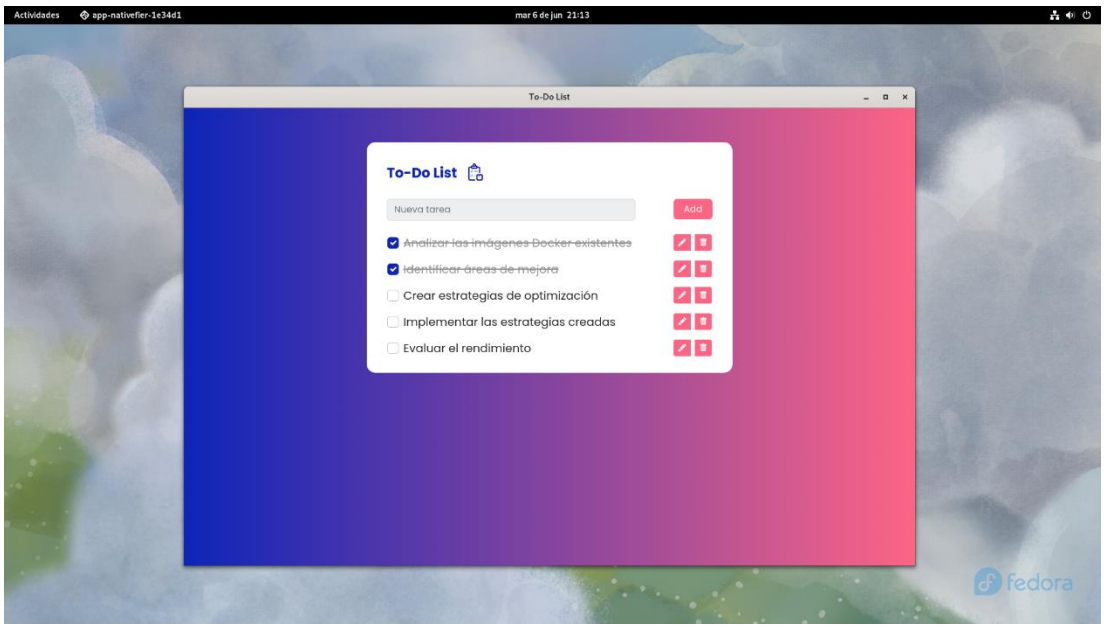


Figura 31. Aplicación ejecutada en Fedora

La figura 31 ilustra la puesta en marcha y el desempeño de la aplicación de escritorio producida mediante nativefier. Se puede apreciar su correcto funcionamiento, sin que se presenten inconvenientes de compatibilidad con el sistema operativo.

3.2. Desarrollo de la propuesta

Una guía de mejores prácticas es un conjunto de recomendaciones y directrices que se han establecido como estándares para realizar una determinada actividad de manera óptima y eficiente. En el contexto de la optimización de imágenes Docker, una guía de mejores prácticas proporciona lineamientos específicos para construir y configurar imágenes Docker de manera eficiente, teniendo en cuenta aspectos como el rendimiento, la seguridad y la eficiencia de recursos.

En este proyecto de investigación, se lleva a cabo la creación de una guía de mejores prácticas para la optimización de imágenes Docker. Esta guía se plantea como una referencia que brinda recomendaciones específicas y estrategias para optimizar el proceso de construcción de imágenes Docker, teniendo en cuenta aspectos como la reducción del tamaño de la imagen, la eliminación de dependencias innecesarias, la configuración adecuada de capas y cachés, entre otros.

I. Introducción a la guía

En las secciones previas de este proyecto de investigación, se ha explorado la importancia de la optimización en la construcción de imágenes Docker y su impacto en el desarrollo con contenedores. En esta guía, el enfoque se centrará en la aplicación práctica y metodológica de estrategias dirigidas específicamente a mejorar la eficiencia y el rendimiento de las imágenes Docker.

El objetivo de esta guía es proporcionar un compendio de tácticas que, cuando se aplican adecuadamente, pueden llevar a la creación de imágenes Docker optimizadas. Esto, a su vez, fomenta un ciclo de desarrollo más ágil y escalable. Al reconocer y abordar los problemas comunes en la construcción de imágenes y al aplicar mejores prácticas, esta guía busca ser un recurso valioso para aquellos comprometidos en el desarrollo con contenedores utilizando Docker.

II. Problemas comunes en la construcción de imágenes Docker

En esta sección, se identifican y analizan varios problemas que comúnmente surgen durante la construcción de imágenes Docker. Comprender estos problemas es crucial para adoptar enfoques efectivos en la optimización de imágenes.

- **Tamaño de la imagen:**

  Descripción: Uno de los problemas más comunes es el tamaño excesivo de las imágenes Docker. Las imágenes grandes pueden consumir más recursos de almacenamiento y ralentizar los tiempos de despliegue.

  Impacto: Afecta negativamente la eficiencia en términos de uso de recursos, aumenta el tiempo de transferencia de imágenes y puede ralentizar el proceso de CI/CD.

  Causas Comunes: Inclusión de paquetes innecesarios, acumulación de archivos temporales, y no limpiar el caché de los administradores de paquetes.

- **Capas ineficientes:**

  Descripción: Docker construye imágenes en capas. A veces, la manera en que se estructuran las capas puede no ser óptima, lo que conduce a redundancias y un uso ineficiente del caché.

  Impacto: El uso ineficiente de las capas puede aumentar el tamaño de la imagen y hacer que la construcción sea menos eficiente.

  Causas Comunes: Uso excesivo de instrucciones RUN en el Dockerfile, no aprovechar el caché de capas y duplicación de datos en múltiples capas.

- **Configuración de seguridad inadecuada:**

  Descripción: Dejar configuraciones de seguridad por defecto o no ajustarlas adecuadamente puede exponer las imágenes y contenedores a vulnerabilidades y riesgos de seguridad.

Impacto: Aumento del riesgo de violaciones de seguridad y exposición de datos sensibles.

Causas Comunes: Uso de imágenes base no seguras, permisos incorrectos en archivos y directorios, y no seguir las prácticas de seguridad estándar.

- **Dependencias innecesarias:**

  Descripción: Incluir archivos o paquetes innecesarios en la imagen puede contribuir al tamaño y complejidad innecesarios de la imagen.

  Impacto: Aumento del tamaño de la imagen, mayor consumo de recursos y posible introducción de vulnerabilidades adicionales.

  Causas Comunes: Instalación de paquetes no esenciales y no eliminar archivos temporales o de compilación.

- **Falta de versionado de imágenes:**

  Descripción: No etiquetar o versionar adecuadamente las imágenes puede llevar a confusiones y dificultades en el seguimiento de cambios y actualizaciones.

  Impacto: Dificulta la gestión de imágenes y la capacidad de revertir a versiones anteriores en caso de problemas.

  Causas Comunes: No asignar etiquetas significativas y no seguir un esquema de versionado coherente.

- **Uso ineficiente de variables de entorno:**

  Descripción: El manejo incorrecto o ineficiente de las variables de entorno puede resultar en configuraciones poco claras y dificultar la reproducibilidad y portabilidad de las imágenes.

  Impacto: Problemas en la configuración de la aplicación, dificultades en la gestión de entornos y potencial exposición de información sensible.

Causas Comunes: Uso inapropiado de variables de entorno para almacenar información sensible, y configuraciones ambiguas o contradictorias.

La identificación y comprensión de estos problemas comunes son cruciales para la toma de decisiones informadas en el proceso de optimización. En las siguientes secciones, se propondrán estrategias para abordar estos problemas y mejorar la eficiencia en la construcción de imágenes Docker.

III. Mejores prácticas en la construcción de imágenes Docker

En esta sección, se presentan prácticas recomendadas para mejorar la eficiencia y seguridad de las imágenes Docker. Estas prácticas tienen como objetivo abordar los problemas comunes identificados anteriormente.

- **Minimizar el tamaño de la imagen:**

  Descripción: Reducir el tamaño de la imagen puede resultar en un uso más eficiente de los recursos y tiempos de despliegue más rápidos.

  Enfoque: Utilizar imágenes base más pequeñas como alpine, evitar la inclusión de paquetes innecesarios y limpiar el caché después de instalar paquetes.

- **Optimizar el uso de capas:**

  Descripción: Optimizar la estructura de las capas puede reducir la redundancia y aprovechar el caché de manera eficiente.

  Enfoque: Combinar múltiples instrucciones RUN en una sola capa y organizar las instrucciones en el Dockerfile para aprovechar el caché de capas.

- **Seguridad en la configuración de imágenes:**

  Descripción: Es vital garantizar que la imagen Docker se construya con prácticas de seguridad en mente.

  Enfoque: Utilizar imágenes base de fuentes confiables, establecer permisos adecuados, y no almacenar información sensible dentro de la imagen.

- **Gestión de dependencias:**

  Descripción: Manejar las dependencias de manera eficiente ayuda a reducir el tamaño de la imagen y minimizar la superficie de ataque.

  Enfoque: Incluir solo las dependencias necesarias y utilizar un gestor de paquetes para manejarlas de manera eficiente.

- **Etiquetado y versionado de imágenes:**

  Descripción: El etiquetado y versionado adecuado de imágenes facilita la gestión y el seguimiento de estas.

  Enfoque: Utilizar etiquetas descriptivas y seguir un esquema de versionado semántico.

- **Uso adecuado de variables de entorno:**

  Descripción: Es importante gestionar correctamente las variables de entorno para asegurar configuraciones claras y reproducibles.

  Enfoque: Definir variables de entorno de manera clara, no almacenar información sensible como variables de entorno y utilizar archivos de configuración cuando sea apropiado.

- **Documentación de la imagen:**

  Descripción: Documentar adecuadamente el Dockerfile y la imagen resultante es importante para facilitar su uso y mantenimiento.

  Enfoque: Incluir comentarios en el Dockerfile y proporcionar documentación detallada sobre cómo utilizar y configurar la imagen.

Estas prácticas recomendadas forman la base para desarrollar estrategias de optimización más avanzadas y específicas, que se abordarán en la siguiente sección. Es esencial considerar cada uno de estos aspectos en función de los requisitos particulares y el contexto en el que se desplegarán las imágenes Docker.

IV. Estrategias de optimización para la construcción de imágenes Docker

Después de considerar los problemas comunes y las mejores prácticas, en esta sección se desarrollan estrategias específicas de optimización que se pueden aplicar durante la construcción de imágenes Docker.

- **Multistage builds:**

  Descripción: Los Multistage Builds permiten dividir la construcción de la imagen en múltiples etapas, lo cual es útil para reducir el tamaño final de la imagen y aislar diferentes aspectos de la construcción.

  Aplicación: Utilizar diferentes etapas en el Dockerfile para separar la compilación de la aplicación y la configuración del entorno de ejecución. Copiar solo los archivos necesarios de una etapa a otra.

- **Optimización de instrucciones del dockerfile:**

  Descripción: Reorganizar y optimizar las instrucciones en el Dockerfile puede resultar en un mejor aprovechamiento del caché de capas y una reducción en el tiempo de construcción.

  Aplicación: Mover las instrucciones que cambian con menos frecuencia hacia la parte superior del Dockerfile. Combina instrucciones RUN relacionadas para reducir el número de capas.

- **Salud y monitorización del contenedor:**

  Descripción: Monitorear el estado de salud y rendimiento del contenedor es esencial para asegurar que la aplicación funcione correctamente.

  Aplicación: Utilizar instrucciones de HEALTHCHECK en el Dockerfile y herramientas de monitorización externas para supervisar el rendimiento del contenedor en tiempo real.

- **Minimizar la superficie de ataque:**

  Descripción: Reducir la superficie de ataque de la imagen implica eliminar componentes innecesarios que podrían ser explotados por agentes maliciosos.

  Aplicación: Desinstalar paquetes innecesarios, cerrar puertos no utilizados y establecer usuarios con privilegios mínimos necesarios para ejecutar la aplicación.

- **Optimización de recursos y limitaciones:**

  Descripción: Controlar el uso de recursos por parte de los contenedores puede mejorar el rendimiento y evitar el agotamiento de recursos del sistema.

  Aplicación: Utilizar opciones de tiempo de ejecución para limitar la cantidad de CPU, memoria y otros recursos que un contenedor puede utilizar.

Es importante destacar que la aplicación de estas estrategias debe ser adaptada al contexto específico del proyecto y las necesidades de la aplicación que se esté contenerizando. Además, es fundamental realizar pruebas exhaustivas para validar el impacto de estas optimizaciones en el rendimiento y la seguridad de las imágenes Docker construidas.

V. Implementación y benchmarking de las estrategias

En esta sección, se aborda la implementación de las estrategias de optimización propuestas y se realiza un benchmarking para evaluar el rendimiento y la eficiencia de las imágenes Docker resultantes. El objetivo es validar y comparar los resultados obtenidos al aplicar las estrategias en un entorno real.

a. Entorno de pruebas

El entorno de pruebas seleccionado para llevar a cabo las evaluaciones y comparaciones entre los viejos y nuevos contenedores será el mismo que se propuso en la sección 3.1.8, titulada "Entorno de producción para pruebas de contenedores". Este entorno, previamente establecido, ofrece un ambiente controlado y adecuado para

realizar pruebas y mediciones de rendimiento. Utilizando este entorno de pruebas consistente, se podrán obtener resultados comparativos precisos y confiables, permitiendo evaluar de manera efectiva las mejoras y optimizaciones implementadas en los nuevos contenedores.

b. Implementación de las estrategias

La creación de un Dockerfile específico con instrucciones precisas para generar imágenes puede considerarse una estrategia de optimización en el desarrollo con contenedores. Mediante la elaboración de un Dockerfile personalizado, se logra un mayor control y modularidad en la configuración y dependencias del contenedor. Esto permite una gestión más eficiente de los recursos y una reducción en el tamaño del contenedor resultante.

Al tener instrucciones específicas en el Dockerfile, es posible seleccionar únicamente los componentes y dependencias necesarios para la aplicación o servicio en cuestión. Se evita la inclusión de elementos innecesarios, lo cual conduce a un menor consumo de recursos y una mejor optimización del rendimiento.

Además, esta estrategia proporciona flexibilidad para ajustar y actualizar fácilmente las dependencias y configuraciones a medida que evoluciona la aplicación. Se pueden incorporar las versiones más recientes y estables de los componentes, así como aplicar cambios específicos para optimizar el rendimiento y la seguridad.

c. Estrategias aplicadas sobre bitnami/moodle

En esta sección, se presentan las estrategias empleadas para optimizar el entorno de pruebas del contenedor bitnami/moodle. Se exploran diversas estrategias y configuraciones implementadas con el propósito de mejorar el rendimiento, la eficiencia y la estabilidad del entorno de pruebas.

Crear un Dockerfile personalizado desde cero sobre la imagen base oficial de bitnami/moodle ofrece varias ventajas en términos de optimización:

- Control total sobre las dependencias: Al crear un Dockerfile desde cero, se tiene un control completo sobre las dependencias y los paquetes instalados en

el contenedor. Esto permite seleccionar únicamente los componentes necesarios para el entorno de pruebas, evitando la inclusión de dependencias innecesarias que podrían afectar el rendimiento y el tamaño del contenedor.

- Configuración específica del rendimiento: Al tener un Dockerfile personalizado, es posible ajustar y optimizar la configuración para mejorar el rendimiento del entorno de pruebas. Esto incluye ajustes como el límite de memoria para PHP, la configuración de la caché y otras configuraciones específicas de Apache y PHP para optimizar el rendimiento de Moodle.
- Flexibilidad para futuras actualizaciones: Al mantener un Dockerfile personalizado, se tiene la flexibilidad de realizar cambios y actualizaciones específicas según las necesidades del entorno de pruebas. Esto permite adaptarse a futuras actualizaciones de bitnami/moodle o implementar cambios específicos sin tener que depender completamente de la imagen base preconfigurada.

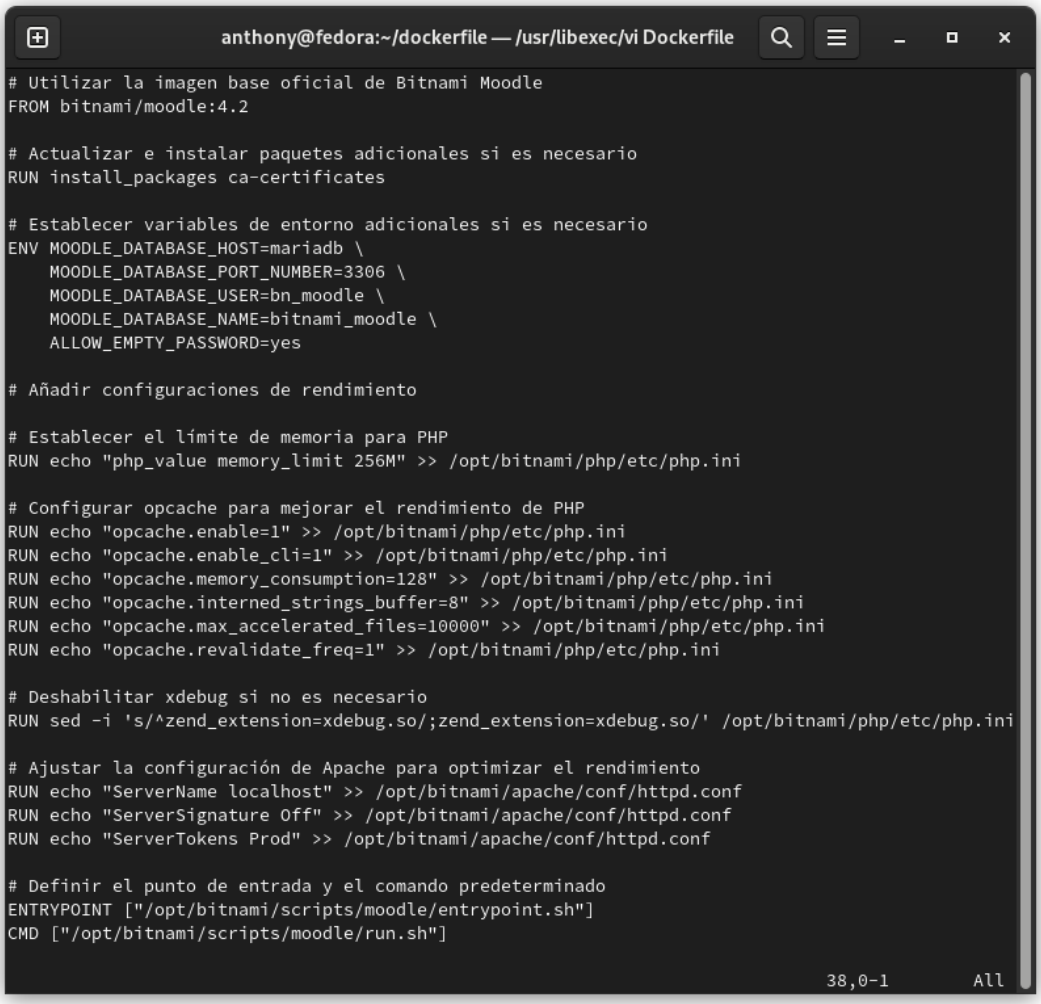


Figura 32. Dockerfile personalizado de bitnami/moodle

Con relación al Dockerfile proporcionado en la figura 32, se han seguido las siguientes estrategias de optimización:

- Configuración de variables de entorno: Se han establecido variables de entorno adicionales para facilitar la conexión con la base de datos MariaDB y permitir un acceso sencillo y seguro.
- Ajustes de rendimiento: Se han realizado ajustes específicos en la configuración de PHP y opcache para mejorar el rendimiento y la eficiencia del entorno.

- Optimización de Apache: Se han realizado cambios en la configuración de Apache para mejorar el rendimiento y la seguridad, como deshabilitar la información de versión y ajustar la configuración del servidor.
- Entrypoint y comando predeterminado: Se han definido el punto de entrada y el comando predeterminado para ejecutar correctamente el contenedor bitnami/moodle.

Estas estrategias de optimización ayudan a mejorar el rendimiento y la eficiencia del entorno de pruebas de bitnami/moodle, asegurando un entorno óptimo para evaluar el comportamiento y desempeño de la aplicación Moodle.

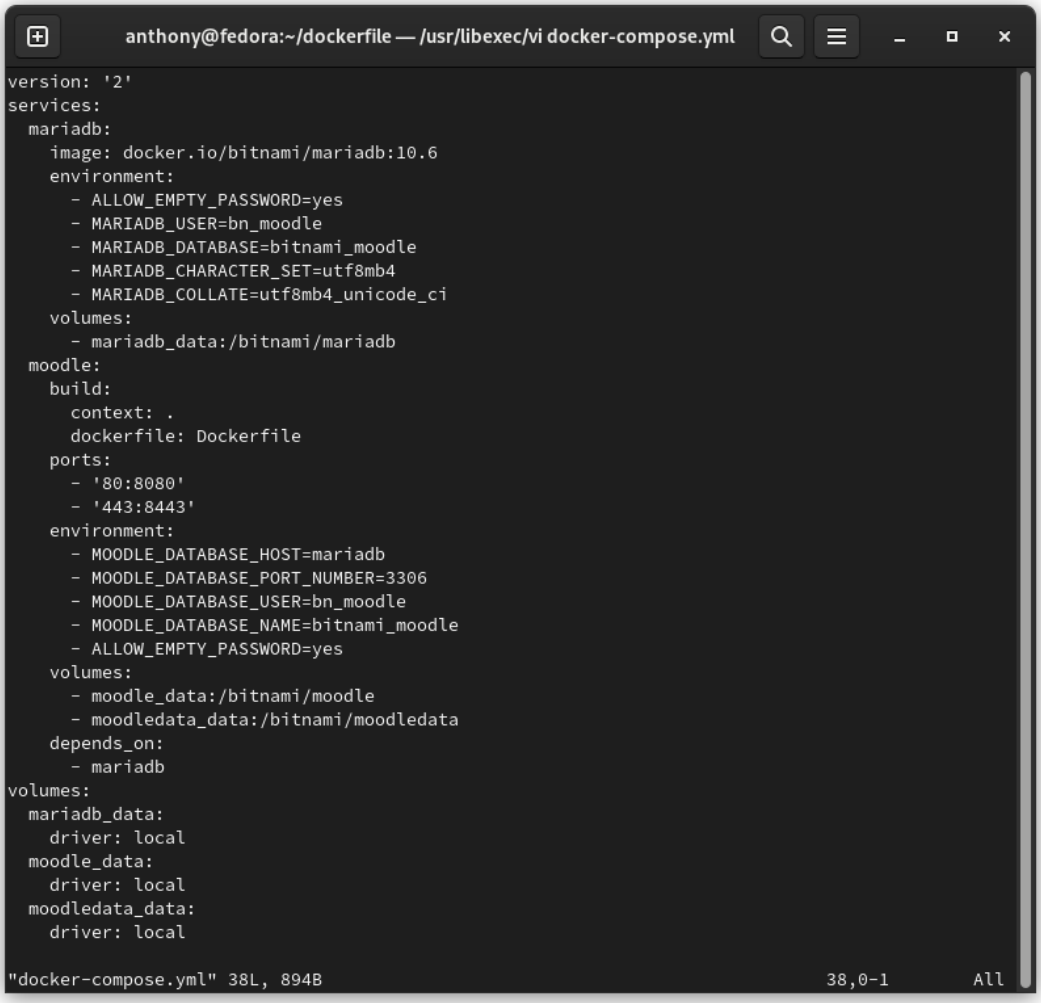


Figura 33. Archivo docker-compose que gestiona los contenedores

En cuanto al archivo docker-compose proporcionado en la figura 33, se han seguido las siguientes estrategias de optimización:

- Definición de servicios independientes: Se han definido dos servicios distintos, uno para el contenedor Bitnami/MariaDB y otro para el contenedor bitnami/moodle. Esto permite una gestión separada y específica de cada servicio, evitando la necesidad de incluir componentes innecesarios en cada contenedor.
- Configuración de volúmenes para persistencia de datos: Se han definido volúmenes específicos para almacenar los datos persistentes de los contenedores bitnami/moodle y Bitnami/MariaDB. Esto garantiza la persistencia de los datos entre reinicios de contenedores y facilita la gestión y el respaldo de los datos generados por las aplicaciones.
- Establecimiento de dependencias entre servicios: Se ha establecido una dependencia entre el servicio de bitnami/moodle y el servicio de Bitnami/MariaDB. Esto garantiza que el contenedor de Bitnami/MariaDB esté en funcionamiento antes de que se inicie el contenedor de bitnami/moodle, asegurando una conexión exitosa entre ambos.

La combinación del Dockerfile y el archivo docker-compose.yml en conjunto ofrece un entorno más optimizado para el despliegue de los contenedores de bitnami/moodle y Bitnami/MariaDB. A continuación, se explican cómo ambos archivos se complementan entre sí:

El Dockerfile personalizado define la construcción y configuración del contenedor de bitnami/moodle desde cero. Se han aplicado estrategias de optimización, como la selección cuidadosa de dependencias, la eliminación de componentes innecesarios y la configuración específica del rendimiento. Esto permite crear un contenedor de bitnami/moodle altamente optimizado y adaptado a las necesidades del entorno de pruebas.

Por otro lado, el archivo docker-compose.yml establece la configuración y la orquestación de los contenedores de bitnami/moodle y Bitnami/MariaDB. Define

cómo se comunican, qué servicios dependen de otros y cómo se gestionan los datos persistentes. Además, permite establecer las variables de entorno necesarias para que los contenedores interactúen correctamente. Esto asegura que los servicios estén correctamente interconectados y listos para su uso.

En conjunto, el Dockerfile personalizado y el archivo docker-compose.yml ofrecen un entorno optimizado y coordinado para el despliegue de los contenedores. Al construir el contenedor de bitnami/moodle desde cero, se garantiza una configuración específica y ajustada a los requerimientos del entorno de pruebas. Por su parte, el archivo docker-compose.yml permite gestionar y orquestar los contenedores de manera eficiente, asegurando su correcta comunicación y funcionalidad.

La combinación de ambos archivos proporciona un entorno más optimizado en términos de rendimiento, eficiencia y estabilidad. Permite reducir el tamaño del contenedor, evitar la inclusión de componentes innecesarios y establecer una configuración específica para cada servicio. Además, facilita la gestión y el mantenimiento del entorno de pruebas al tener una configuración centralizada y unificada.

d. Estrategias aplicadas sobre node.js

En esta sección se presentan las estrategias empleadas para optimizar el entorno de pruebas basado en Node. Estas estrategias se han aplicado con el objetivo de maximizar el rendimiento, la eficiencia y la escalabilidad del entorno de pruebas Node, lo que contribuye a obtener resultados óptimos en el desarrollo con contenedores.

Crear un Dockerfile personalizado desde cero de Node para utilizar exclusivamente la herramienta Nativefier ofrece varias ventajas en términos de optimización:

- Reducción del tamaño de la imagen: Al crear un Dockerfile personalizado, se pueden seleccionar y agregar solo los componentes necesarios para la ejecución de Nativefier. Esto permite eliminar cualquier paquete o biblioteca innecesaria que pueda aumentar el tamaño de la imagen. Como resultado, la imagen resultante será más liviana y consumirá menos espacio en el disco, lo que mejora la eficiencia y el rendimiento del contenedor.

- Minimización de dependencias: Al crear un Dockerfile personalizado, se puede especificar y gestionar las dependencias requeridas exclusivamente para la ejecución de Nativefier. Esto ayuda a evitar la inclusión de paquetes y bibliotecas adicionales que podrían no ser necesarios para el funcionamiento de la herramienta. Al reducir las dependencias, se mejora la eficiencia y se evitan posibles conflictos o problemas relacionados con las dependencias no utilizadas.

- Mayor control y personalización: Al crear un Dockerfile personalizado, se tiene un mayor control sobre la configuración del entorno de ejecución. Esto permite ajustar parámetros específicos de Nativefier, como variables de entorno, configuraciones de red, volúmenes, puertos, entre otros, según las necesidades del proyecto. Al personalizar el entorno de esta manera, se puede optimizar el rendimiento y el comportamiento de la herramienta de acuerdo con los requisitos del proyecto.

- Facilidad de mantenimiento y actualización: Al utilizar un Dockerfile personalizado, se puede mantener y actualizar fácilmente la imagen del contenedor de Nativefier. En lugar de depender de una imagen oficial general de Node que puede contener componentes adicionales o actualizaciones no deseadas, se puede mantener un mayor control sobre las actualizaciones y las dependencias específicas de Nativefier. Esto permite una gestión más eficiente de las actualizaciones y un mantenimiento simplificado a lo largo del ciclo de vida del proyecto.

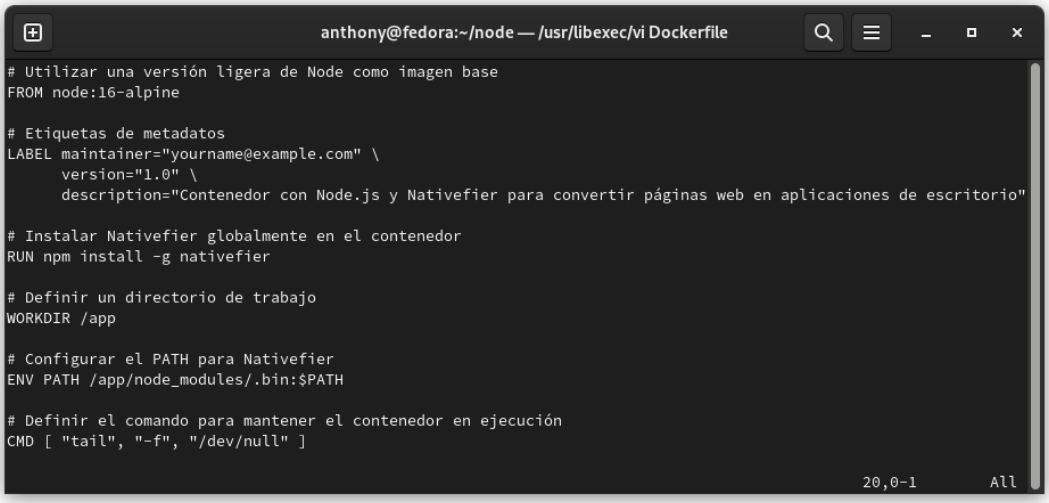


Figura 34. Figura 27. Dockerfile personalizado de node.js

Con relación al Dockerfile proporcionado en la figura 34, se han seguido las siguientes estrategias de optimización:

- Utilización de una imagen base ligera: Se optó por utilizar la imagen de Node:16-alpine, que es una versión más liviana de Node. Esto ayuda a reducir el tamaño total de la imagen y minimizar los recursos utilizados durante la ejecución.
- Etiquetas de metadatos: Se incluyeron etiquetas de metadatos en el Dockerfile para proporcionar información relevante sobre el mantenimiento, versión y descripción de la imagen. Esto facilita la identificación y gestión de la imagen en un entorno de desarrollo colaborativo.
- Instalación de Nativefier globalmente: Se utilizó el comando npm install -g nativefier para instalar la herramienta Nativefier de forma global en el contenedor. Esto garantiza que la herramienta esté disponible en cualquier ubicación del contenedor sin la necesidad de especificar rutas adicionales.
- Configuración del PATH: Se configuró el PATH del contenedor para incluir el directorio de los módulos de Node donde se encuentra el ejecutable de Nativefier (/app/node_modules/.bin). Esto permite ejecutar el comando nativefier directamente desde cualquier ubicación dentro del contenedor.

- Definición del comando por defecto: Se definió el comando tail -f /dev/null como el comando por defecto del contenedor. Esto mantiene el contenedor en ejecución sin realizar ninguna acción adicional. Esta estrategia es comúnmente utilizada para mantener el contenedor en un estado "activo" mientras se realizan otras operaciones o tareas.

En conjunto, estas estrategias de optimización permiten crear una imagen Docker eficiente y funcional para convertir páginas web en aplicaciones de escritorio utilizando Nativefier. Se maximiza el rendimiento al utilizar una imagen base ligera, se asegura la disponibilidad de Nativefier en el contenedor y se facilita su ejecución desde cualquier ubicación. Estas estrategias contribuyen a un entorno de desarrollo óptimo y eficiente.

e. Benchmarking de rendimiento

Evaluación de rendimiento de bitnami/moodle optimizado

En esta sección se realiza nuevamente el proceso descrito en la sección 3.1.8, "Implementación de benchmarking sobre bitnami/moodle", pero esta vez se aplica al nuevo contenedor creado. El objetivo es obtener nuevos datos de rendimiento y eficiencia que permitan realizar una comparación entre el contenedor anterior y el nuevo. Esto proporcionará información relevante para evaluar y analizar las mejoras logradas a través de la optimización del nuevo contenedor en comparación con el anterior.

Pruebas de carga y estrés con Siege

En esta etapa de la investigación, se realizarán pruebas de estrés en el nuevo contenedor de bitnami/moodle utilizando la herramienta "siege" con el objetivo de evaluar su rendimiento en un escenario de carga. Estas pruebas consisten en someter al servidor local a un alto número de solicitudes concurrentes, simulando una carga significativa. De esta manera, se podrá evaluar la capacidad del contenedor y del servidor para manejar esta carga y mantener un rendimiento óptimo.

Pruebas de estrés al servidor local

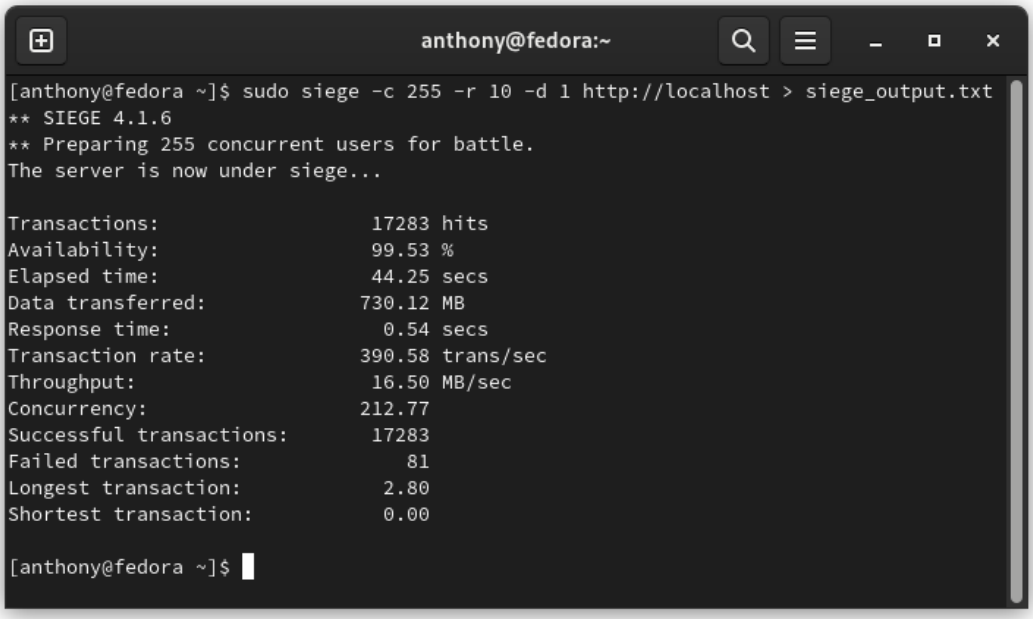


Figura 35. Primera prueba de estrés al servidor local

Durante el desarrollo de la investigación, se repitió el proceso en cinco ocasiones independientes, generando conjuntos de datos y resultados específicos en cada repetición. Estos resultados se presentan gráficamente en las figuras 35, 36, 37, 38 y 39, brindando una representación visual clara y concisa de los datos obtenidos en cada repetición.

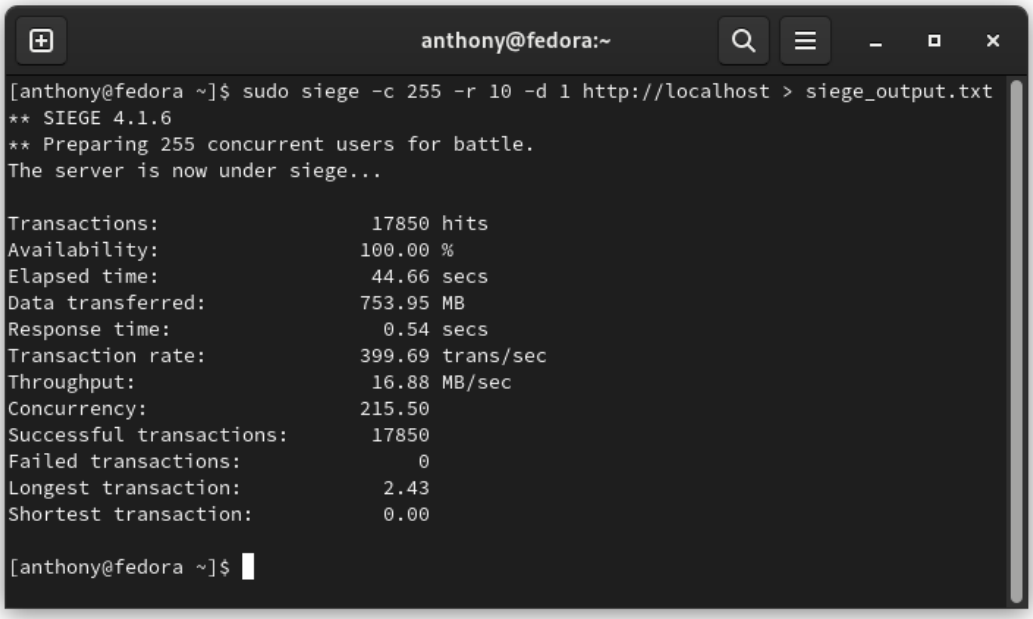


Figura 36. Segunda prueba de estrés al servidor local

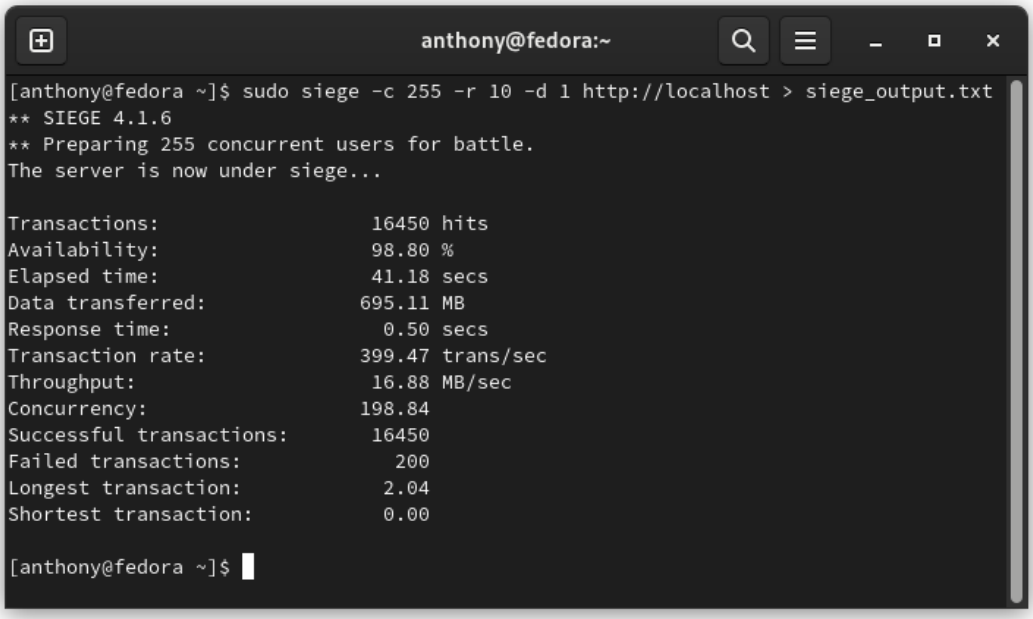


Figura 37. Tercera prueba de estrés al servidor local

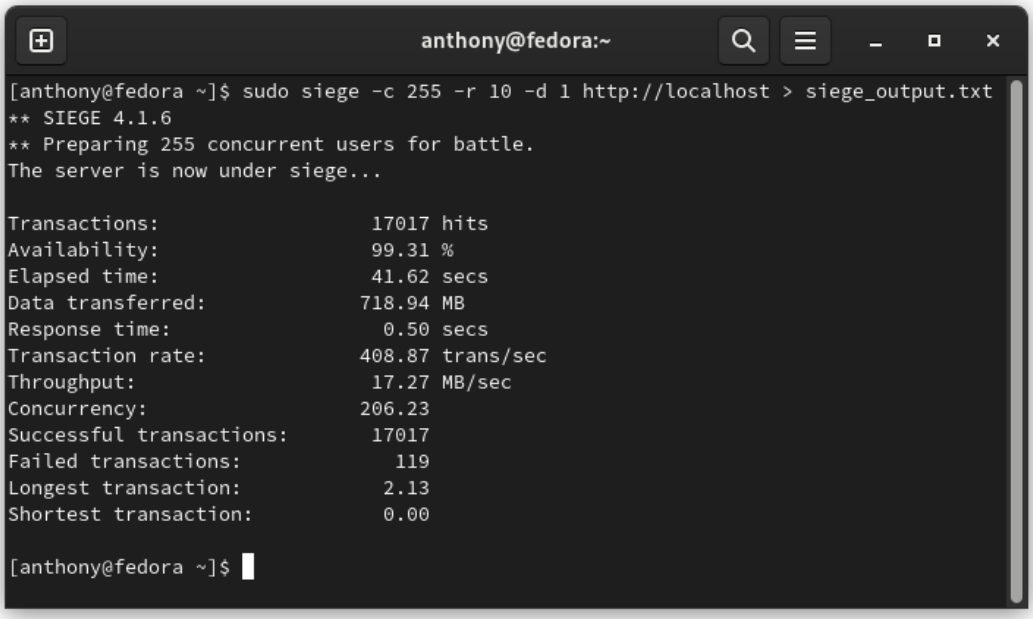


Figura 38. Cuarta prueba de estrés al servidor local

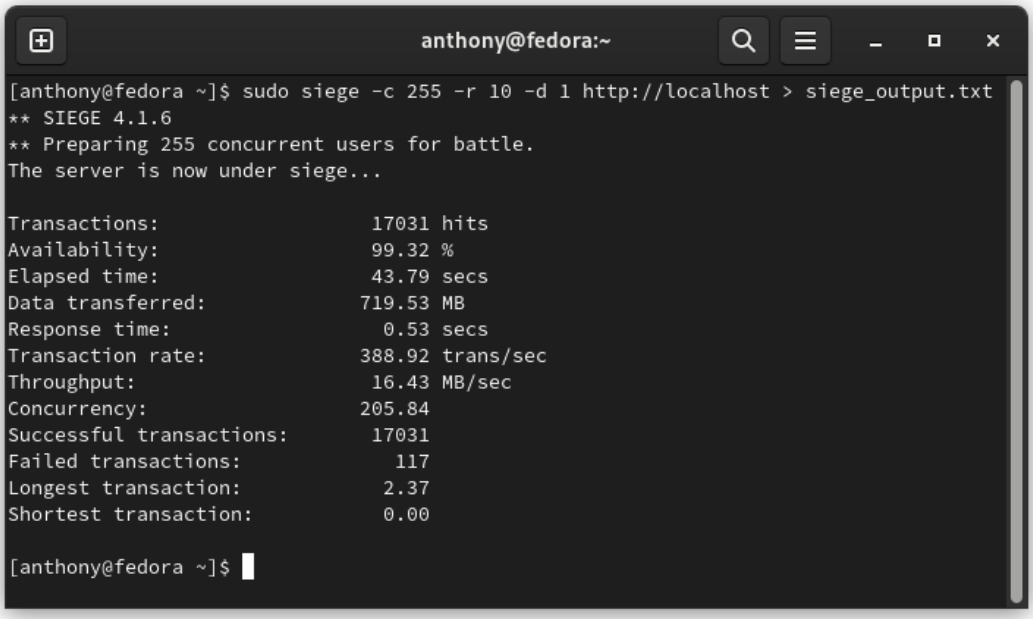


Figura 39. Quinta prueba de estrés al servidor local

Selección de datos para hacer comparativas

Entre los datos proporcionados, se seleccionaron nuevamente aquellos que fueron identificados como relevantes en la sección 3.1.8. Estos datos se consideran fundamentales para realizar comparaciones y contrastes con el contenedor anterior, permitiendo así obtener una evaluación precisa de los cambios y mejoras implementados en el nuevo contenedor.

- Availability
- Elapsed time
- Response time
- Transaction rate
- Throughput
- Concurrency

Estos datos desempeñan un papel fundamental en la comparación del rendimiento y la eficiencia de los distintos contenedores, ya que brindan información relevante sobre aspectos como la disponibilidad, la duración de las pruebas de carga, los tiempos de respuesta, la capacidad de procesamiento y la capacidad de transferencia de datos.

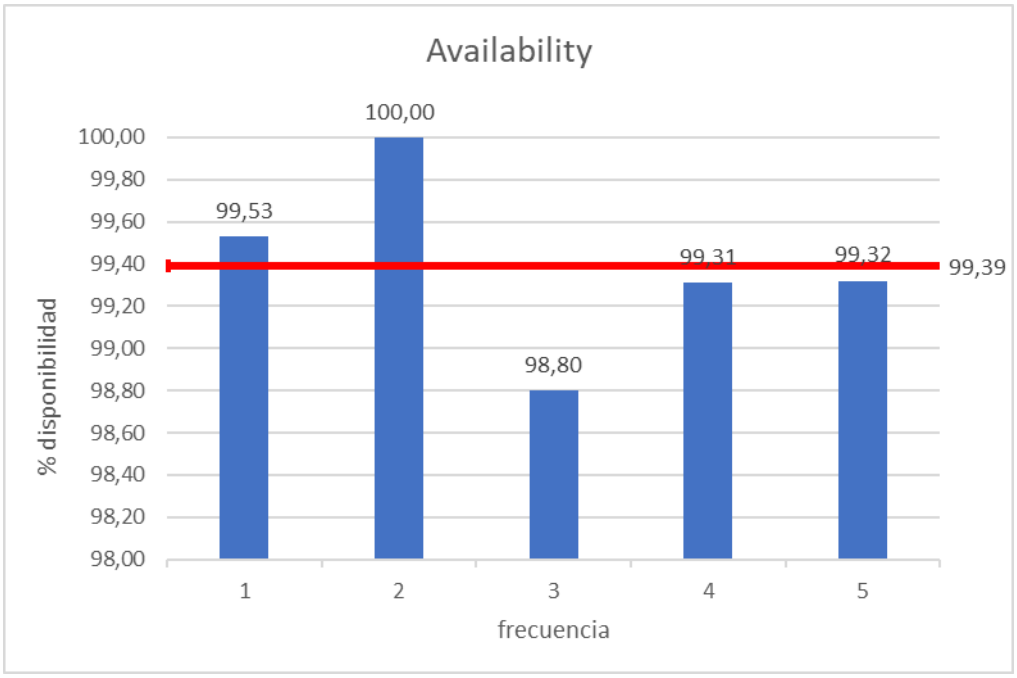


Figura 40. Gráfico con las estadísticas recolectadas de "Availability"

En base a la figura 40, se concluye que, durante las cinco ejecuciones de la prueba de estrés en el contenedor, se logró un promedio del 99,39% en el indicador de "disponibilidad" o "availability".

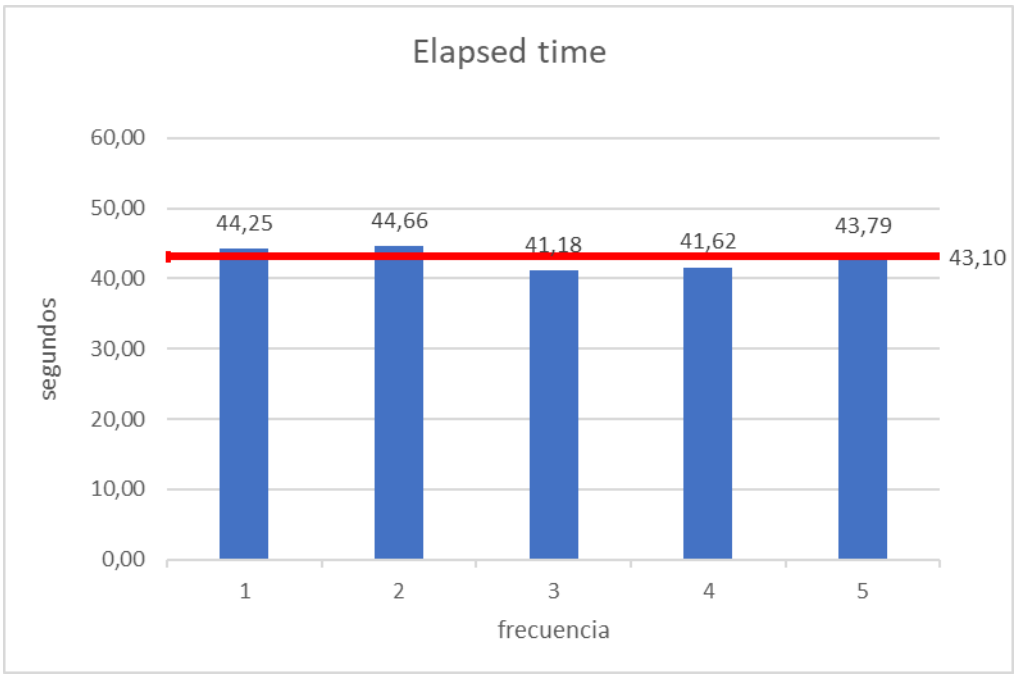


Figura 41. Gráfico con las estadísticas recolectadas de "Elapsed time"

En base a la figura 41, se pudo determinar que, durante las cinco ejecuciones de la prueba de estrés en el contenedor, se obtuvo un promedio de 43,10 segundos para el indicador de "elapsed time" o "tiempo transcurrido".

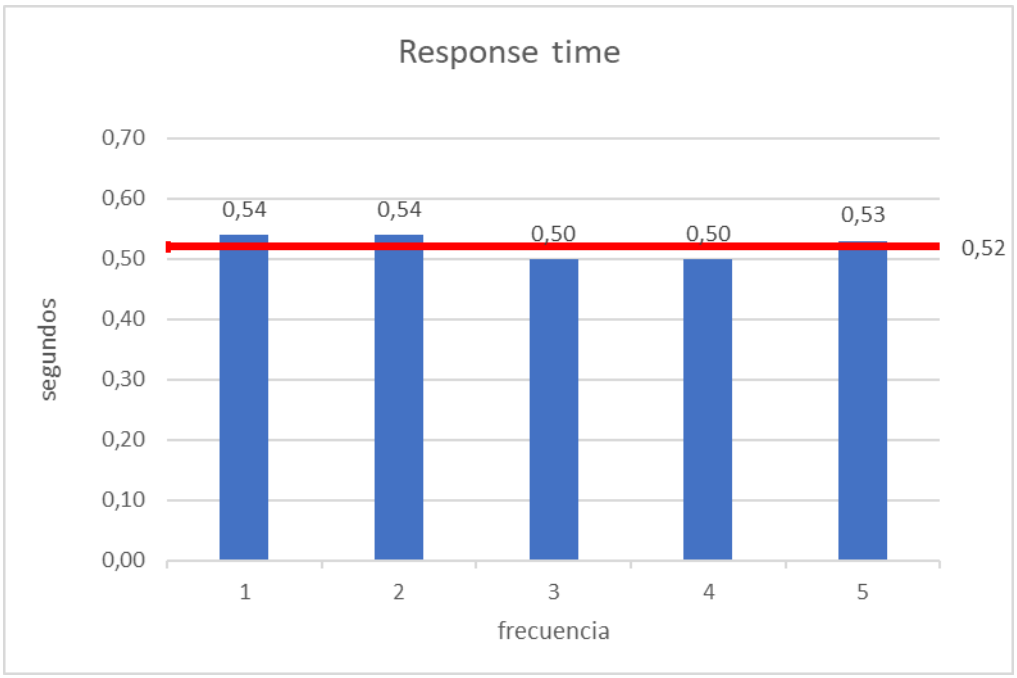


Figura 42. Gráfico con las estadísticas recolectadas de "Response time"

En base a la figura 42, se determinó que, durante las cinco ejecuciones de la prueba de estrés en el contenedor, se obtuvo un promedio de 0,52 segundos para el indicador de "response time" o "tiempo de respuesta".

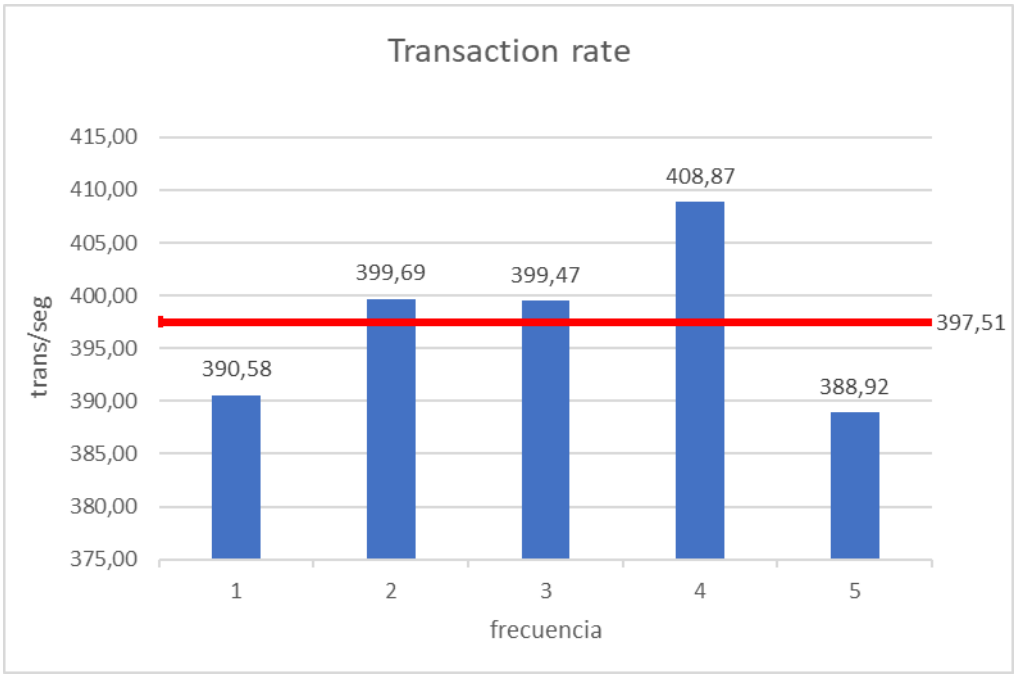


Figura 43. Gráfico con las estadísticas recolectadas de "Transaction rate"

En base a la figura 43, se determinó que, durante las cinco ejecuciones de la prueba de estrés en el contenedor, se obtuvo un promedio de 397,51 transacciones por segundo para el indicador de "transaction rate" o "tasa de transacciones".

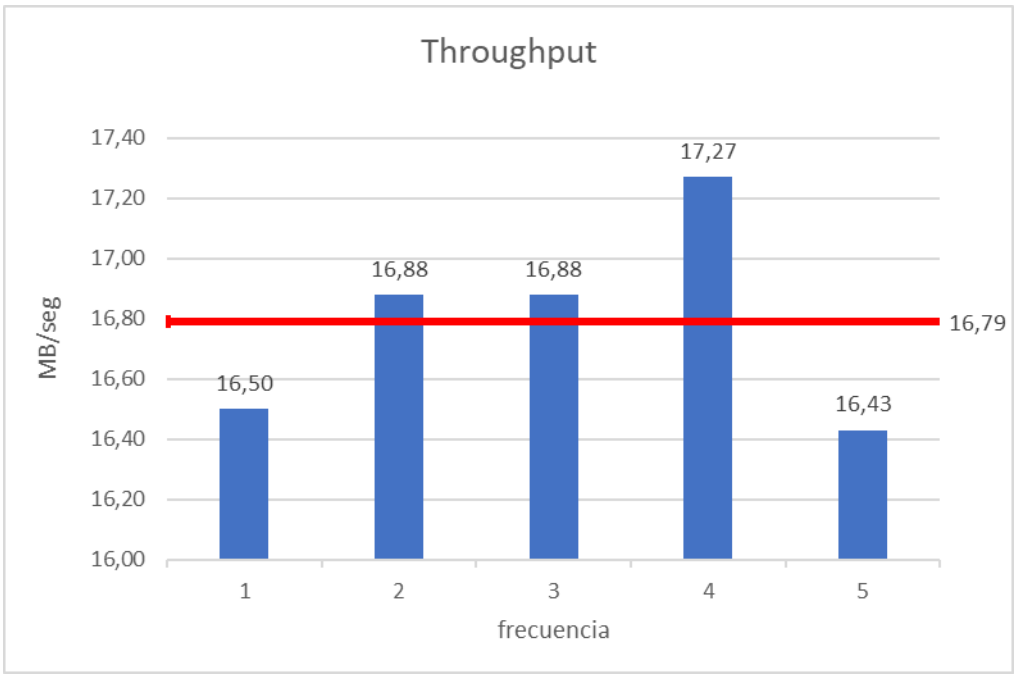


Figura 44. Gráfico con las estadísticas recolectadas de "Throughput"

En base a la figura 44, se determinó que, durante las cinco ejecuciones de la prueba de estrés en el contenedor, se obtuvo un promedio de 16,79 megabytes por segundo (MB/seg) para el indicador de "throughput" o "rendimiento de transferencia".

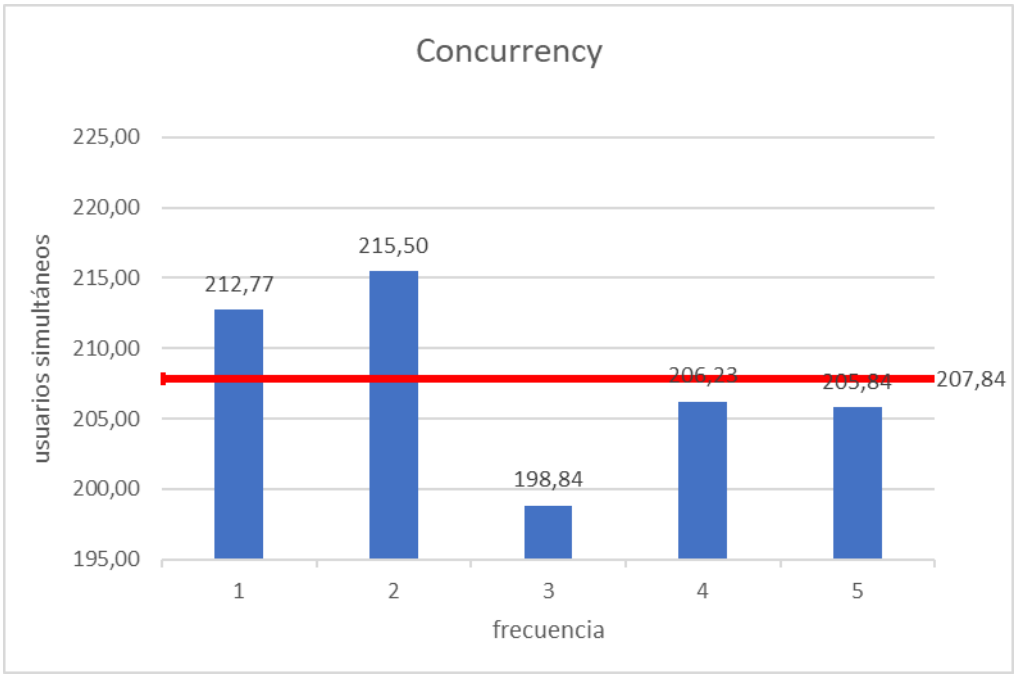


Figura 45. Gráfico con las estadísticas recolectadas de "Concurrency"

En base a la figura 45, se determinó que, durante las cinco ejecuciones de la prueba de estrés en el contenedor, se obtuvo un promedio de 207,84 usuarios simultáneos para el indicador de "concurrency" o "concurrencia".

Monitoreo del sistema con Stacer mientras se ejecuta Siege

Figura 46. Monitoreo del sistema durante las pruebas de estrés

Monitoreo del contenedor con docker stats mientras se ejecuta Siege

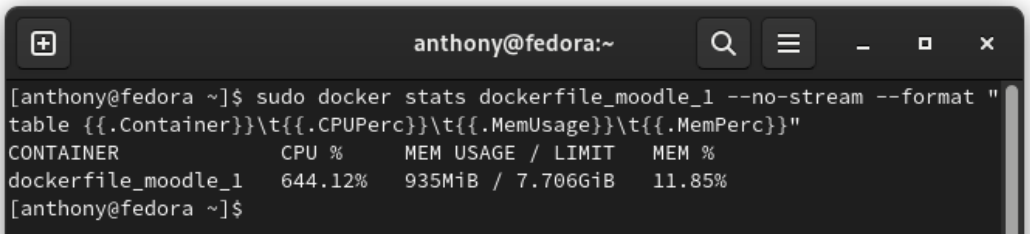


Figura 47. Monitoreo del contenedor en la 1ra prueba de estrés al servidor local

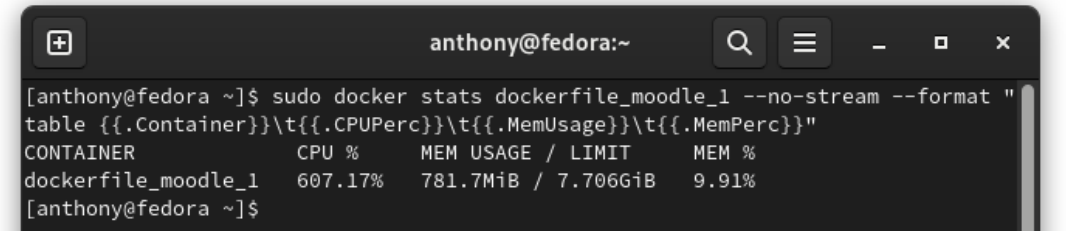


Figura 48. Monitoreo del contenedor en la 2da prueba de estrés al servidor local

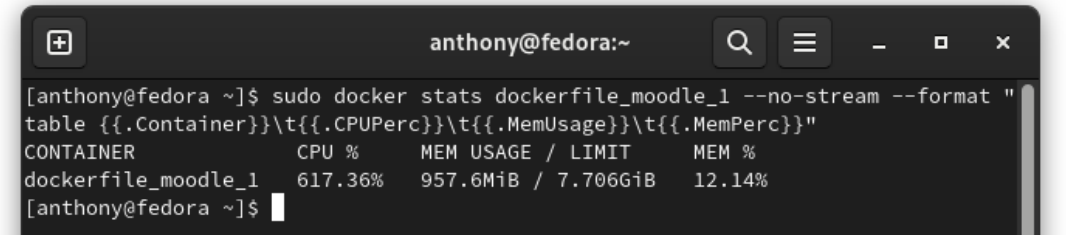


Figura 49. Monitoreo del contenedor en la 3ra prueba de estrés al servidor local

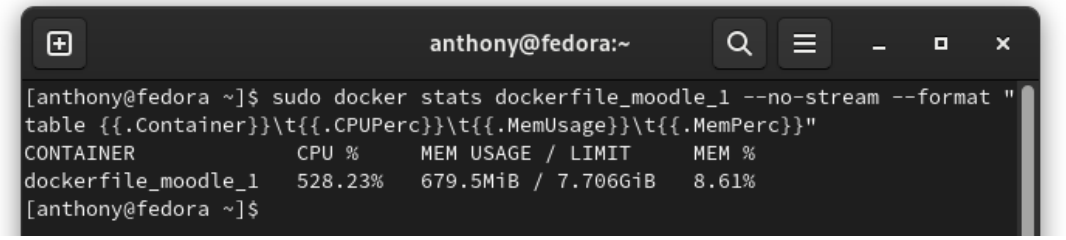


Figura 50. Monitoreo del contenedor en la 4ta prueba de estrés al servidor local

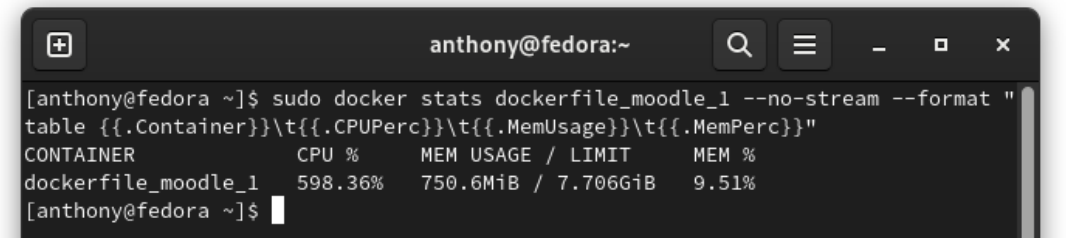


Figura 51. Monitoreo del contenedor en la 5ta prueba de estrés al servidor local

Datos para hacer comparativas

En base a la sección 3.1.8, se han seleccionado nuevamente los mismos conjuntos de datos, ya que son pertinentes para realizar comparativas con el contenedor anterior. Estos datos proporcionan información detallada sobre el uso de la CPU y la memoria de los contenedores, lo cual es esencial para evaluar el rendimiento y la eficiencia de ambos entornos.

- CPU %
- MEM USAGE / LIMIT
- MEM %

Al utilizar los mismos conjuntos de datos, se garantiza la consistencia en las mediciones y se facilita la comparación directa entre los contenedores, brindando una base sólida para realizar análisis precisos y concluyentes.

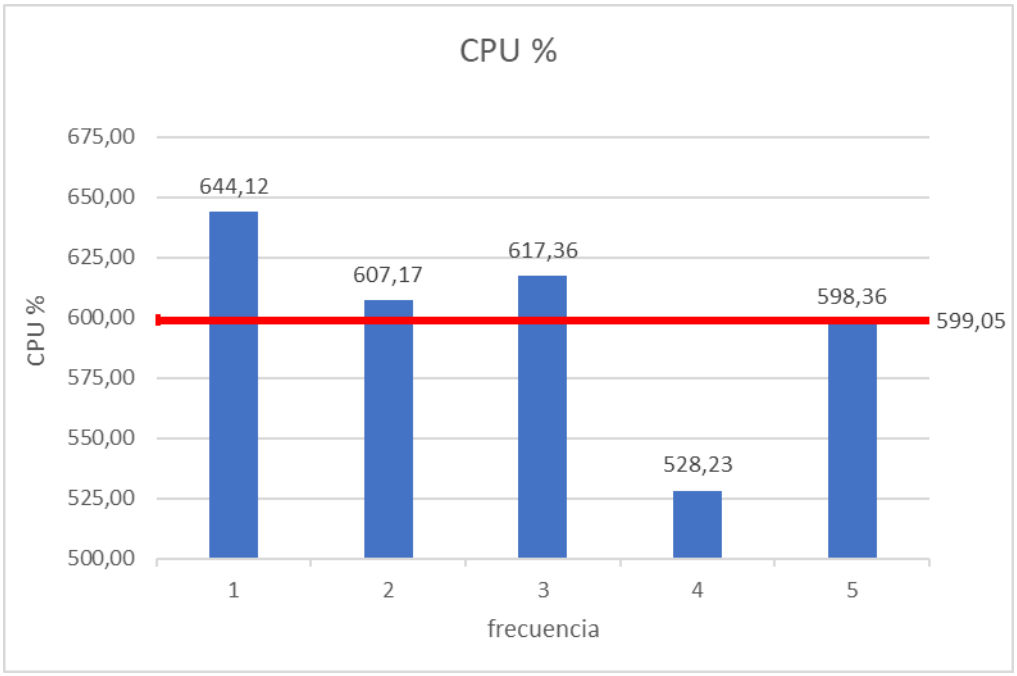


Figura 52. Gráfico con las estadísticas recolectadas de "CPU %"

En base a la figura 52, se determinó que durante las cinco ocasiones en que se ejecutó la prueba de estrés en el contenedor, se obtuvo un promedio de 599,05 % de uso de

CPU para el apartado "CPU %".

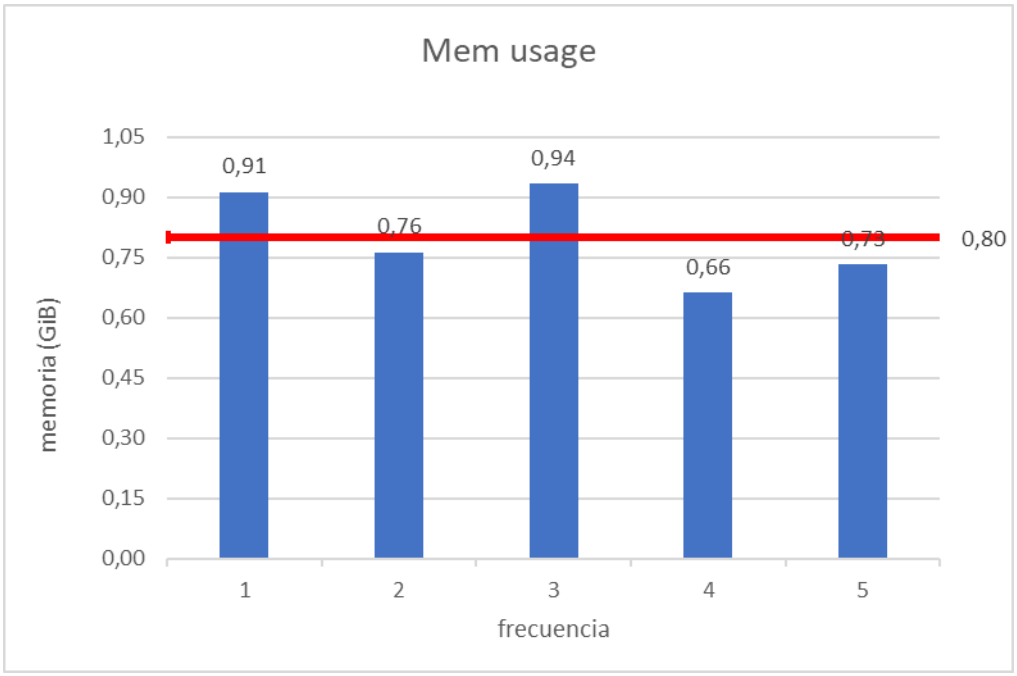


Figura 53. Gráfico con las estadísticas recolectadas de "Mem usage"

En base a la figura 53, se determinó que durante las cinco ocasiones en que se ejecutó la prueba de estrés en el contenedor, se obtuvo un promedio de 0,80 GiB de memoria utilizada por el contenedor para el apartado "mem usage".

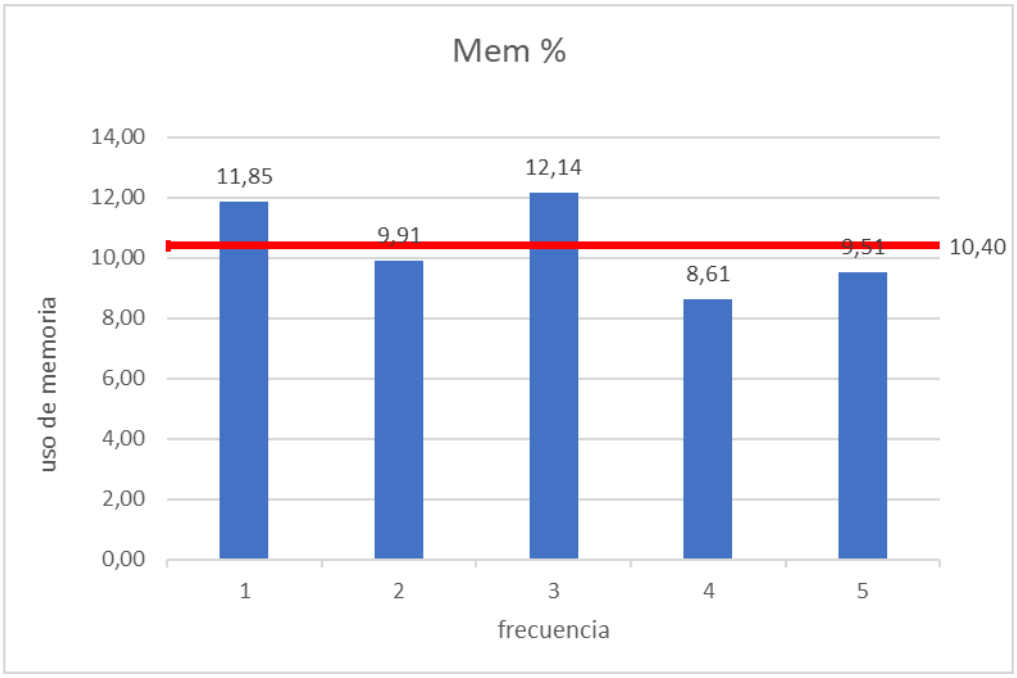


Figura 54. Gráfico con las estadísticas recolectadas de "Mem %"

En base a la figura 54, se determinó que durante las cinco ocasiones en que se ejecutó la prueba de estrés en el contenedor, se obtuvo un promedio de 10,40 % de uso de memoria para el apartado "mem %".

Evaluación de rendimiento de node.js optimizado

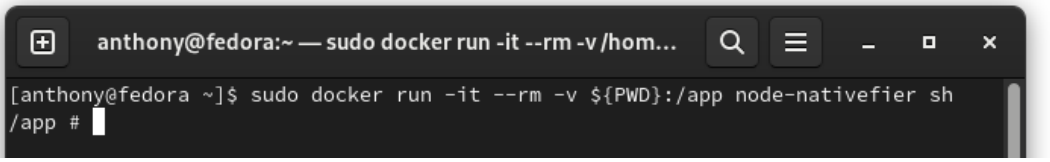


Figura 55. Comando para ejecutar el contenedor

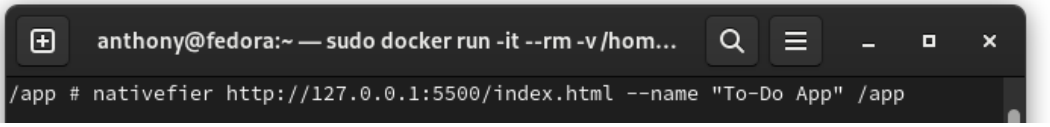


Figura 56. Conversión de la página web en aplicación de escritorio

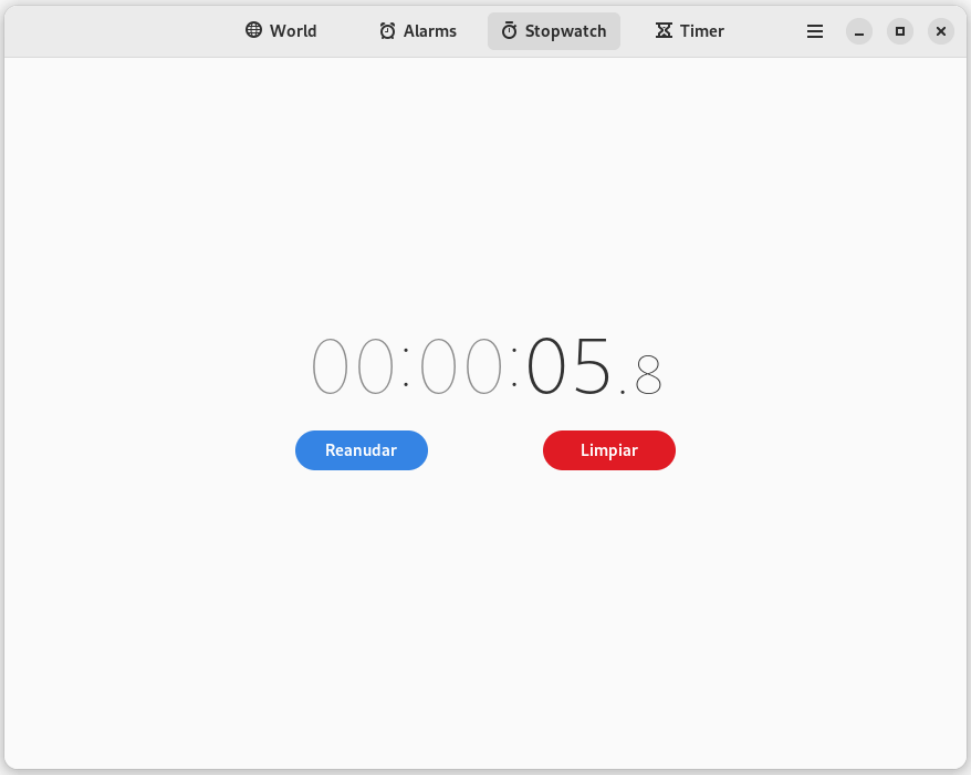


Figura 57. Tiempo de conversión de la página web a aplicación de escritorio

La figura 28 muestra el tiempo necesario para convertir la página web en una aplicación de escritorio utilizando nativefier. En este caso, se registra un intervalo de aproximadamente 05.8 segundos desde que se ejecuta el comando correspondiente hasta que se completa la transformación. Este dato es relevante ya que proporciona una indicación del rendimiento y la eficiencia del proceso de conversión.

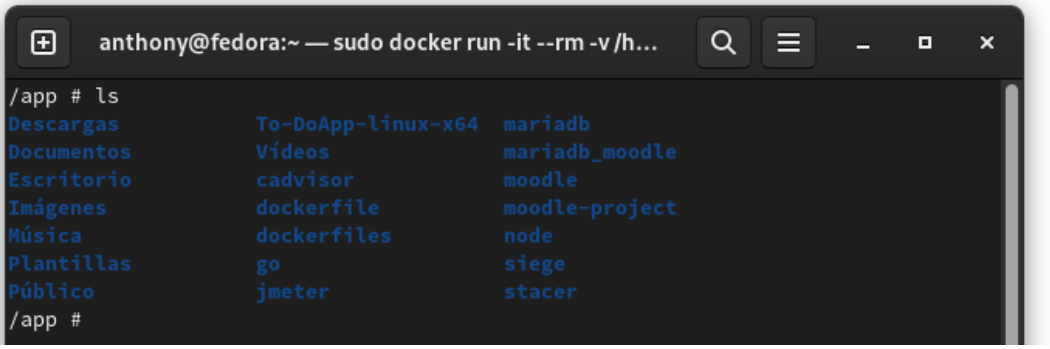


Figura 58. Aplicación de escritorio generada por nativefier

En la figura 58 se puede observar la generación de un directorio llamado "To-DoApp-linux-x64" dentro del contenedor de Node.js. Este directorio específico es el lugar donde se almacena la aplicación de escritorio que ha sido creada utilizando nativefier.

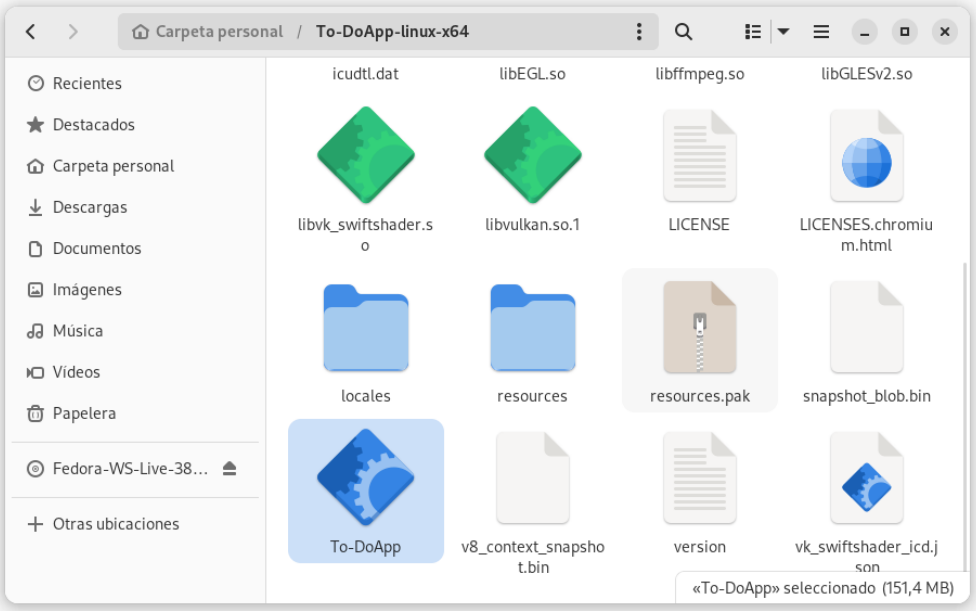


Figura 59. Vistazo a la aplicación de escritorio en el gestor de archivos de Fedora

La figura 30 muestra una vista del directorio donde se encuentra almacenada la aplicación de escritorio en el gestor de archivos de Fedora. En este caso específico, el directorio se ha nombrado como "APP".

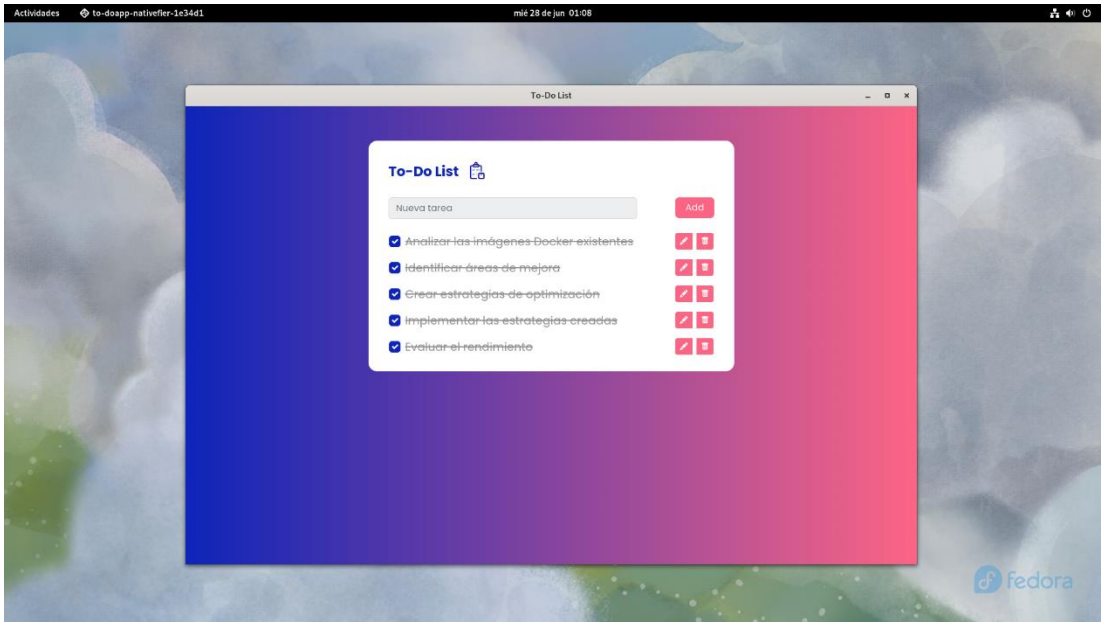


Figura 60. Aplicación ejecutada en Fedora

En la figura 60 se muestra la implementación y el rendimiento de la aplicación de escritorio creada con nativefier. La imagen representa el funcionamiento exitoso de la aplicación, sin enfrentar problemas de compatibilidad con el sistema operativo.

f. Análisis de resultados

Análisis del tamaño de las imágenes Docker

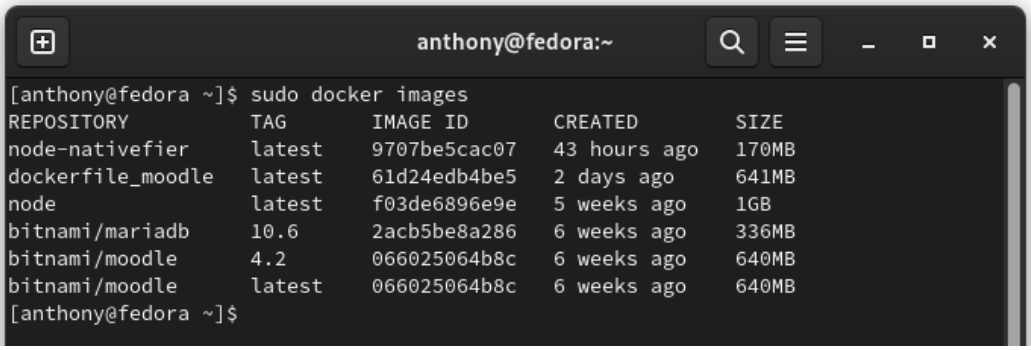


Figura 61. Listado de las imágenes Docker disponibles en el sistema

Utilizando los datos recopilados de acuerdo con la figura 61, se llevó a cabo un análisis comparativo para examinar y extraer conclusiones pertinentes. A través de la creación

de tablas comparativas, se buscó estudiar en detalle la información recolectada y realizar un análisis profundo de los resultados obtenidos.

Tabla 17. Comparación del tamaño de las imágenes Docker

Tamaño de las imágenes Docker disponibles		
	Imagen no optimizada	**Imagen optimizada**
Imagen moodle	640 MB	641 MB
Imagen mariadb	336 MB	-
Imagen node.js	1 GB	170 MB

Autor: Anthony López

Tras analizar los datos presentados en la tabla 17, se puede deducir lo siguiente:

La imagen optimizada de moodle muestra un ligero aumento en su tamaño en comparación con la no optimizada, debido a que se enfocó en optimizar aspectos internos de la imagen, como la estructura de capas y la eliminación de dependencias innecesarias. Esto indica que la optimización interna puede tener un impacto positivo en el rendimiento y la eficiencia del contenedor de moodle.

En el caso de la imagen de mariadb, no se desarrolló una nueva imagen optimizada, ya que se utilizó la misma imagen para desplegar ambos contenedores. Por lo tanto, no se proporciona información sobre el tamaño de la imagen optimizada en este contexto.

La optimización del contenedor de node.js se centró en la creación de un entorno exclusivo para ejecutar la herramienta nativefier. Como resultado, se logró reducir significativamente el tamaño de la imagen, pasando de 1 GB a 170 MB. Esta reducción en el tamaño indica que se aplicaron estrategias de optimización, como la eliminación de dependencias innecesarias y la selección de una imagen base más liviana. Esta optimización específica para el contenedor de node.js busca mejorar el rendimiento y la eficiencia al utilizar la herramienta Nativefier.

En resumen, los datos revelan que la optimización de imágenes Docker puede tener diferentes enfoques y resultados según el contenedor en cuestión. La optimización interna en moodle y la optimización específica para node.js demuestran el impacto positivo que pueden tener en el rendimiento y la eficiencia de los contenedores en sus respectivos contextos de uso.

Análisis de los contenedores Docker

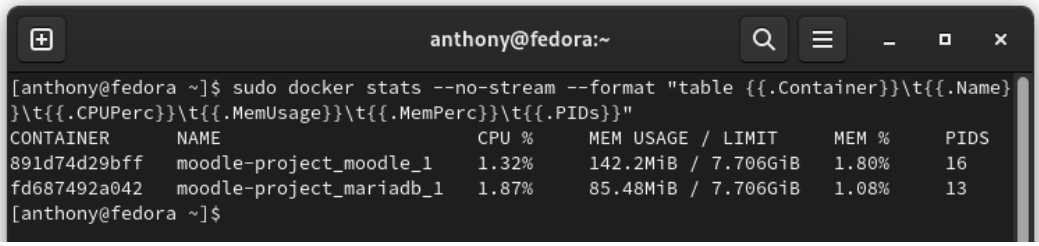


Figura 62. Recursos utilizados por los contenedores "no optimizados"

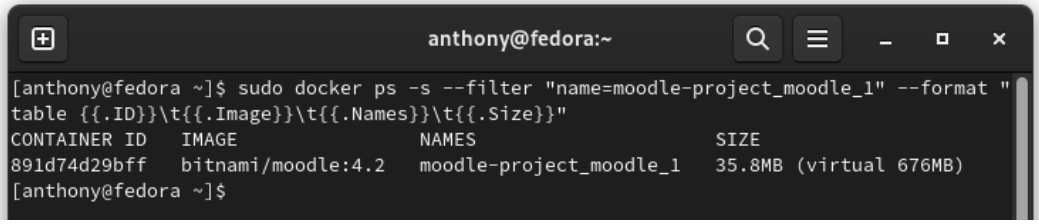


Figura 63. Tamaño del contenedor moodle "no optimizado"

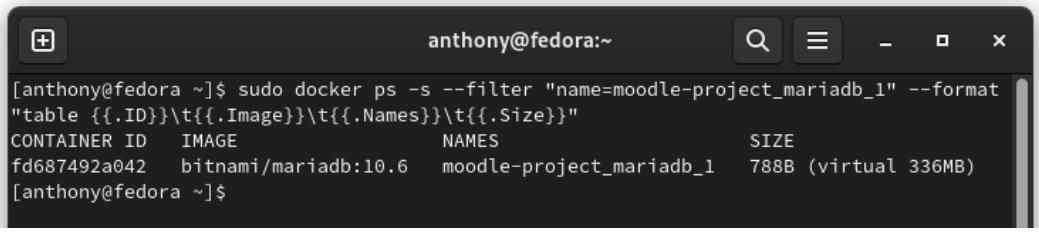


Figura 64. Tamaño del contenedor mariadb "no optimizado"

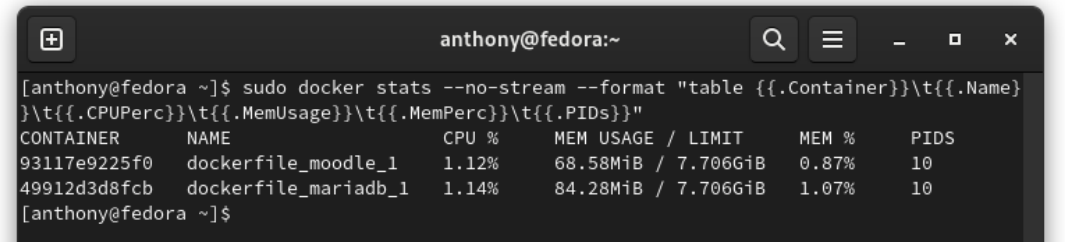


Figura 65. Recursos utilizados por los contenedores "optimizados"

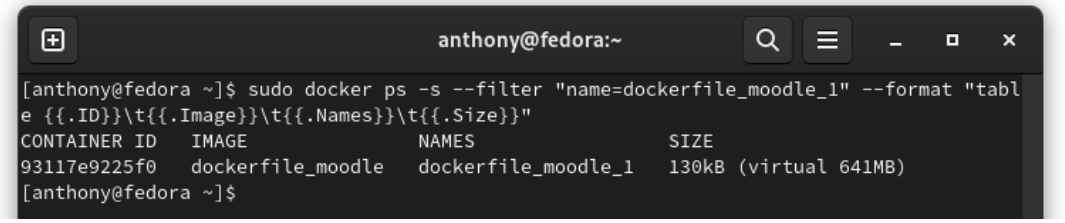


Figura 66. Tamaño del contenedor moodle "optimizado"

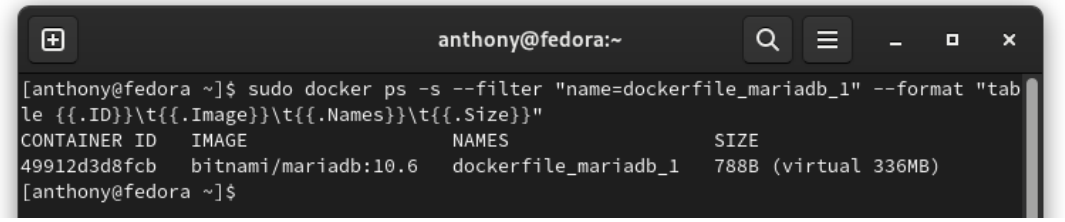


Figura 67. Tamaño del contenedor mariadb "optimizado"

Basándose en los datos recopilados según las figuras 62, 63, 64, 65, 66 y 67, se realizó un análisis comparativo con el objetivo de examinar y obtener conclusiones relevantes. Mediante la creación de tablas de comparación, se buscó estudiar la información recolectada y realizar un análisis de los resultados obtenidos.

Tabla 18. Comparación de recursos usados por el contenedor Moodle

Estadísticas de uso de recursos del contenedor moodle		
	Contenedor no optimizado	**Contenedor optimizado**
CPU%	1,32%	1,12%
Mem usage	142,2 MiB	68,58 MiB
Mem%	1,80%	0,87%
PIDS	16	10

Autor: Anthony López

Tras analizar los datos presentados en la tabla 18, se puede deducir lo siguiente:

En cuanto al uso de la CPU, se observa una ligera reducción en el porcentaje de uso en el contenedor optimizado en comparación con el contenedor no optimizado. Esto indica que la optimización implementada ha logrado una mejor gestión de los recursos de la CPU, lo que puede contribuir a un rendimiento más eficiente y una mayor capacidad de respuesta.

En términos de uso de memoria, se evidencia una notable disminución tanto en la cantidad absoluta de memoria utilizada como en el porcentaje de uso en el contenedor optimizado. Esta reducción indica que se ha logrado una mejor gestión de la memoria, lo que puede resultar en un uso más eficiente de los recursos y una mayor estabilidad del contenedor.

En relación a los PIDs (identificadores de proceso), se observa una disminución en el número de procesos en el contenedor optimizado en comparación con el contenedor no optimizado. Esto sugiere una optimización en la gestión de los procesos, lo que puede contribuir a un mejor rendimiento y una mayor estabilidad del contenedor.

En resumen, los datos obtenidos de la comparativa entre el contenedor no optimizado y el contenedor optimizado revelan mejoras significativas en el uso de recursos, tanto en la CPU como en la memoria. Además, se observa una optimización en la gestión

de los procesos. Estos resultados indican que la implementación de estrategias de optimización ha llevado a un mejor rendimiento y eficiencia en el contenedor optimizado.

Tabla 19. Comparación de recursos usados por el contenedor mariadb

Estadísticas de uso de recursos del contenedor mariadb		
	Contenedor no optimizado	**Contenedor optimizado**
CPU%	1,87%	1,14%
Mem usage	85,48 MiB	84,28 MiB
Mem%	1,08%	1,07%
PIDS	13	10

Autor: Anthony López

Tras analizar los datos proporcionados en la tabla 19, se puede deducir lo siguiente:

En relación al uso de la CPU, se observa que el contenedor optimizado presenta un porcentaje ligeramente más bajo en comparación con el contenedor no optimizado. Esto indica que la implementación de estrategias de optimización ha permitido una mejor gestión de los recursos de la CPU, lo que puede resultar en un rendimiento más eficiente y una mayor capacidad de respuesta del contenedor.

En cuanto al uso de la memoria, se aprecia una diferencia mínima entre ambos contenedores. Ambos presentan un nivel de uso de memoria muy similar, lo que sugiere que la optimización realizada no ha tenido un impacto significativo en este aspecto.

En relación al número de PIDs (identificadores de procesos), se observa que el contenedor optimizado tiene un menor número de procesos en comparación con el contenedor no optimizado. Esta reducción en los PIDs indica una optimización en la gestión de los procesos, lo que puede contribuir a un mejor rendimiento y una mayor estabilidad del contenedor optimizado.

En resumen, los datos obtenidos indican que el contenedor optimizado ha logrado una mejor eficiencia en el uso de la CPU y una optimización en la gestión de los procesos en comparación con el contenedor no optimizado. Aunque no se observan diferencias significativas en el uso de la memoria, la mejora en el rendimiento y la estabilidad del contenedor optimizado son aspectos destacables de las estrategias de optimización implementadas.

Tabla 20. Comparación de consumo de espacio en disco por el contenedor moodle

Estadísticas de consumo de espacio en disco del contenedor de moodle		
	Contenedor no optimizado	**Contenedor optimizado**
SIZE	35,8 MB	130 kB

Autor: Anthony López

Después de analizar los datos presentados en la tabla 20, se puede deducir lo siguiente:

En relación al consumo de espacio en disco, se observa una diferencia significativa entre el contenedor no optimizado y el contenedor optimizado de Moodle. El contenedor no optimizado ocupa un tamaño de 35,8 MB, mientras que el contenedor optimizado ha logrado reducir considerablemente su tamaño a tan solo 130 kB.

Estos resultados demuestran que las estrategias de optimización implementadas en el contenedor de Moodle han sido efectivas en la reducción del espacio en disco utilizado. Esta optimización puede resultar en ventajas como un uso más eficiente de los recursos de almacenamiento y una reducción de los tiempos de despliegue y transferencia de la imagen del contenedor.

En resumen, el contenedor optimizado de Moodle ha logrado reducir de manera significativa el consumo de espacio en disco en comparación con el contenedor no optimizado. Esta optimización contribuye a una mejor gestión de los recursos y a una mayor eficiencia en el despliegue y uso de la imagen del contenedor.

Tabla 21. Comparación de consumo de espacio en disco por el contenedor mariadb

Estadísticas de consumo de espacio en disco del contenedor de mariadb		
	Contenedor no optimizado	**Contenedor optimizado**
SIZE	788 B	788 B

Autor: Anthony López

Después de analizar los datos presentados en la tabla 21, se puede deducir lo siguiente:

En relación al consumo de espacio en disco, se observa que tanto el contenedor no optimizado como el contenedor optimizado de MariaDB tienen un tamaño igual de 788 B. Esta igualdad en el tamaño se debe a que no se desarrolló una nueva imagen optimizada específicamente para el contenedor de MariaDB, utilizando la misma imagen para ambos contenedores.

Aunque no se han logrado reducir el tamaño del contenedor de MariaDB, es importante destacar que la optimización se centró en otros aspectos relevantes para el rendimiento y eficiencia del contenedor, como la configuración de recursos, ajustes de rendimiento y seguridad.

En resumen, si bien no se observa una diferencia en el tamaño del contenedor de MariaDB entre el enfoque no optimizado y el optimizado, es importante considerar que la optimización se centró en otros aspectos clave para el rendimiento y eficiencia del contenedor. Esto demuestra la importancia de abordar diferentes aspectos en la optimización de contenedores, no solo el tamaño en disco.

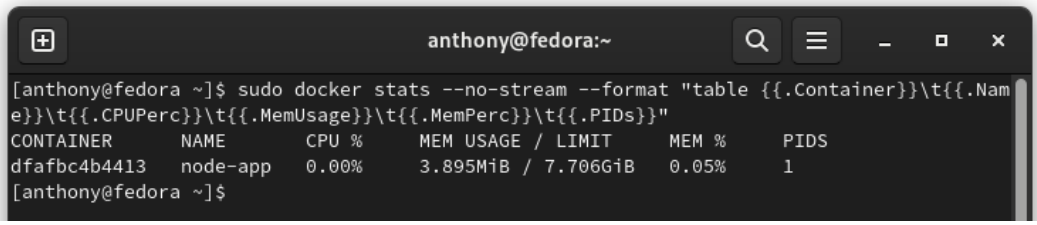


Figura 68. Recursos utilizados por el contenedor node "no optimizado"

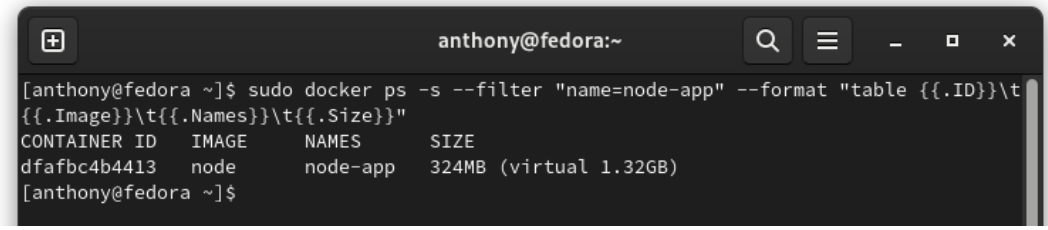


Figura 69. Tamaño del contenedor node "no optimizado"

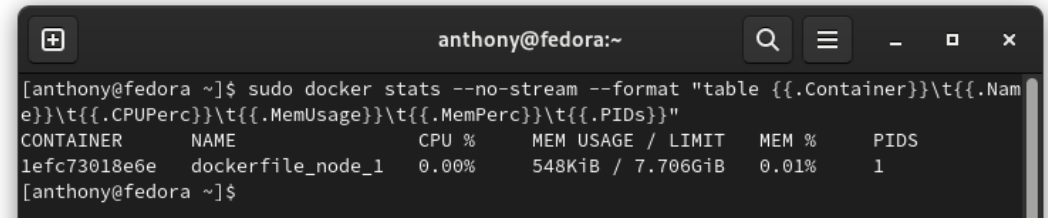


Figura 70. Recursos utilizados por el contenedor node "optimizado"

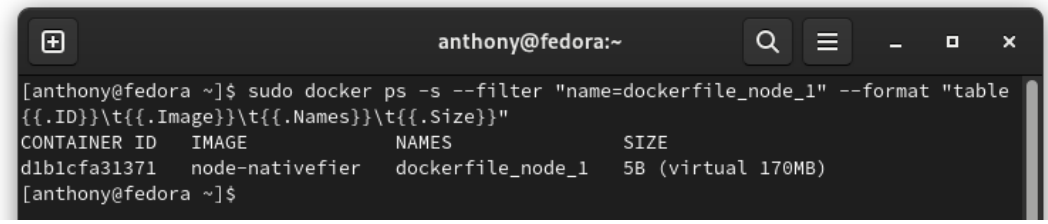


Figura 71. Tamaño del contenedor node "optimizado"

Basándose en los datos recopilados según las figuras 68, 69, 70 y 71, se realizó un análisis comparativo detallado con el objetivo de extraer conclusiones relevantes. Mediante la creación de tablas comparativas, se examinó la información recolectada y se llevó a cabo un análisis de los resultados obtenidos.

Tabla 22. Comparación de recursos usados por el contenedor node.js

Estadísticas de uso de recursos del contenedor de node.js		
	Contenedor no optimizado	**Contenedor optimizado**
CPU%	0,00%	0,00%
Mem usage	3,895 MiB	548 KiB
Mem%	0,05%	0,01%
PIDS	1	1

Autor: Anthony López

Tras analizar los datos presentados en la tabla 22, se puede deducir lo siguiente:

En cuanto al uso de recursos del contenedor de node.js, se observa que tanto el contenedor no optimizado como el contenedor optimizado presentan valores muy bajos de consumo de CPU y memoria. En ambos casos, el porcentaje de uso de CPU es de 0,00%, lo que indica un bajo nivel de carga en el procesamiento. En cuanto al consumo de memoria, se aprecia una diferencia significativa, siendo el contenedor no optimizado de 3,895 MiB y el contenedor optimizado de 548 KiB. Esto refleja una mejora sustancial en el uso de memoria, donde el contenedor optimizado logra una eficiencia notablemente mayor.

Además, se observa que ambos contenedores tienen un único proceso en ejecución, lo cual indica una correcta administración de los procesos dentro del contenedor.

En resumen, los resultados demuestran que el contenedor optimizado de node.js logra un uso más eficiente de los recursos, con un menor consumo de memoria en comparación con el contenedor no optimizado.

Tabla 23. Comparación de consumo de espacio en disco por el contenedor node.js

Estadísticas de consumo de espacio en disco del contenedor de node.js		
	Contenedor no optimizado	**Contenedor optimizado**
SIZE	324 MB	5 B

Autor: Anthony López

Al analizar los datos presentados en la tabla 23, se puede deducir lo siguiente:

En primer lugar, se observa una diferencia significativa en el tamaño de los contenedores. Mientras que el contenedor no optimizado ocupa 324 MB en disco, el contenedor optimizado utiliza tan solo 5 B de espacio. Esta reducción drástica en el consumo de espacio demuestra la eficacia de las estrategias de optimización implementadas en el contenedor optimizado.

Es importante destacar que la optimización del espacio en disco contribuye a un uso más eficiente de los recursos y permite un mejor desempeño general del contenedor. Al ocupar menos espacio en disco, se mejora la escalabilidad y la eficiencia en el despliegue y ejecución de la aplicación.

En resumen, el contenedor optimizado de node.js logra una optimización significativa en el consumo de espacio en disco, lo cual resulta en un uso más eficiente de los recursos y una mejora en el desempeño del contenedor.

Análisis de resultados de benchmarking en los contenedores Docker

Utilizando los datos recolectados a través del benchmarking realizado con diferentes herramientas y técnicas en los contenedores "no optimizados" y "optimizados", se llevó a cabo un análisis comparativo con el propósito de extraer conclusiones pertinentes. Para facilitar el análisis de los resultados obtenidos, se elaboraron tablas comparativas que permitieron una evaluación detallada de los datos.

Tabla 24. Comparación de pruebas de estrés al servidor local

Pruebas de estrés al servidor local		
	Contenedor no optimizado	**Contenedor optimizado**
Availability	99,48 %	99,39 %
Elapsed time	50,32 s	43,10 s
Response time	0,62 s	0,52 s
Transaction rate	345,1 trans/s	397,51 trans/s
Throughput	14,58 MB/s	16,79 MB/s
Concurrency	213,34 usuarios sim.	207,84 usuarios sim.

Autor: Anthony López

Al analizar los datos presentados en la tabla 24, se puede deducir lo siguiente:

En primer lugar, se observa que tanto el contenedor no optimizado como el contenedor optimizado logran un alto nivel de disponibilidad, con valores del 99,48% y 99,39% respectivamente. Esto indica que ambos contenedores son capaces de responder a las solicitudes de manera eficiente y mantener un nivel de servicio adecuado.

En cuanto al tiempo transcurrido durante las pruebas, se puede apreciar que el contenedor optimizado muestra un mejor rendimiento, con un tiempo transcurrido de 43,10 segundos frente a los 50,32 segundos del contenedor no optimizado. Esta diferencia indica que el contenedor optimizado es capaz de procesar las solicitudes de manera más rápida y eficiente.

En cuanto al tiempo de respuesta, se observa que el contenedor optimizado presenta un tiempo de respuesta promedio de 0,52 segundos, mientras que el contenedor no optimizado tiene un tiempo de respuesta promedio de 0,62 segundos. Esto sugiere que el contenedor optimizado es capaz de atender las solicitudes con mayor agilidad y eficiencia.

Además, se destaca que el contenedor optimizado muestra un mayor rendimiento en términos de tasa de transacciones, con un valor de 397,51 transacciones por segundo, en comparación con las 345,1 transacciones por segundo del contenedor no optimizado. Esto indica que el contenedor optimizado es capaz de manejar un mayor volumen de transacciones en un período de tiempo determinado.

En cuanto al rendimiento en términos de throughput, se observa que el contenedor optimizado tiene un valor de 16,79 MB/s, mientras que el contenedor no optimizado tiene un valor de 14,58 MB/s. Esto indica que el contenedor optimizado es capaz de procesar y transferir datos de manera más eficiente.

Por último, en cuanto a la concurrencia, se puede apreciar que ambos contenedores son capaces de manejar un alto número de usuarios simultáneos. El contenedor no optimizado muestra una capacidad de concurrencia de 213,34 usuarios simultáneos, mientras que el contenedor optimizado tiene una capacidad de concurrencia de 207,84 usuarios simultáneos.

En resumen, los resultados de las pruebas de estrés indican que el contenedor optimizado presenta un mejor rendimiento en términos de tiempo transcurrido, tiempo de respuesta, tasa de transacciones, throughput y capacidad de concurrencia.

Tabla 25. Comparación de datos recolectados por Stacer

Monitoreo del sistema con Stacer mientras se ejecuta siege		
	Contenedor no optimizado	**Contenedor optimizado**
CPU	100%	100%
Memoria	3,9 GiB	3,1 GiB

Autor: Anthony López

Al analizar los datos presentados en la tabla 25, se puede deducir lo siguiente:

En primer lugar, se observa que tanto el contenedor no optimizado como el contenedor optimizado muestran una utilización del CPU del 100%. Esto indica que durante la

ejecución de las pruebas de carga con Siege, ambos contenedores están haciendo uso máximo del recurso del CPU.

En cuanto a la memoria, se puede apreciar que el contenedor no optimizado muestra un consumo de memoria de 3,9 GiB, mientras que el contenedor optimizado tiene un consumo de memoria de 3,1 GiB. Esto sugiere que el contenedor optimizado utiliza de manera más eficiente el recurso de memoria, logrando un menor consumo en comparación con el contenedor no optimizado.

En resumen, los resultados del monitoreo del sistema con Stacer durante la ejecución de Siege muestran que tanto el contenedor no optimizado como el contenedor optimizado experimentan un alto uso del CPU. Sin embargo, el contenedor optimizado muestra una mayor eficiencia en el consumo de memoria en comparación con el contenedor no optimizado.

Tabla 26. Comparación de datos recogidos por docker stats

Monitoreo del contenedor con docker stats mientras se ejecuta siege		
	Contenedor no optimizado	**Contenedor optimizado**
CPU%	606,41%	599,05%
Mem usage	1,081 GiB	0,80 GiB
Mem %	14,03%	10,40 %

Autor: Anthony López

Al analizar los datos presentados en la tabla 26, se puede deducir lo siguiente:

En primer lugar, se observa que tanto el contenedor no optimizado como el contenedor optimizado muestran un alto uso del CPU durante la ejecución de siege. El contenedor no optimizado alcanza un uso del CPU del 606,41%, mientras que el contenedor optimizado alcanza un uso del CPU del 599,05%. Estos valores indican que ambos contenedores están sometidos a una carga intensiva en términos de utilización del CPU.

En cuanto a la memoria, se puede apreciar que el contenedor no optimizado muestra un consumo de memoria de 1,081 GiB, mientras que el contenedor optimizado tiene un consumo de memoria de 0,80 GiB. Estos datos reflejan una diferencia en la eficiencia del consumo de memoria entre ambos contenedores, siendo el contenedor optimizado más eficiente en la utilización de este recurso.

El porcentaje de uso de memoria también muestra una diferencia notable entre los contenedores. El contenedor no optimizado utiliza un 14,03% de la memoria, mientras que el contenedor optimizado utiliza un 10,40% de la memoria. Esto indica que el contenedor optimizado logra una utilización más eficiente de la memoria en comparación con el contenedor no optimizado.

En resumen, los resultados del monitoreo del contenedor con docker stats durante la ejecución de siege muestran que tanto el contenedor no optimizado como el contenedor optimizado experimentan un alto uso del CPU. Sin embargo, el contenedor optimizado demuestra una mayor eficiencia en el consumo de memoria y en el porcentaje de uso de memoria en comparación con el contenedor no optimizado.

Tabla 27. Comparación de rendimiento de herramientas de desarrollo

Benchmarking de rendimiento de herramientas de desarrollo		
	Contenedor no optimizado	**Contenedor optimizado**
Tiempo	12,2 s	05,8 s

Autor: Anthony López

Al analizar los datos presentados en la tabla 27, se puede deducir lo siguiente:

Los resultados muestran una clara diferencia en el tiempo requerido por las herramientas de desarrollo en los contenedores no optimizado y optimizado. El contenedor no optimizado presenta un tiempo de ejecución de 12,2 segundos, mientras que el contenedor optimizado logra un tiempo de ejecución significativamente menor, de tan solo 5,8 segundos. Esta reducción en el tiempo de ejecución indica una mejora

notable en el rendimiento de las herramientas de desarrollo al aplicar estrategias de optimización.

La diferencia en los tiempos de ejecución demuestra la eficacia de las estrategias de optimización implementadas en el contenedor optimizado. Estas estrategias pueden incluir ajustes de configuración, eliminación de procesos innecesarios o mejora en la gestión de recursos. El resultado es una mayor eficiencia en el procesamiento y una reducción del tiempo requerido para realizar las tareas de desarrollo.

En resumen, los resultados del benchmarking de rendimiento de herramientas de desarrollo evidencian que el contenedor optimizado ofrece un rendimiento superior al contenedor no optimizado, logrando un tiempo de ejecución significativamente más rápido.

CAPÍTULO IV.- CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

- El análisis de la construcción de imágenes Docker revela que la comprensión de las capas, operaciones y configuración empleadas es esencial para identificar oportunidades de optimización. Esta visión profunda permitió minimizar el tamaño y la complejidad de las imágenes, resultando en contenedores más eficientes y ágiles.
- El benchmarking es una herramienta valiosa para evaluar el rendimiento de los contenedores generados a partir de microservicios. Esta técnica permite medir el desempeño y comparar diferentes configuraciones, lo que facilita la identificación de áreas de mejora y la selección de las estrategias más eficientes.
- La elaboración de estrategias de optimización en la construcción de imágenes Docker es un paso fundamental para desarrollar contenedores más eficientes. Esto implica adoptar mejores prácticas, como el uso de imágenes base oficiales, reducción de capas, configuración adecuada de recursos y eliminación de dependencias innecesarias.
- La optimización en el desarrollo con contenedores es un proceso continuo y dinámico que requiere estar al tanto de las últimas tendencias, herramientas y técnicas emergentes en el entorno de Docker. Mantenerse actualizado y adoptar las mejores prácticas vigentes es fundamental para lograr construir imágenes Docker eficientes y aprovechar al máximo los avances tecnológicos disponibles.

4.2. Recomendaciones

- Se recomienda utilizar imágenes base oficiales y reducir el número de capas al mínimo necesario. Esto permitirá mantener un entorno limpio y evitar la inclusión de componentes innecesarios, mejorando así el rendimiento y la eficiencia de los contenedores.

- Es fundamental implementar un proceso de benchmarking periódico para evaluar el rendimiento de los contenedores generados a partir de microservicios. Este enfoque permitirá identificar posibles cuellos de botella, áreas de mejora y oportunidades para optimizar el rendimiento y la escalabilidad de los contenedores, asegurando un desempeño óptimo en todo momento.

- Se recomienda adoptar prácticas recomendadas en la construcción de imágenes Docker, como el uso de imágenes base oficiales, minimización de capas, configuración adecuada de recursos y eliminación de dependencias innecesarias. Al seguir estas prácticas, se logrará la generación de contenedores más eficientes, reduciendo el tamaño y la complejidad de las imágenes y mejorando su rendimiento general.

BIBLIOGRAFÍA

[1] L. E. Santander Alcívar y G. M. Zambrano Parrales, «Análisis de la tecnología docker como contenedor de aplicaciones», bachelorThesis, Calceta: ESPAM MFL, 2022. Accedido: 7 de noviembre de 2022. [En línea]. Disponible en: http://repositorio.espam.edu.ec/handle/42000/1713

[2] L. E. G. López y C. A. G. Alarcón, «Extensión de la arquitectura Docker para el despliegue automático de contenedores», Ingeniare, n.o 29, Art. n.o 29, nov. 2020, doi: 10.18041/1909-2458/ingeniare.29.7432.

[3] L. C. Arboleda Bonilla, «Aplicación del sistema de contenedores Docker, como alternativa a sistemas de virtualización para mejorar el testeo de aplicaciones en un entorno de desarrollo C#», bachelorThesis, Universidad Técnica de Ambato. Facultad de Ingeniería en Sistemas, Electrónica e Industrial. Carrera de Tecnologías de la Información, 2022. Accedido: 15 de noviembre de 2022. [En línea]. Disponible en: https://repositorio.uta.edu.ec:8443/jspui/handle/123456789/36645

[4] Y. D. Vizcaino Quiroz, «Optimización de aplicaciones Java Enterprise monolíticas mediante el uso de contenedores Docker», bachelorThesis, 2021. Accedido: 24 de noviembre de 2022. [En línea]. Disponible en: http://repositorio.utn.edu.ec/handle/123456789/10981

[5] F. G. Montalvo Ochoa, «Análisis, diseño y desarrollo de un sistema para la gestión del seguimiento de pasantías y prácticas pre-profesionales a través de la implementación de microservicios en Docker para la Universidad Politécnica Salesiana», bachelorThesis, 2020. Accedido: 24 de noviembre de 2022. [En línea]. Disponible en: http://dspace.ups.edu.ec/handle/123456789/19483

[6] J. C. Antepara Reyes y L. N. Villamar Flores, «Análisis de vulnerabilidades y definición de contramedidas en la seguridad de aplicaciones, basadas en contenedores usando la herramienta docker», Thesis, Universidad de Guayaquil. Facultad de Ciencias Matemáticas y Físicas. Carrera de Ingeniera en Networking y Telecomunicaciones, 2020. Accedido: 24 de noviembre de 2022. [En línea]. Disponible en: http://repositorio.ug.edu.ec/handle/redug/48780

[7] J. L. Pacheco Laje, «Estudio comparativo entre una arquitectura con microservicios y contenedores dockers y una arquitectura tradicional (monolítica) con comprobación aplicativa», Thesis, Universidad de Guayaquil. Facultad de Ciencias Matemáticas y Físicas. Carrera de Ingeniería En Sistemas Computacionales, 2018. Accedido: 24 de noviembre de 2022. [En línea]. Disponible en: http://repositorio.ug.edu.ec/handle/redug/32755

[8] M. D. C. Gomez Fuentes, J. Cervantes Ojeda, y P. P. Gonzalez Perez, «Fundamentos de ingeniería de software», 2019, Accedido: 25 de abril de 2023. [En línea]. Disponible en: http://ilitia.cua.uam.mx:8080/jspui/handle/123456789/1000

[9] AdmItsqmet, «Que es desarrollo de software», ITSQMET, 21 de octubre de 2022. https://itsqmet.edu.ec/desarrollo-de-software/ (accedido 1 de diciembre de 2022).

[10] E. I. González Jaimes et al., «Estrategia didáctica de enseñanza y aprendizaje para programadores de software», RIDE. Revista Iberoamericana para la Investigación y el Desarrollo Educativo, vol. 9, n.o 17, pp. 688-712, dic. 2018, doi: 10.23913/ride.v9i17.402.

[11] L. J. Gonçales, K. Farias, y B. C. da Silva, «Measuring the cognitive load of software developers: An extended Systematic Mapping Study», Information and Software Technology, vol. 136, p. 106563, ago. 2021, doi: 10.1016/j.infsof.2021.106563.

[12] E. G. Maida y J. Pacienzia, «Metodologías de desarrollo de software», 2015, Accedido: 24 de noviembre de 2022. [En línea]. Disponible en: https://repositorio.uca.edu.ar/handle/123456789/522

[13] S. M. Velásquez, J. D. V. Montoya, M. E. G. Adasme, E. J. R. Zapata, A. A. Pino, y S. L. Marín, «Una revisión comparativa de la literatura acerca de metodologías tradicionales y modernas de desarrollo de software», Revista CINTEX, vol. 24, n.o 2, Art. n.o 2, dic. 2019, doi: 10.33131/24222208.334.

[14] J. McAffer, «Getting Started With Open Source Governance», Computer, vol. 52, n.o 10, pp. 92-96, oct. 2019, doi: 10.1109/MC.2019.2929568.

[15] J. L. Briceño Mariño y C. Barrera Williams, «Propuesta de despliegue continuo y virtualización basada en contenedores para ambientes de pruebas de software», dic. 2020, Accedido: 25 de abril de 2023. [En línea]. Disponible en: http://repository.udistrital.edu.co/handle/11349/28005

[16] C. B. Pareja Valerio y L. J. Burgos Robles, «La arquitectura de software basada en microservicios: Una revisión sistemática de la literatura», Universidad Peruana Unión, dic. 2019, Accedido: 25 de enero de 2023. [En línea]. Disponible en: https://repositorio.upeu.edu.pe/handle/20.500.12840/2524

[17] I. Salinas Lizoain, «Optimización y bastionado de imágenes de contenedores Docker», 2020, Accedido: 25 de enero de 2023. [En línea]. Disponible en: https://academica-e.unavarra.es/xmlui/handle/2454/37616

[18] R. Salvatierra Rodriguez, «CONTENEDORES LINUX Y SU RAPIDA IMPLEMENTACION EN APLICACIONES WEB», Thesis, 2019. Accedido: 25 de enero de 2023. [En línea]. Disponible en: http://ddigital.umss.edu.bo:8080/jspui/handle/123456789/14446

[19] B. Piedade, J. P. Dias, y F. F. Correia, «An empirical study on visual programming docker compose configurations», en Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, en MODELS '20. New York, NY, USA: Association for Computing Machinery, oct. 2020, pp. 1-10. doi: 10.1145/3417990.3420194.

[20] B. S. Kim, S. H. Lee, Y. R. Lee, Y. H. Park, y J. Jeong, «Design and Implementation of Cloud Docker Application Architecture Based on Machine Learning in Container Management for Smart Manufacturing», Applied Sciences, vol. 12, n.o 13, Art. n.o 13, ene. 2022, doi: 10.3390/app12136737.