UNIVERSIDAD TÉCNICA DEL NORTE

Facultad de Ingeniería en Ciencias Aplicadas

Carrera de Software

Estudio comparativo del rendimiento de base de datos PostgreSQL en ambientes locales y en contenedores Docker para fomentar el levantamiento de una arquitectura DevOps.

Trabajo de grado previo a la obtención del título de Ingeniero de Software presentado ante la ilustre Universidad Técnica del Norte.

Autor:

Cinthya Graciela Arias Lema

Director:

PhD. José Antonio Quiña Mera MSc.

Ibarra – Ecuador

2024

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1004565402		
APELLIDOS Y NOMBRES:	ARIAS LEMA CINTHYA GRACIELA		
DIRECCIÓN:	PEGUCHE - OTAVALO		
EMAIL:	cgariasl@utn.edu.ec gracielalema98@gmail.com		
TELÉFONO FIJO:	S/N	**TELÉFONO MÓVIL:**	0959848609

DATOS DE LA OBRA	
TÍTULO:	ESTUDIO COMPARATIVO DEL RENDIMIENTO DE BASE DE DATOS POSTGRESQL EN AMBIENTES LOCALES Y EN CONTENEDORES DOCKER PARA FOMENTAR EL LEVANTAMIENTO DE UNA ARQUITECTURA DEVOPS.
AUTOR(ES):	CINTHYA GRACIELA ARIAS LEMA
FECHA:	11/03/2024
PROGRAMA:	PREGRADO
TÍTULO POR EL QUE OPTA:	INGENIERA DE SOFTWARE
DIRECTOR:	PhD. ANTONIO QUIÑA MSC
ASESOR 1:	ING. MAURICIO REA MSC
ASESOR 2:	ING. MARCO PUSDÁ MSC.

2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los once días del mes de marzo de 2024

EL AUTOR:

ESTUDIANTE

Cinthya Graciela Arias Lema

C.I 1004565402

CERTIFICACIÓN DIRECTOR

Ibarra, 11 de marzo del 2024

CERTIFICACIÓN DIRECTOR DEL TRABAJO DE TITULACIÓN

Por medio del presente yo PhD. Antonio Quiña Mera MSc, certifico que la Srta. CINTHYA GRACIELA ARIAS LEMA portador de la cédula de ciudadanía número 100456540-2, ha trabajado en el desarrollo del proyecto de grado **"**Estudio comparativo del rendimiento de base de datos postgreSQL en ambientes locales y en contenedores docker para fomentar el levantamiento de una arquitectura devOps.**",** previo a la obtención del Título de Ingeniero en Software realizado con interés profesional y responsabilidad que certifico con honor de verdad.

Es todo en cuanto puedo certificar a la verdad

Atentamente

1002322384
JOSE
ANTONIO
QUIÑA MERA

Firmado digitalmente por 1002322384 JOSE ANTONIO QUIÑA MERA
Fecha: 2024.03.11 15:11:58 -05'00'

_____

PhD. Antonio Quiña Mera MSc.

DIRECTOR DE TRABAJO DE GRADO

DEDICATORIA

A mis padres José Rafael Arias Lema y María Marina Lema Oyagata, quienes han sido mi mayor fuente de amor, apoyo y sabiduría a lo largo de mi vida. Su constante aliento y sacrificio han sido la luz que ha guiado mi camino en este viaje académico. Cada logro alcanzado en este trabajo de grado es un reflejo de su inquebrantable dedicación y amor.

A mis hermanos, cuyo afecto, compañía y aliento han sido pilares fundamentales en mi vida. A través de cada desafío y éxito, han estado a mi lado, brindándome su apoyo incondicional y motivándome a seguir adelante.

A todos aquellos que han sido parte de mi viaje, ya sea brindando su amistad, ofreciendo su consejo, o compartiendo su amor y aliento a lo largo de los años, contribuyendo de diversas maneras a mi crecimiento personal y académico. Su presencia ha sido fundamental en este logro.

AGRADECIMIENTO

Agradezco a Dios por ser mi guía y fortaleza en este camino académico, por concederme la sabiduría y perseverancia para alcanzar esta meta, a mis padres, María Marina Lema Oyagata y José Rafael Arias Lema, por su amor y apoyo incondicionales, y a mis queridos hermanos, cuyo aliento y compañía fueron esenciales en momentos difíciles.

A mi tutor de tesis, PhD Antonio Quiña Mera MSc, por su orientación experta, su paciencia infinita y su invaluable mentoría a lo largo de este proceso. Agradezco también a mis opositores Ing. Mauricio Rea MSc y Ing. Marco Pusdá MSc por sus valiosas contribuciones y sus comentarios constructivos, que enriquecieron significativamente este trabajo.

Agradezco a mis docentes por compartir su conocimiento y su experiencia, por inspirarme con su pasión por la enseñanza y por guiarme en mi crecimiento académico y profesional.

A mis compañeros de clase y amigos por su constante apoyo, sus palabras de ánimo. Sus risas, su compañía y su colaboración hicieron de este viaje una experiencia inolvidable. A todos ustedes, gracias por estar siempre a mi lado.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLA

RESUMEN

El presente trabajo de grado aborda el análisis comparativo del rendimiento entre la virtualización de contenedores en Docker y el entorno local, específicamente enfocado en el despliegue de base de datos PostgreSQL para impulsar el desarrollo de una arquitectura DevOps.

Se presenta el marco teórico que sustenta el estudio, destacando las herramientas tecnológicas clave, como la virtualización de contenedores en Docker, el ambiente local y una comparativa entre ambos. Además, se examina aspectos metodológicos relevantes, como la experimentación de la Ingeniería de Software y los estándares de calidad.

Se describe el entorno experimental, incluyendo los objetivos, factores, tratamientos, variables, hipótesis y diseño del experimento. Se proporciona una visión general de los casos de uso, las tareas experimentales y la instrumentación utilizada para la recolección de datos.

Se lleva a cabo un análisis estadístico de la eficiencia, evaluando la comparativa entre los diferentes entornos en términos de rendimiento. Además, se interpreta la información recopilada y se extraen conclusiones significativas que orientan la toma de decisiones en cuanto a la implementación de bases de datos PostgreSQL en entornos de desarrollo. Se destaca que el mejor ambiente para ejecutar los casos de uso fue Docker, ya que demostró resultados superiores en comparación con el entorno local en todos los casos analizados.

Palabras claves: contenedores, docker, experimentación computacional, guía Wholin, eficiencia de rendimiento, ambiente Localhost.

ABSTRACT

This degree work addresses the comparative analysis of performance between container virtualization in Docker and the local environment, specifically focused on the deployment of the PostgreSQL database to promote the development of a DevOps architecture.

The theoretical framework that supports the study is presented, highlighting the key technological tools, such as container virtualization in Docker, the local environment and a comparison between the two. In addition, relevant methodological aspects are examined, such as Software Engineering experimentation and quality standards.

The experimental setting is described, including the objectives, factors, treatments, variables, hypotheses, and design of the experiment. An overview of the use cases, experimental tasks, and instrumentation used for data collection is provided.

A statistical analysis of efficiency is carried out, evaluating the comparison between the different environments in terms of performance. In addition, the information collected is interpreted and significant conclusions are drawn that guide decision-making regarding the implementation of PostgreSQL databases in development and deployment environments. It is highlighted that the best environment to execute the use cases was Docker, since it demonstrated superior results compared to the local environment in all the cases analyzed.

Keywords: containers, docker, computational experimentation, Wholin guide, performance efficiency, Localhost environment.

INTRODUCCIÓN

Tema

Estudio comparativo del rendimiento de base de datos PostgreSQL en ambientes locales y en contenedores Docker para fomentar el levantamiento de una arquitectura DevOps.

Problema

Antecedentes

Actualmente se observa que a nivel empresarial tanto en sectores privados como públicos, incluso en instituciones educativas es escasa la actualización continua en el proceso de desarrollo de software, Según (Zeron, 2019) la tecnología informática cambia rápidamente y el no realizar actualizaciones regularmente puede provocar problemas para la empresa, generando pérdidas económicas. Según el equipo de Datadog, donde se evalúa la adopción de Docker “Casi una cuarta parte de las empresas han adoptado Docker” a principios de abril de 2018, el 23,4% de los clientes de Datadog habían adoptado Docker, desde 2015, la proporción de clientes que ejecutan Docker han crecido a un ritmo de entre 3 y 5 puntos por año (Luz, 2021).

Según (Cuesta & González, 2016) Define a Docker como un proyecto open source para empaquetar, transportar y ejecutar cualquier aplicación como contenedor ligero. Facilitando el lanzamiento de calidad superior, mejor escalabilidad de aplicaciones y una mayor aislamiento de aplicaciones.

En la carrera de Software (CSOFT), los estudiantes muestran carencias en el manejo y actualización de las nuevas herramientas, procesos, metodologías y lenguajes utilizados en el desarrollo de software. Además, se observa una falta de actualización en las mallas curriculares y los sílabos de los cursos, lo que dificulta la adaptación a las nuevas tecnologías que surgen con frecuencia.

Por otra parte, es importante conocer la problemática que genera el uso de base de datos como los colapsos en peticiones de información de alta robustez, produciendo caídas, estas caídas muchas veces causan impactos importantes, ya que puede paralizar cualquier actividad. Además la falta y el consumo exesivo de recursos que se puede presentar, en ciertos casos al trabajar en ambientes locales se requiere de requisitos de la infraestructura para el entorno y su configuración puede ser mucho más extensa (Funes, 2018).

Situación Actual

Actualmente se conoce que la gran mayoría de entidades hacen uso de almacenamiento de información de gran cantidad, que requiere la seguridad y el manejo rápido de las mismas. La tecnología evoluciona constantemente y con ella las mejoras en el desarrollo de bases de datos haciendo que las empresas y departamentos de TI (Tecnología de la información) adopten nuevas tendencias que faciliten el trabajo (Simbaña Alarcón, 2016).

Dentro de la carrera de Software de la UTN se desconoce sobre las nuevas herramientas tecnológicas por lo que la gran parte de los estudiantes siguen el mismo enfoque de semestres anteriores ya que existe una deficiencia en la actualización de los contenidos del syllabus que se imparten dentro de la carrera, por esta razón se hace este proyecto que da a conocer sobre el estudio de rendimiento entre ambos ambientes, ayudando en la selección de herramientas, dando nuevas alternativa de integración de base de datos para fomentar una arquitectura DevOps (Unión de procesos y tecnología para ofrecer valor a los clientes de forma constante) mejorando el rendimiento y creando proyectos de más calidad en menor tiempo (Azure, 2019).

Prospectiva

Para la solución de aquellos inconvenientes, la presente investigación pretende realizar la comparativa de instancias de base de datos PostgreSQL en ambientes locales y en contenedores Docker, dando alternativa de trabajo para la mejor selección de estos dos ambientes, con el fin de no tener problemas de disponibilidad de recursos que se presenten en los estudiantes de la materia de Fábrica de Software.

Planteamiento del problema

El alto grado de desconocimiento de las nuevas tendencias tecnológicas como los contenedores Docker influye en la formación profesional del estudiante poniendo en riesgo la desactualización al integrarse en diferentes empresas.

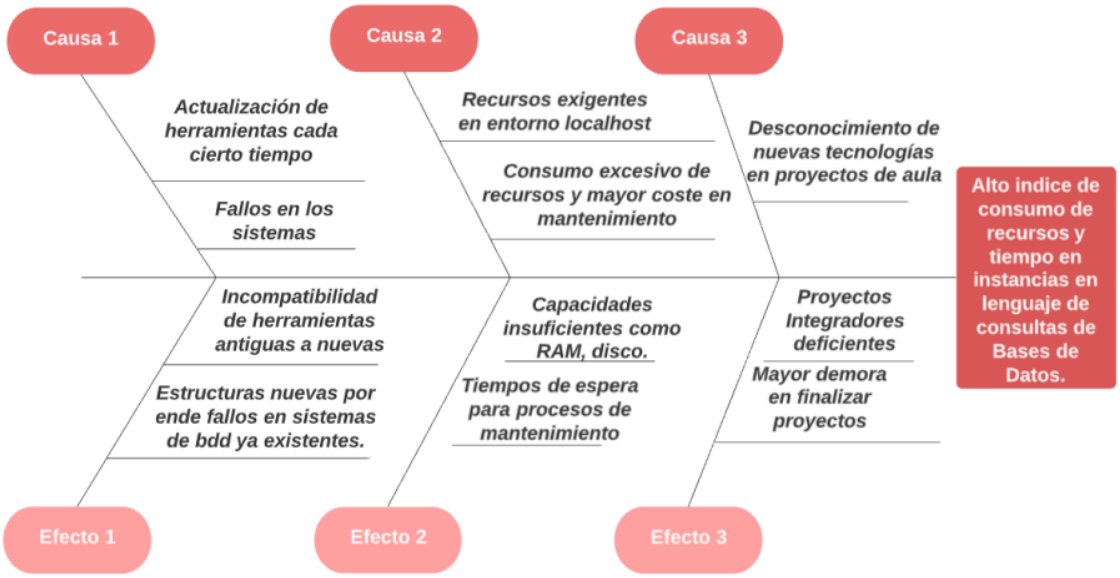


Figura 1: Árbol de problema

Objetivos

Objetivo General.

Realizar un estudio comparativo del rendimiento de bases de datos PostgreSQL en ambientes locales y en contenedores Docker basados en métricas de la norma ISO/IEC 25023 para fomentar el levantamiento de una arquitectura DevOps.

Objetivos Específicos

- Desarrollar un marco teórico de instancias en contenedores Docker y en entorno local mediante la instancia de base de datos en despliegues continuos.
- Realizar un estudio comparativo de la implementación y configuración de contenedores Docker de base de datos y en entorno localhost.
- Analizar los resultados obtenidos utilizando estadística descriptiva con la curva de distribución normal.

Alcance

La comparativa de este proyecto está enfocado a los estudiantes de la carrera de Software de la UTN, tanto docentes como estudiantes son los encargados de nutrir las nuevas alternativas en el proceso de desarrollo de software para fomentar la creación de arquitecturas DevOps.

En este proyecto se realizará el estudio comparativo de instancias de bases de datos PostgreSQL en contenedores Docker como en ambiente local (localhost) para innovar y

mejorar el proceso de desarrollo de software en proyectos de aula, en ambos ambientes la implementación será realizada en un equipo personal del estudiante.

PostgreSQL está dentro de la lista de tecnologías más comunes que se ejecutan en Docker, llevando el tercer puesto siendo la base de datos relacional de código abierto que ha ido subiendo de rango cada año (Rodríguez, 2019).

Para ambos casos se realizará una aplicación Backend que simulará el módulo de transferencia y consulta de cuentas de un sistema bancario, en donde se implementará 5 casos de uso en cada uno de los ambientes. Para verificar su optimización y comportamiento en ambos ambientes se evaluará el rendimiento de datos y el lenguaje de consultas para bases de datos relacionales (create, insert, select, detele), mediante métricas comparativas expuestas en la ISO/IEC 25023 con la característica "*medidas de eficiencia del desempeño*". Los resultados obtenidos se analizarán mediante estadística descriptiva, de esta manera poder utilizar y seleccionar uno de los dos ambientes estudiadas.

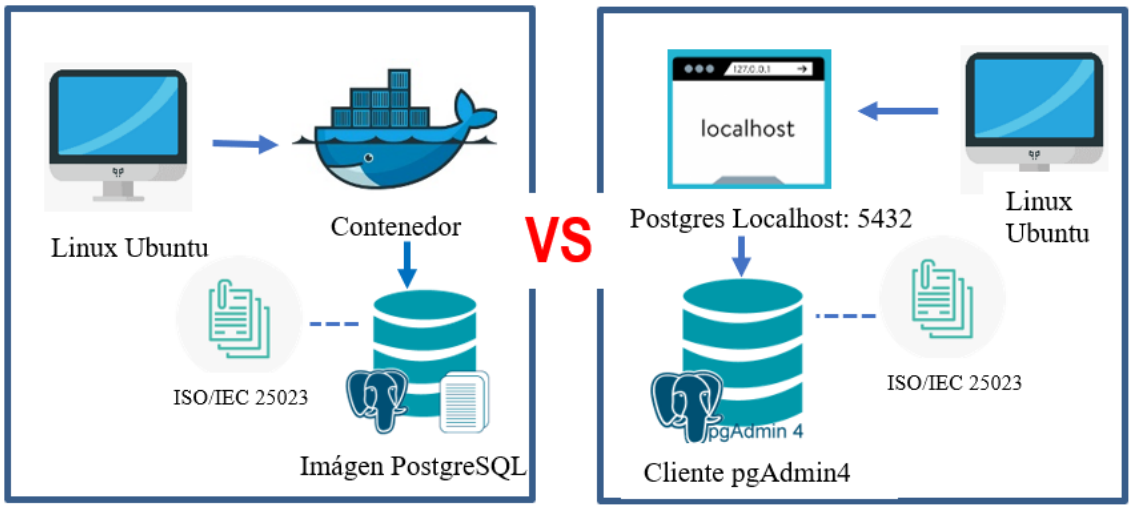


Figura 2: Estructura del Proyecto

Metodología

Para cumplir con el primer objetivo se aplicará una metodología de Revisión sistemática de literatura (SLR) que recopila y analiza críticamente múltiples estudios o trabajos de investigación, basado en criterios y filtros de base de datos científicas que permitirá desarrollar la base teórica (García, 2021).

Para el cumplimiento del objetivo dos se definirá una serie de métricas que estarán orientadas al desempeño y rendimiento dentro de la instancia de lenguaje de consultas de base de datos en contenedores Docker y un ambiente local, que tendrán como base en el

análisis de las métricas de la ISO/IEC 25023 con la característica "*medidas de eficiencia del desempeño*".

Finalmente, para el cumplimiento del objetivo tres se analizará el comportamiento obtenido de las métricas de evaluación como son: medidas de comportamiento temporal, medidas de utilización de recursos y medidas de capacidad, en ambos ambientes. Mediante la teoría de distribución normal se realizará la interpretación de los datos obtenidos.

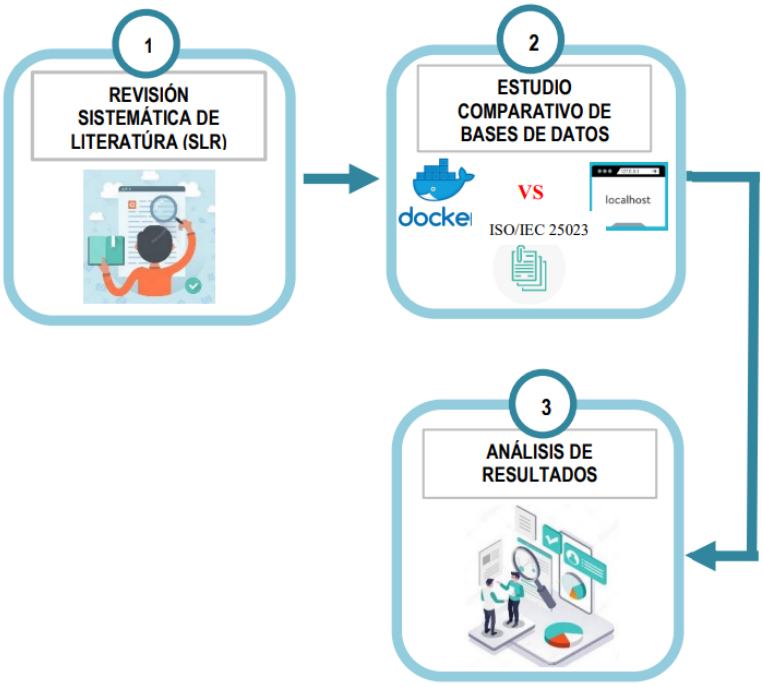


Figura 3: Proceso metodológico

Justificación

Justificación metodológica.

A medida que la tecnología va en aumento y los avances tecnológicos cada vez son más grandes, se ha implementado una serie de herramientas, servicios y nuevos softwares de TI y entre ellos están los contenedores Docker, tecnología que permite la creación y el uso de contenedores de Linux. Además, que ha sido adoptado por muchas empresas para su mejor rendición, permitiendo su ejecución en cualquier maquina con Docker instalado independientemente del Sistema Operativo o recursos que sean distintos entre ellos.

Algunos desarrolladores tienen conocimiento de los contenedores Docker y la facilidad que brinda la creación de instancias de bases de datos. La mayoría de las empresas, instituciones u organizaciones trabajan con sistemas de bases comunes como son en entorno Localhost, llevando un sin número de procesos para que dicha Base de Datos inicie su función

de almacenamiento de información, empezando desde instalaciones complejas y configuraciones distintas por cada tipo de máquina o SO, muchas de estas empresas desconocen las nuevas tecnologías que agilita el uso de Base de Datos y su función principal. Este estudio comparativo permitirá a los desarrolladores a optar por el uso de nuevas tecnologías.

El presente proyecto se enfoca en el objetivo de Desarrollo Sostenible planteados por la ONU y UNESCO: Objetivo 9. “Construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación” (*Objetivos de Desarrollo Sostenible | Programa De Las Naciones Unidas Para El Desarrollo*, n.d.)

Justificación Tecnológica.

Este proyecto facilitara a los desarrolladores o personal tecnológico a la elección de herramientas y entornos para el trabajo e implementación de bases de datos, facilitando la manipulación de datos incluso desde su instalación de herramientas y su facilidad en integrar distintas bases de datos en computadoras con diferentes características sin ningún problema, esto especialmente para empresas que manejen gran cantidad de información ayudando a tener un sistema organizado que optimice tiempo y recursos.

Justificación Educativa.

Es importante utilizar nuevas técnicas, métodos, herramientas que se presentan en la actualidad, para ello dentro de las instituciones educativas se debe fomentar el estudio y actualización continuo tanto por el docente como en un estudio autónomo por parte de los estudiantes, para mejorar su producto de software iniciando con un correcto proceso en el diseño de software, mejorando las competencias de estudio con las nuevas tendencias tecnológicas para así fortalecer los resultados de aprendizaje de la materia de Fábrica de software.

CAPÍTULO 1

1. Marco Teórico

En este capítulo, se establece el marco teórico que proporciona el fundamento para la investigación en cuestión. Se abordan conceptos fundamentales, teorías relevantes y herramientas tecnológicas clave que servirán como base para la exploración y análisis del presente trabajo. En primer lugar, se examinan las herramientas tecnológicas modernas utilizadas destacando su importancia y aplicación en el contexto actual. Posteriormente, se profundiza en aspectos específicos como la virtualización de contenedores en Docker,

analizando su funcionalidad, ventajas y contribuciones al desarrollo y despliegue de aplicaciones en entornos informáticos.

1.1. Herramientas tecnológicas.

1.1.1. Virtualización de contenedores en Docker

La virtualización de contenedores en Docker representa una innovación significativa en el ámbito del desarrollo de software. Docker, como proyecto de código abierto, revoluciona el despliegue y gestión de aplicaciones al automatizar este proceso en contenedores de software. Estos contenedores ofrecen una capa adicional de abstracción, permitiendo la ejecución de aplicaciones de forma independiente en distintos entornos sin preocuparse por las diferencias entre sistemas operativos subyacentes. Esta tecnología, descrita por (Arjona Quijano, 2019), simplifica el desarrollo y el despliegue de aplicaciones al proporcionar un entorno consistente y portátil, lo que facilita la implementación ágil y eficiente de sistemas distribuidos.

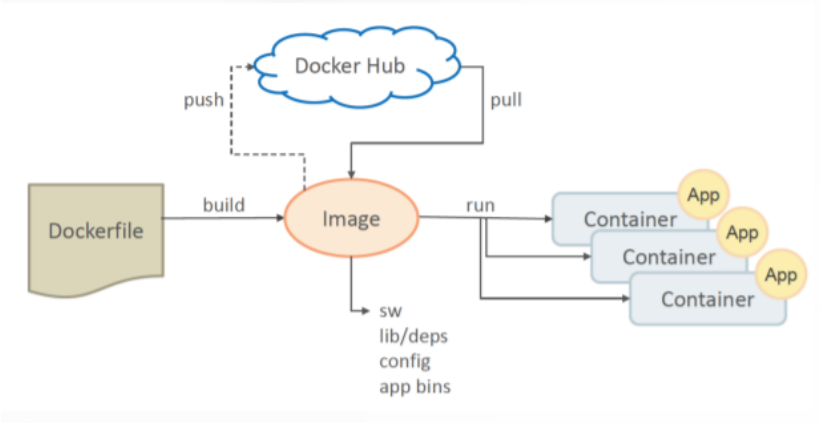


Figura 4: Cómo funciona Docker

Fuente: (Arjona Quijano, 2019)

Docker, como se ha discutido anteriormente, ofrece una eficiente solución de virtualización a través de su característico entorno encapsulado conocido como contenedor. Estos contenedores, por diseño, funcionan de manera aislada, proporcionando un nivel de seguridad y portabilidad esencial al encapsular todos los elementos necesarios para su operación, lo que reduce la dependencia del entorno subyacente. Internamente, Docker opera bajo una arquitectura de tipo 'cliente-servidor', donde el cliente Docker facilita la comunicación con el Daemon, encargado de realizar tareas cruciales como la creación, ejecución y distribución de los contenedores Docker. Este enfoque brinda una plataforma robusta y flexible para el desarrollo y despliegue de aplicaciones en entornos diversos y dinámicos. En palabras de (Arjona Quijano, 2019), Docker representa una herramienta fundamental que automatiza

el despliegue de aplicaciones dentro de contenedores de software, simplificando así el proceso de desarrollo y distribución de software en entornos heterogéneos.

El panorama arquitectónico descrito por (Silva, 2021) se presenta de manera concisa y comprensible a través de la siguiente figura, que ilustra la arquitectura cliente-servidor de Docker. Esta representación gráfica encapsula los componentes esenciales y las interacciones clave dentro del sistema, ofreciendo una visión general de la estructura subyacente. La figura sirve como una herramienta visual poderosa para comprender la complejidad y las relaciones entre los diversos elementos arquitectónicos de Docker. Con esta representación visual, se aprecia la disposición de los componentes, lo que facilita la comprensión de la organización y el funcionamiento del sistema en su totalidad.

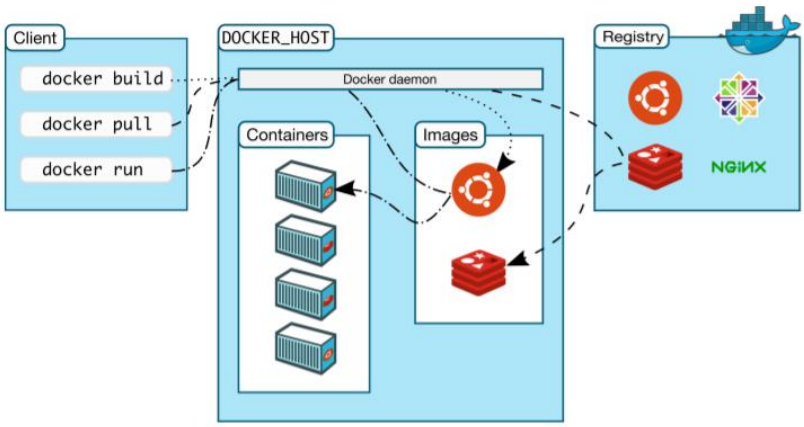


Figura 5: Arquitectura cliente – servidor de Docker

Fuente: (Silva, 2021)

De la imagen anterior destacan tres bloques principales: el cliente, el servidor (Docker host) y el registro (Registry). Por una parte, el cliente es la principal forma de comunicarse con el servicio de Docker. Ejecutando comandos como Docker run se envían al servicio peticiones mediante la API REST interna que gestionan los contenedores (Silva, 2021).

Conceptos básicos de Docker

- ### Imágenes Docker

En el universo de Docker, las imágenes representan mucho más que simples archivos; son plantillas sólidas y versátiles que contienen todas las instrucciones necesarias para crear un entorno de ejecución completamente funcional. Estas imágenes, por lo general, se construyen sobre la base de otras ya existentes, a las que se les añaden configuraciones y personalizaciones adicionales. Por ejemplo, podríamos partir de una imagen base de Ubuntu y, mediante un proceso de construcción, instalar un servidor Apache y nuestra aplicación

NodeJS, resultando en una imagen completamente nueva y adaptada a nuestras necesidades (Silva, 2021).

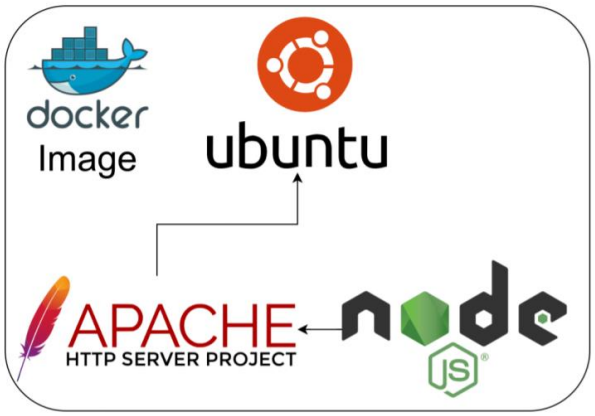


Figura 6: Ejemplo de imagen

Fuente: (Silva, 2021)

Para la creación de nuevas imágenes, Docker utiliza un archivo llamado Dockerfile, el cual contiene todas las directivas necesarias para la construcción de la imagen desde cero o basada en una existente. Este enfoque, similar al de los Makefiles, asegura una construcción eficiente y optimizada, donde solo se reconstruyen las partes que han sido modificadas, lo que resulta en imágenes más ligeras y ágiles en comparación con las máquinas virtuales.

- **Contenedores Docker**

Los contenedores en Docker son la verdadera esencia de la plataforma: representan instancias en tiempo de ejecución de una imagen específica. Cada contenedor es capaz de ejecutar aplicaciones de forma aislada y posee su propio sistema de archivos, espacio de trabajo y configuraciones de red. Estos contenedores pueden crearse, iniciarse, detenerse, moverse o eliminarse con facilidad, proporcionando así un entorno de ejecución ágil y flexible para nuestras aplicaciones (Pacheco Laje, 2018).

Aunque por defecto los contenedores se ejecutan de forma aislada en la máquina anfitriona, es posible configurar y gestionar el nivel de aislamiento para redes, almacenamiento y otros subsistemas subyacentes. Es importante tener en cuenta que un contenedor está directamente vinculado a la imagen que lo originó y a las configuraciones establecidas durante su creación. Cualquier modificación realizada durante su ciclo de vida se perderá al detener y eliminar el contenedor, incluyendo cualquier cambio en el sistema de archivos subyacente.

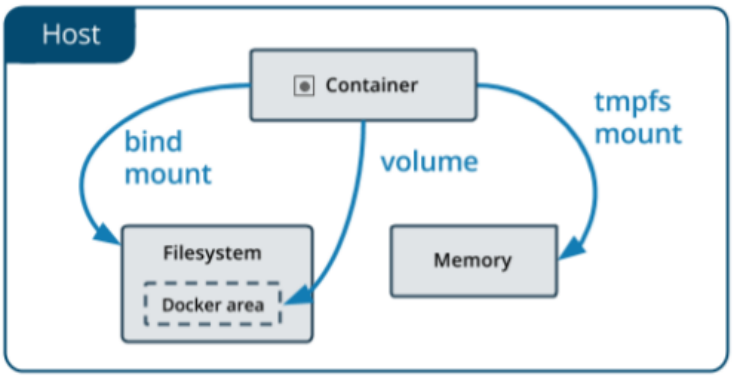


Figura 7: Contenedor – almacenamiento de información

Fuente: (Silva, 2021)

- **Volúmenes Docker**

Los volúmenes en Docker proporcionan una forma eficaz y segura de gestionar el almacenamiento persistente en la plataforma. Estos volúmenes, que se crean explícitamente mediante el comando "docker volume create", ofrecen un espacio de almacenamiento independiente de los contenedores y gestionado directamente por Docker. Al asociar un volumen a un contenedor, éste se monta como un directorio interno, lo que permite compartir datos entre contenedores y garantizar la persistencia de la información incluso si ningún contenedor lo está utilizando en ese momento.

Los volúmenes Docker se almacenan en un área específica dentro de la plataforma, aislados del sistema operativo subyacente y garantizando así su portabilidad y fiabilidad en cualquier entorno Según (Silva, 2021).

Según (Silva, 2021) Los volúmenes se almacenan en el área de Docker, aislados del sistema dentro del sistema.

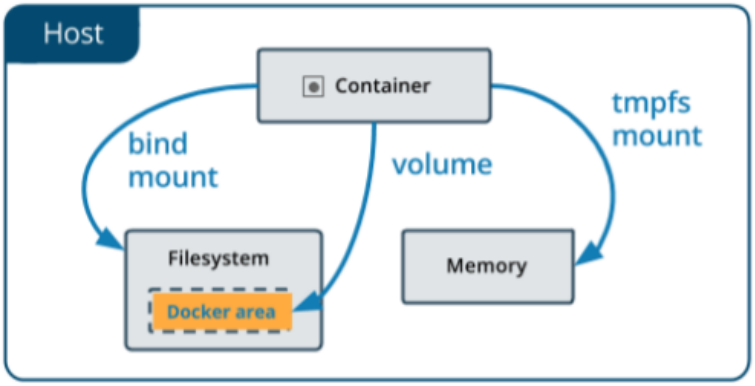


Figura 8: Ubicación del almacenamiento de volúmenes de Docker

Fuente: (Silva, 2021)

Ventajas y Desventajas

Al recurrir a contenedores Docker, los administradores y desarrolladores de software se benefician de la capacidad para ejecutar aplicaciones en un entorno altamente seguro. Este enfoque ofrece una mayor coherencia entre los entornos de prueba y producción, lo que conduce a una optimización eficiente de los recursos disponibles (Guijarro et al., 2019). Pero, pese a sus ventajas, también se presentan desafíos y limitaciones a considerar. En la Tabla 1, se esquematizan tanto las ventajas como las desventajas principales asociadas al uso de contenedores Docker.

Tabla 1: Ventajas y desventajas de Docker

Ventajas	Desventajas
Los contenedores se pueden duplicar fácilmente y comenzar en cuestión de segundos, lo que facilita la escalabilidad y la rápida implementación de aplicaciones.	Docker requiere un Kernel mínimo de versión 3.8 para su ejecución, lo que puede limitar su compatibilidad con sistemas más antiguos.
Docker consume recursos de hardware de manera eficiente, dirigiendo los recursos directamente a la aplicación en ejecución, lo que resulta en un mejor rendimiento del sistema.	Algunas versiones de Docker pueden presentar errores debido al constante desarrollo y actualización de la plataforma.
Es sencillo automatizar e integrar Docker en los procesos de desarrollo y despliegue de software, lo que aumenta la eficiencia y la velocidad de desarrollo.	En sistemas operativos específicos como Linux, Docker solo es compatible con arquitecturas de 64 bits, restringiendo su uso en sistemas más antiguos.
Las imágenes e instancias de Docker suelen ocupar menos espacio en comparación con las máquinas virtuales tradicionales, lo que permite un uso más eficiente del almacenamiento.	En Windows, Docker solo es compatible con sistemas operativos de 64 bits a partir de Windows 8, lo que limita su disponibilidad en versiones anteriores del sistema.
Existen numerosas imágenes disponibles que pueden ser	Para descargar e implementar imágenes Docker, se requiere acceso a Internet, lo que

descargadas y personalizadas según las necesidades específicas del proyecto, lo que facilita la creación y configuración de entornos de desarrollo.	puede ser una limitación en entornos sin conectividad constante.
Los contenedores Docker están aislados de la máquina física, lo que proporciona un nivel adicional de seguridad y facilita la administración de los contenedores.	Aunque Docker ofrece un control de versiones similar a Git para administrar imágenes y contenedores, el constante desarrollo de la plataforma puede generar inconsistencias entre versiones.

Fuente: (Pacheco, 2018)

1.1.2. Ambiente local

En el desarrollo de software, la configuración del entorno de trabajo desempeña un papel esencial en la eficiencia y efectividad de los procesos de creación y prueba de aplicaciones. El término "ambiente local" se refiere al entorno de desarrollo configurado directamente en la máquina del usuario, proporcionando un espacio controlado donde los programadores pueden diseñar, implementar y depurar sus soluciones.

Este entorno local, que ha sido la piedra angular del desarrollo de software durante décadas, se enfrenta hoy a la creciente influencia de tecnologías emergentes, como la virtualización y los contenedores. En esta sección, exploraremos en detalle el concepto de ambiente local, sus ventajas tradicionales y las complejidades que surgen al comparar su rendimiento con soluciones más modernas, como la virtualización en Docker. Este análisis crítico permitirá comprender mejor las dinámicas entre la eficacia probada del ambiente local y las innovaciones que buscan transformar la manera en que concebimos y ejecutamos aplicaciones en el panorama actual del desarrollo de software.

Ventajas y Desventajas

- **Ventajas**

Según (CUSTOM, 2019) , la virtualización ha traído consigo beneficios significativos, como la reducción de costos gracias a la optimización del espacio, la rápida asignación de recursos de servidor y una mejor capacidad de recuperación en casos de desastre, ya que los recursos de hardware ya no están ligados al centro de datos principal. Además, se destaca que la consolidación física ha permitido una mayor utilización de servidores, liberando recursos que pueden ser reasignados para otras tareas como el control de calidad o el desarrollo. Sin embargo, se señalan ciertos inconvenientes de este enfoque, como la

sobrecarga de recursos debido a la independencia de cada máquina virtual en cuanto a su sistema operativo, lo que puede afectar la memoria y el almacenamiento. Además, se destaca que la portabilidad de las aplicaciones se ve limitada entre diferentes entornos, como nubes públicas, nubes privadas y centros de datos tradicionales.

- **Desventajas**

Las máquinas virtuales, al igual que los contenedores Docker, pueden encapsular tanto hardware como software para ejecutarse en un entorno emulado. Esta combinación de elementos ofrece una representación completa y funcional de un sistema informático, proporcionando un entorno aislado y seguro para ejecutar aplicaciones y servicios. Sin embargo, a pesar de sus ventajas, las máquinas virtuales presentan desafíos significativos en términos de velocidad de iteración y coste de almacenamiento (Buchanan, 2019).

Velocidad de iteración.

En cuanto a la velocidad de iteración, el proceso de desarrollo y regeneración de máquinas virtuales puede resultar tedioso y consume mucho tiempo debido a la naturaleza compleja de la pila de software completa que abarcan. Cualquier modificación realizada en una máquina virtual requiere un tiempo considerable para regenerarla, así como para validar su comportamiento, lo que puede ralentizar el proceso de desarrollo y despliegue de aplicaciones.

Coste de tamaño de almacenamiento.

Por otro lado, el coste asociado al tamaño de almacenamiento de las máquinas virtuales también es un factor importante para considerar. Estas máquinas pueden ocupar mucho espacio de almacenamiento, a menudo alcanzando varios gigabytes en tamaño. Esta situación puede dar lugar a problemas de falta de espacio en el disco de la máquina anfitriona que aloja las máquinas virtuales, lo que puede afectar negativamente al rendimiento y la disponibilidad de los recursos de la infraestructura de TI (Buchanan, 2019).

1.1.3. Comparativa entre Docker y localhost

La comparación entre Docker y la ejecución en localhost revela diferencias significativas en términos de arquitectura y eficiencia. Mientras que en las máquinas virtuales tradicionales se establecen tres capas base en el servidor host, que incluyen la infraestructura, el sistema operativo host y un hipervisor, en Docker el servidor host se limita a la infraestructura y el sistema operativo. Además, Docker utiliza un motor de contenedor que

mantiene los contenedores aislados, pero les permite compartir los servicios del sistema operativo base único (De La Torre, 2022).

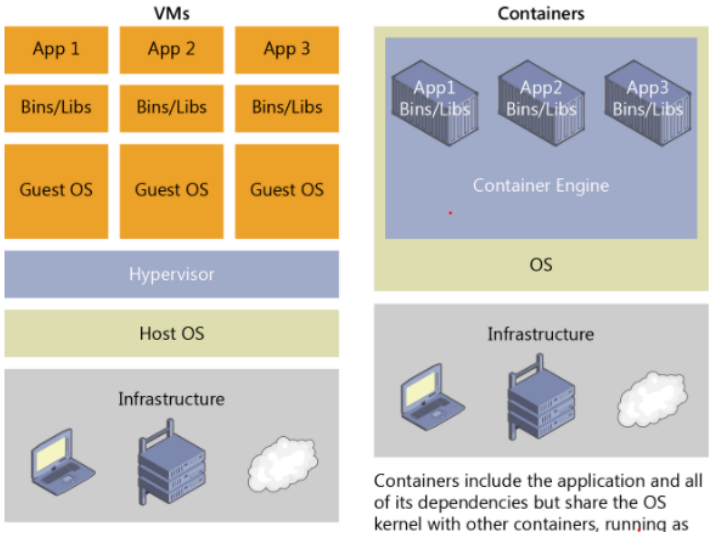


Figura 9: Arquitectura comparativa localhost vs Docker

Fuente: (De La Torre, 2022)

Esta diferencia en la estructura subyacente tiene importantes implicaciones en cuanto a la eficiencia y el rendimiento. Los contenedores Docker requieren menos recursos en comparación con las máquinas virtuales, ya que no necesitan un sistema operativo completo, lo que les permite iniciarse rápidamente y desplegarse con facilidad. Esta característica permite una mayor densidad de servicios en la misma unidad de hardware, lo que conduce a una reducción de los costos operativos.

Sin embargo, como consecuencia de compartir el mismo kernel del sistema operativo, los contenedores Docker ofrecen un menor nivel de aislamiento en comparación con las máquinas virtuales. Aunque esta falta de aislamiento puede ser un efecto secundario, es importante considerarla al diseñar y desplegar aplicaciones en entornos de contenedores.

1.1.4. Base de datos

El correcto manejo de las bases de datos es fundamental para las prácticas de programación contemporáneas. Una base de datos bien optimizada, respaldada por procesos automatizados de copia de seguridad, no solo agiliza el trabajo del desarrollador de aplicaciones, sino que también influye directamente en el rendimiento de las propias aplicaciones que dependen de dichas bases de datos (Benítez, 2015).

La gestión de la información constituye una disciplina esencial que implica la recepción, procesamiento y posterior entrega de datos transformados a un público específico,

respondiendo así a necesidades concretas. En este sentido, una base de datos actúa como una herramienta central, encargada de recopilar, organizar y relacionar datos de manera que permita una búsqueda y recuperación eficientes con el apoyo de sistemas informáticos. Hoy en día, estas bases de datos no solo facilitan la gestión de información, sino que también son fundamentales para el desarrollo de análisis complejos, contando con motores especializados para la generación de informes detallados (TIC, 2019).

PostgreSQL

Actualmente, PostgreSQL ha ganado reconocimiento como uno de los sistemas de gestión de bases de datos de código abierto más robustos del mercado. Sus características administrativas, así como su fiabilidad en cuanto a disponibilidad de datos, seguridad e integridad de la información en las últimas versiones, lo equiparan con opciones comerciales de renombre como Oracle y SQL Server.

PostgreSQL es compatible con la mayoría de las características del estándar ANSI SQL:2008, incluido el control de claves primarias y foráneas, uniones, vistas, disparadores y procedimientos almacenados. Además, incorpora la mayoría de los tipos de datos presentes en SQL:2008, como INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP e INTERVAL, junto con la capacidad de almacenar objetos binarios de gran tamaño, como imágenes, sonidos y vídeo (Narvaez, 2014).

En el ámbito profesional, los expertos en bases de datos suelen colaborar estrechamente con otros profesionales de la informática, la tecnología y disciplinas afines. Su labor se centra en la gestión de sistemas de datos que proporcionan la información necesaria para llevar a cabo diversas acciones profesionales (Helma, 2010).

PostgreSQL se configura como un sistema de gestión de bases de datos relacionales que opera a través de distintos niveles jerárquicos. El nivel más externo se denomina clúster, y en un mismo servidor es posible tener múltiples clústeres instalados, siempre y cuando cada uno de ellos utilice un puerto distinto, siendo el puerto predeterminado el 5432. Cada clúster contiene bases de datos, que a su vez albergan esquemas que contienen diversos objetos, tales como tablas, vistas, secuencias, entre otros, que se utilizan en el contexto de técnicas de almacenamiento de datos (Domínguez, 2020).

Base de datos relacionales

Según (PIÑEIRO GOMEZ, 2013), una base de datos puede definirse como una colección de depósito de datos integrados con redundancia controlada y una estructura que refleje las interrelaciones y restricciones existentes en el mundo real.

Los datos deben ser gestionados de manera independiente respecto a los usuarios y aplicaciones que los emplean. Es esencial que la definición y descripción de cada tipo de dato se mantengan junto a ellos para garantizar su integridad y comprensión adecuadas. Esto implica almacenar los metadatos asociados con los datos, lo que permite una gestión más efectiva y un uso más eficiente en diversos contextos de aplicación y usuarios (Departamento de ciencias de la computación e I.A, 2013).

Entre los diversos paradigmas adoptados en el ámbito de las tecnologías de la información, ninguno ha alcanzado una consolidación tan sólida y ampliamente aceptada como el modelo relacional en las bases de datos. Se podría argumentar que el éxito actual del enfoque orientado a objetos se atribuye a la firme implantación de este modelo en la implementación de sistemas de bases de datos (Osorio Rivera, 2008).

Además, según (Baxendale & Codd, 1970), el modelo relacional de bases de datos es un modelo de datos para la gestión de bases de datos inventado por E.F. Codd en 1970. Se basa en el principio de relaciones matemáticas y lógicas entre datos almacenados en tablas. Este modelo proporciona una estructura organizativa flexible y eficiente para el almacenamiento y recuperación de datos, lo que lo convierte en una opción preferida en numerosas aplicaciones de bases de datos.

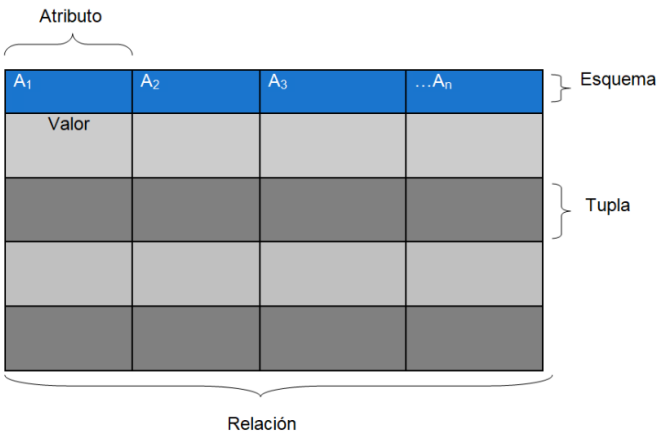


Figura 10: Estructura de una tabla de una base de datos relacional

Fuente: (Sánchez, 2004)

Lenguaje de consultas

El almacenamiento de información en bases de datos es crucial para la operación eficiente de cualquier organización en la era digital actual. Los Sistemas de Gestión de Bases de Datos (SGBD) desempeñan un papel fundamental al ocultar la complejidad asociada con la administración de los datos almacenados.

- **Lenguaje de consulta estructurado (SQL)**

El lenguaje de consulta estructurado (SQL) es un estándar de facto en la manipulación y gestión de datos en bases de datos relacionales. Normalizado por el Instituto Nacional Estadounidense de Estándares (ANSI), el SQL proporciona un conjunto de comandos que permiten realizar operaciones sobre los datos y la estructura de las bases de datos. Aunque existe una norma ANSI para SQL, cada motor de base de datos puede tener sus propias extensiones y peculiaridades, lo que puede generar diferencias en la sintaxis y el comportamiento del lenguaje entre diferentes sistemas de gestión de bases de datos (SGBD). Sin embargo, se puede garantizar que las sentencias SQL escritas según el estándar ANSI serán interpretables por cualquier motor de base de datos compatible (IBM, 2018).

1.2. Developer Operation (DevOps)

DevOps surge de la combinación de "desarrollo" y "operaciones", siendo un conjunto de prácticas y valores culturales basados en los principios ágiles del desarrollo de software. Su enfoque central es la integración continua (CI) y la entrega continua (CD) de software, promoviendo una nueva cultura en el desarrollo e implementación de software (Mamani Rodríguez et al., 2020).

DevOps se caracteriza como un cambio impulsado por la urgencia de eliminar las brechas entre los equipos de desarrollo y operaciones dentro de una empresa. Con este fin, se implementan tácticas como la integración y entrega en curso, las cuales no solo acortan el tiempo de lanzamiento al mercado, sino que también fomentan la excelencia del software. El triunfo de DevOps radica en su respaldo desde los niveles directivos más elevados de la empresa y en su adopción por parte de todos los departamentos implicados (Riti, 2018).

Actualmente, DevOps está ganando popularidad debido a su capacidad para proporcionar una entrega rápida y rentable de aplicaciones al mercado. Numerosas empresas, incluyendo Google, Netflix, Amazon, LinkedIn y Spotify, han adoptado ampliamente los principios de DevOps para mejorar la velocidad y calidad del despliegue de software. Sin embargo, la adopción de estas técnicas presenta desafíos, especialmente en términos de cambio cultural que afecta a todos los niveles y departamentos de una empresa. Además, varias fábricas de software han implementado DevOps con el fin de optimizar procesos y ofrecer servicios de alta calidad (Lucateli Bernardi et al., 2017).

1.2.1. Beneficios de DevOps

Numerosas empresas optan por adoptar DevOps con el fin de mejorar la calidad del software y la eficiencia en el despliegue. Este enfoque se basa en prácticas de integración y

entrega continua, lo que facilita la detección temprana de errores al integrar el código en el repositorio y permite desplegar el software directamente en el entorno de control de calidad de manera repetida (Riti, 2018).

De acuerdo con investigaciones realizadas por (Castillo et al., 2020), las organizaciones han identificado una serie de beneficios después de implementar prácticas de DevOps.

Figura 11: Beneficios de DevOps

Fuente: (Alberca, 2023)

1.3. Metodología de investigación

Antes de profundizar en los detalles de los distintos marcos de trabajo y enfoques metodológicos en el ámbito de la ingeniería de software, es importante establecer una base sólida sobre la importancia de la metodología de investigación en este campo. La metodología de investigación proporciona el marco estructural necesario para llevar a cabo un estudio de manera sistemática, rigurosa y científica. En el contexto de la ingeniería de software, esta metodología juega un papel crucial en la planificación, ejecución y evaluación de proyectos, así como en la generación de conocimiento y la toma de decisiones informadas.

1.3.1. Marco de trabajo Scrum

El marco de trabajo Scrum es una metodología ágil de gran popularidad en el ámbito del desarrollo de software y la gestión de proyectos, conocida por su capacidad para ajustarse a cambios rápidos y ofrecer entregas incrementales. Su origen se remonta a 1986, cuando Hirotaka Takeuchi e Ikujiro Nonaka lo presentaron por primera vez en un artículo de investigación en Harvard Business Review (Takeuchi & Nonaka, 1986). Desde entonces,

Scrum ha experimentado una evolución significativa y se ha consolidado como un enfoque ampliamente adoptado para la gestión de proyectos en diversos sectores industriales.

Para profundizar en el funcionamiento de Scrum y entender cómo se llevan a cabo las actividades dentro de este marco de trabajo ágil, podemos ver su proceso en la Figura 12.

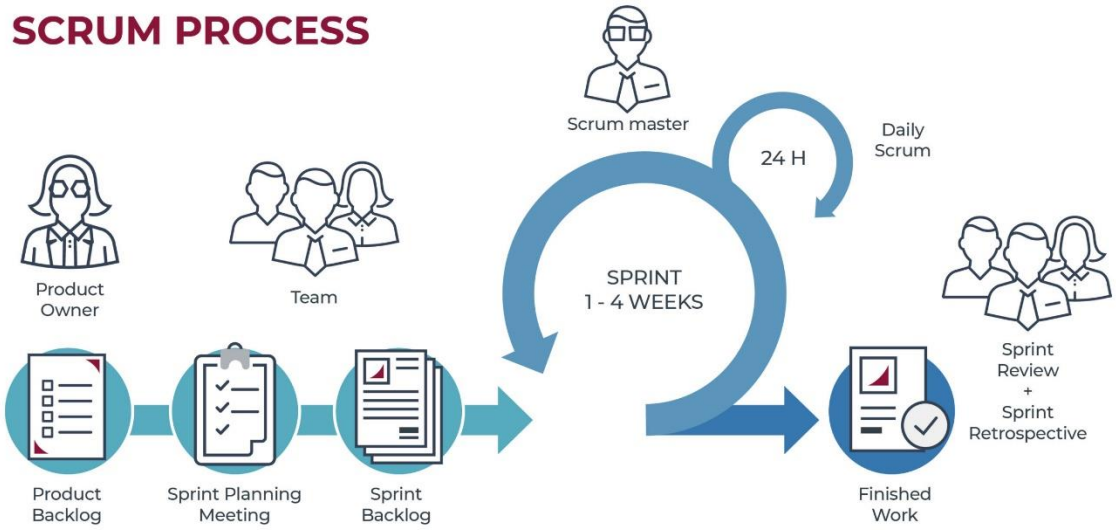


Figura 12: Proceso experimental

Fuente: (De Toro, 2022)

1.3.2. Experimentación en la Ingeniería de Software

La realización de experimentos en ingeniería de software facilita la identificación de las razones detrás de ciertos resultados. Sin embargo, este proceso no es simple, ya que implica una preparación meticulosa, la ejecución adecuada de los experimentos y un análisis exhaustivo de los datos obtenidos. Una de las principales ventajas de llevar a cabo experimentos es el control riguroso sobre los sujetos, objetos y la instrumentación utilizada. Este control garantiza la obtención de conclusiones más precisas y confiables, así como la posibilidad de realizar análisis estadísticos utilizando métodos de verificación de hipótesis (Wohlin et al., 2012).

Proceso de Experimentación

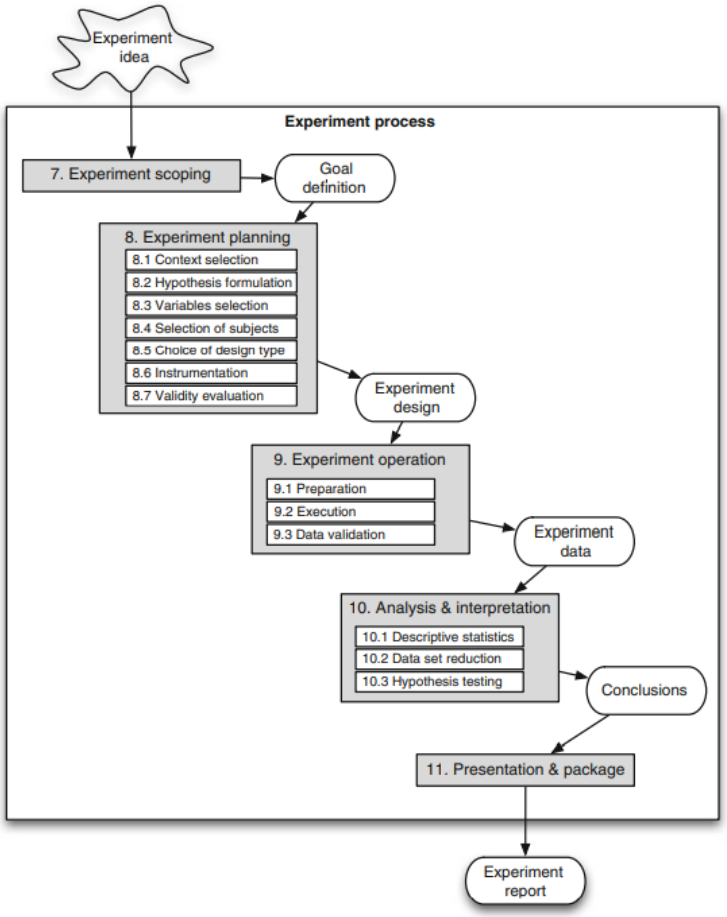


Figura 13: Proceso experimental

Fuente: (Wohlin et al., 2012)

Tabla 2: Proceso para la experimentación computacional

Procesos	Descripción
Determinación del alcance	En esta etapa se establecen de manera clara los objetivos, metas y el alcance del experimento. Es crucial definir el problema que se abordará y el objetivo que se busca alcanzar.
Planificación	Se detalla el contexto de la experimentación, incluyendo aspectos como el personal involucrado y el entorno en el que se llevará a cabo el experimento.
Operación	Esta fase comprende la preparación, ejecución y validación de los datos obtenidos durante el experimento. Se asegura

	que el proceso se lleve a cabo de manera adecuada y que los datos sean recolectados de manera precisa.
Análisis e interpretación	Una vez recopilados los datos, se procede a analizarlos e interpretarlos. Se busca identificar patrones, relaciones o tendencias que puedan proporcionar entendimientos relevantes para la investigación
Presentación y paquete	En esta última etapa, se presentan los resultados del experimento junto con una documentación detallada de los hallazgos. Es importante comunicar de manera clara y efectiva los resultados obtenidos y las conclusiones derivadas del análisis realizado.

Fuente: (Wohlin et al., 2012)

1.4. Estándares de calidad

La calidad medida y evaluada en base a modelos de calidad específicos de la organización no se puede comparar con la calidad de otros productos de software. Para aliviar este problema, ISO/IEC definió estándares internacionales denominados serie SQuaRE (Requisitos y evaluación de la calidad de los sistemas y el software) para la medición y evaluación integral de la calidad; sin embargo, estos estándares incluyen mediciones ambiguas, lo que dificulta su aplicación. En este documento se propone un marco inicial de medición de calidad integral, que incluye un plan de medición claro basado en ISO/IEC 25022 y 25023. Un estudio de caso confirma la utilidad del marco. Como trabajo futuro, introduciremos el marco en varios dominios. Y luego, revisamos y refinamos las medidas y los planes de evaluación para mejorar la viabilidad y la utilidad (Nakai et al., 2016).

Figura 14: Modelo de la calidad del producto

Fuente: (ISO 2510, 2015)

Medidas de Eficiencia de Desempeño

Las medidas de eficiencia del desempeño le permiten evaluar el rendimiento con relación a la cantidad de los recursos utilizados en determinadas condiciones. Además, se puede incluir otros productos de software como configuraciones de hardware y software del sistema (ISO 2510, 2015).

Este conjunto de métricas se desglosa en varias subcategorías, incluyendo el comportamiento del tiempo, que indica el nivel de satisfacción en cuanto a la respuesta, el tiempo de procesamiento y el rendimiento del sistema en la realización de sus funciones; la utilización de recursos, que mide si la cantidad y variedad de recursos utilizados por el sistema son adecuados para llevar a cabo su función; y la capacidad, que determina en qué medida el sistema cumple con los requisitos establecidos para sus parámetros..

Para evaluar la eficiencia de la Base de datos en contenedores Docker y en Localhost se aplicó la métrica de comportamiento del tiempo.

- **Tiempo medio de respuesta*:*** Hace referencia al tiempo medio en el que se demora una solicitud o un proceso asíncrono al completar (ISO 2510, 2015).

CAPÍTULO 2

2. Desarrollo del proyecto

Este capítulo presenta el proceso de creación de dos productos de software esenciales para respaldar el desarrollo de la investigación. En primer lugar, se detalla la elaboración de servicios API REST con base de datos PostgreSQL en un entorno localhost, que servirán como punto de referencia para el posterior análisis comparativo. Seguidamente, se describe la implementación de servicios API REST con base de datos PostgreSQL en contenedores Docker, los cuales se configuran para proporcionar un entorno más versátil y escalable. Ambos productos son desarrollados utilizando la metodología ágil Scrum, lo que facilita una gestión eficaz del proyecto y una adaptación flexible a los cambios en los requisitos y objetivos. El propósito principal de esta fase es crear la infraestructura necesaria para llevar a cabo la comparación del tiempo de respuesta entre los servicios API REST con base de datos en el entorno localhost y en contenedores Docker.

2.1. Análisis de servicios API REST a desarrollar.

2.1.1. Roles del proyecto

Durante la etapa de desarrollo del proyecto dentro del marco de trabajo Scrum, se designaron tres roles principales: el Scrum Master, el Product Owner y el Equipo de Desarrollo. En la Tabla 3 se especifican las funciones correspondientes a cada uno de estos roles, junto con la identificación de los integrantes del equipo asignados a cada uno.

Tabla 3: Roles y miembros en el desarrollo del proyecto

Miembro	Descripción	Rol
PhD. Antonio Quiña	Supervisor del presente trabajo de grado y profesor en la Universidad Técnica del Norte.	Dueño del producto (Product Owner).
Srta. Cinthya Arias	Estudiante de ingeniería de software en la Universidad Técnica del Norte	Líder del proyecto (Scrum Máster).
Srta. Cinthya Arias	Estudiante de ingeniería de software en la Universidad Técnica del Norte	Equipo de desarrollo (Development Team).

2.1.2. Levantamiento de requisitos

Durante la fase de levantamiento de requisitos del proyecto, se implementó un enfoque colaborativo en estrecha colaboración con el propietario del producto. Se priorizó el uso de historias de usuario como principal método para recopilar y documentar los requisitos, tanto funcionales como no funcionales, del sistema. Esta metodología promovió una comprensión detallada y compartida entre el equipo de desarrollo y el propietario del producto, garantizando una captura precisa y exhaustiva de las necesidades del cliente. Asimismo, esta colaboración facilitó una iteración continua y una retroalimentación constante, lo que contribuyó significativamente a alinear las expectativas del cliente con las capacidades del equipo de desarrollo a lo largo del proceso.

Tabla 4: Historia de Usuario N°1 – Registro de usuarios

	Historia de Usuario
ID: HU-01	**Rol:** Desarrollador Backend

Nombre historia: Registro de usuarios		
Prioridad: Alta	**Dependencia:** Ninguna	**Estimación:** 30
Descripción: Como desarrollador Backend, necesito contar con una funcionalidad que me permita registrar nuevos usuarios a través de un servicio API REST, facilitando así la incorporación de múltiples de usuarios de manera eficiente.		
Criterios de aceptación: - La aplicación debe permitir la creación de nuevos usuarios. - Debe ser posible generar registros de usuarios de forma automática, con la capacidad de especificar la cantidad deseada (1, 10, 100, 1000, 10000). - Todos los datos de usuario registrados deben ser almacenados correctamente en la base de datos. - Se debe medir y mostrar el tiempo de ejecución de la operación de registro de usuarios para garantizar la eficiencia del proceso.		

Tabla 5: Historia de Usuario N°2 – Consulta de usuarios

		Historia de Usuario
ID: HU-02	**Rol:** Desarrollador Backend	
Nombre: Consultar usuarios		
Prioridad: Alta	**Dependencia:** HU-01	**Estimación:** 30
Descripción: Como desarrollador Backend, necesito la capacidad de realizar consultas a la tabla de usuarios a través de un servicio REST para visualizar sus datos de manera efectiva.		
Criterios de aceptación: - La aplicación debe permitir la consulta de la tabla de usuarios utilizando la arquitectura REST.		

- Se debe implementar la visualización de la consulta para diferentes cantidades de registros (1, 10, 100, 1000, 10000). - Se debe medir y mostrar el tiempo de ejecución de la consulta para evaluar su eficiencia y rendimiento.

Tabla 6: Historia de Usuario N°3 – Consulta de cuentas por usuario

		Historia de Usuario
ID: HU-03	**Usuario:** Desarrollador Backend	
Nombre: Consultar cuentas por usuario		
Prioridad: Alta	**Dependencia:** Ninguna	**Estimación:** 30
Descripción: Como desarrollador Backend, necesito la capacidad de realizar consultas a una relación de tablas entre "users" y "account" a través de un servicio REST para visualizar sus datos de manera efectiva.		
Criterios de aceptación: - La aplicación debe permitir la consulta de la relación de tablas utilizando la arquitectura REST. - Se debe implementar la visualización de la consulta para diferentes cantidades de registros (1, 10, 100, 1000, 10000). - Se debe medir y mostrar el tiempo de ejecución de la consulta para evaluar su eficiencia y rendimiento.		

Tabla 7: Historia de Usuario N°4 – Consulta de movimientos de una cuenta con tipo de movimiento

	Historia de Usuario
ID: HU-04	**Usuario:** Desarrollador Backend
Nombre: Consultar movimientos de una cuenta con tipo de movimiento	

Prioridad: Alta	**Dependencia:** Ninguna	**Estimación:** 30
Descripción: Como desarrollador Backend, necesito la capacidad de realizar consultas a una relación de tablas entre "movements", "types_movement" y "account" a través de un servicio REST para visualizar sus datos de manera eficiente.		
Criterios de aceptación: - La aplicación debe permitir la consulta de la relación de tablas utilizando la arquitectura REST. - Se debe implementar la visualización de la consulta para diferentes cantidades de registros (1, 10, 100, 1000, 10000). - Se debe medir y mostrar el tiempo de ejecución de la consulta para evaluar su eficiencia y rendimiento.		

Tabla 8: Historia de Usuario N°5 – Consulta de todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta.

		Historia de Usuario
ID: HU-05	**Usuario:** Desarrollador Backend	
Nombre: Consultar todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta.		
Prioridad: Alta	**Dependencia:** N/A	**Estimación:** 30
Descripción: Como desarrollador Backend, requiero la capacidad de realizar consultas a una relación de tablas que incluya "movements", "types_movement", "account" y "types_account" mediante un servicio REST para visualizar sus datos de manera detallada y precisa.		
Criterios de aceptación: - La aplicación debe permitir la consulta de la relación de tablas utilizando la arquitectura REST.		

- Se debe implementar la visualización de la consulta para diferentes cantidades de registros (1, 10, 100, 1000, 10000).
- Se debe medir y mostrar el tiempo de ejecución de la consulta para evaluar su eficiencia y rendimiento.

2.1.3. Product Backlog

Después de detallar cada Historia de Usuario, se procedió a la creación del Product Backlog, que constituye una lista ordenada por prioridades de las funcionalidades y requisitos destinados. En la Tabla 9 se presenta el Product Backlog con las historias de usuario priorizadas según las decisiones del Product Owner. Este proceso de priorización facilita una visión clara y estructurada de las necesidades del cliente y orienta el desarrollo del proyecto de manera eficiente, asegurando que las funcionalidades más relevantes sean abordadas en primer lugar y adaptándose a las necesidades cambiantes del entorno durante el ciclo de desarrollo.

Tabla 9: Product Backlog

Orden	ID	Descripción	Nivel de consulta	Estimación
1	HU-01	Insertar usuarios.	1	30
2	HU-02	Consultar usuarios.	2	30
3	HU-03	Consultar cuentas por usuario	3	30
4	HU-04	Consultar movimientos de una cuenta con tipo de movimiento	4	30
5	HU-05	Consultar todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta.	5	30

2.1.4. Modelo de Base de Datos

En la Figura 13 se presenta el diseño de la base de datos desarrollado para respaldar la funcionalidad de un sistema bancario en una fase inicial de prueba de concepto, con un enfoque específico en el módulo de transferencias bancarias. Este diseño comprende cinco entidades distintas, cada una con atributos específicos que permiten representar de manera completa la estructura y las relaciones de los datos necesarios para el sistema. Para la creación y visualización de este diseño, se utilizó la herramienta DBeaver, lo que facilitó la gestión y comprensión global de la base de datos.

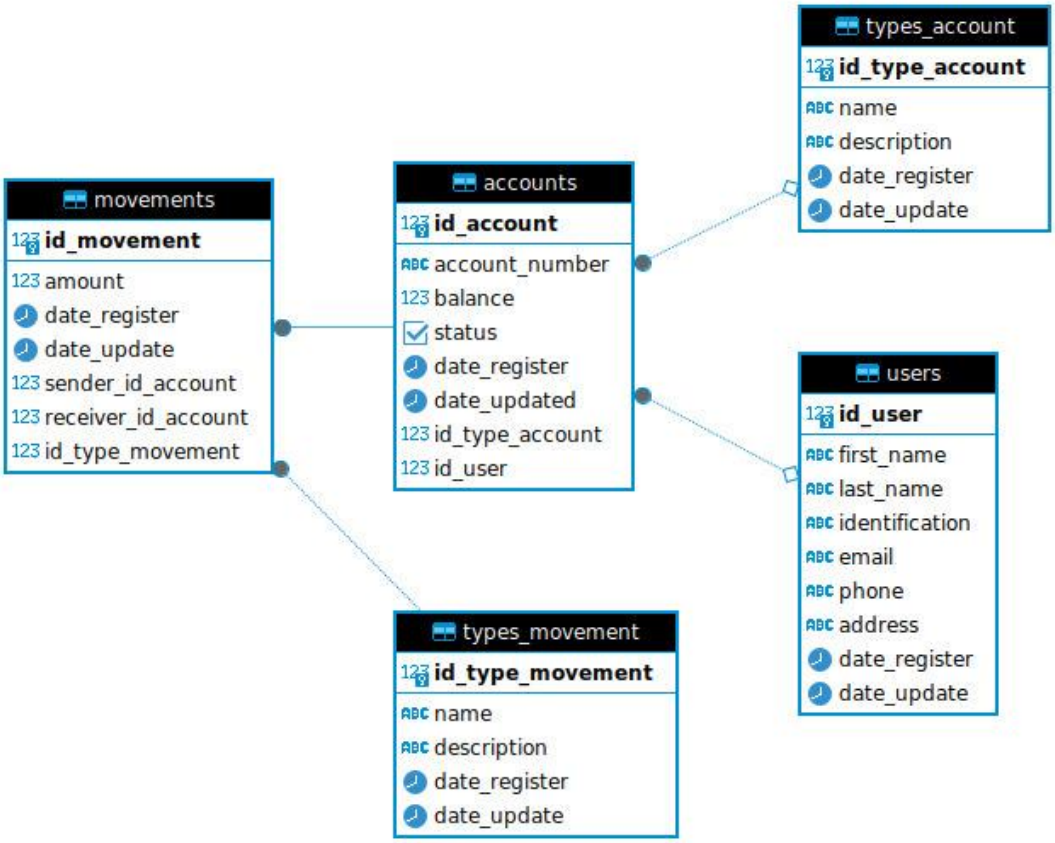


Figura 15: Diseño de la base de datos

2.1.5. Configuración de contenedores Docker

En esta sección, se detalla el proceso de configuración de los contenedores Docker para la implementación de una base de datos PostgreSQL. Se describe el contenido del archivo **docker-compose.yaml**, que define la estructura y configuración de los contenedores. Además, se explica el procedimiento para establecer la conexión con la base de datos dentro del entorno Docker y cómo acceder a la base de datos dockerizado.

Definición del archivo Docker-compose.yaml

El archivo docker-compose.yaml define la estructura y configuración de los servicios dentro del entorno Docker. A continuación, se presenta una explicación del contenido de este archivo:

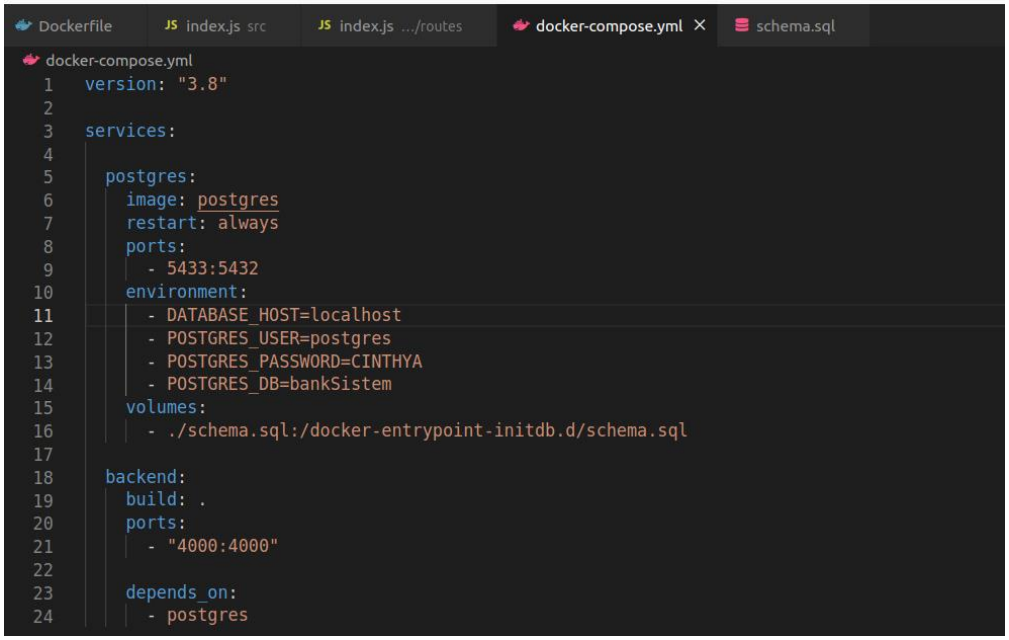


Figura 16: Archivo docker-compose.yaml

- **versión: “3.8”:** Indica la versión de la especificación de Compose que se está utilizando en este archivo.

- **services:** Define los servicios que se ejecutarán como contenedores.

- **postgres**: Define el servicio de la base de datos PostgreSQL.
  image: postgres: Especifica la imagen de Docker a utilizar para crear el contenedor de PostgreSQL.
  restart: always: Indica que el contenedor debe reiniciarse siempre que se detenga.
  ports: Mapea el puerto 5433 del host al puerto 5432 del contenedor, permitiendo acceder a la base de datos PostgreSQL desde el host.
  environment: Define las variables de entorno necesarias para la configuración de PostgreSQL:
  DATABASE_HOST=localhost: Especifica la dirección del host de la base de datos.
  POSTGRES_USER=postgres: Establece el nombre de usuario para acceder a PostgreSQL.
  POSTGRES_PASSWORD=CINTHYA: Establece la contraseña del usuario para acceder a PostgreSQL.
  POSTGRES_DB=bankSistem: Define el nombre de la base de datos a crear en PostgreSQL.
  volumes: Mapea el archivo schema.sql del host al directorio /docker-entrypoint-initdb.d dentro del contenedor, permitiendo que se ejecute al inicio para inicializar la base de datos.

- **backend:** Define el servicio para el Backend de la aplicación.
  build: Especifica cómo se debe construir el contenedor para este servicio.
  ports: Mapea el puerto 4000 del host al puerto 4000 del contenedor, permitiendo acceder al backend desde el host.
  depends_on: Indica que este servicio depende del servicio postgres, lo que garantiza que el contenedor de PostgreSQL se inicie antes de que se inicie el backend.

Este archivo docker-compose.yaml define la configuración para ejecutar dos servicios: uno para la base de datos PostgreSQL y otro para el backend de la aplicación. El servicio de PostgreSQL se configura con las variables de entorno necesarias y un volumen para inicializar la base de datos con el archivo schema.sql. El servicio de backend depende del servicio de PostgreSQL y se mapea al puerto 4000 para acceder a la aplicación desde el host.

Acceso a la base de datos dockerizado

Después de ejecutar el comando Docker-compose exec postgres psql -U postgres -d bankSistem -c "\dt" como se ve en la Figura 17, se accede al contenedor de la base de datos PostgreSQL, donde se especifica el usuario (-U postgres) y la base de datos (-d bankSistem). Posteriormente, se utiliza la opción "-c" para ejecutar un comando directamente en el cliente de PostgreSQL. En este caso, el comando "\dt" se emplea para listar todas las tablas creadas en la base de datos "bankSistem". Esto permite visualizar de manera rápida y sencilla la estructura de la base de datos y las tablas disponibles, lo cual es útil para verificar el estado y la organización de los datos almacenados.

```
cinthya@cinthya-HP-Pavilion-Laptop-15-cw0xxx:~/Documentos/TESIS/
DESARROLLO/docker/api-bank-system$ docker-compose exec postgres
psql -U postgres -d bankSistem -c "\dt"
            List of relations
 Schema |      Name      | Type  |  Owner
--------+----------------+-------+----------
 public | accounts       | table | postgres
 public | movements      | table | postgres
 public | types_account  | table | postgres
 public | types_movement | table | postgres
 public | users          | table | postgres
(5 rows)
```

Figura 17: Acceso a la base de datos dockerizado

2.2. Desarrollo del producto Backend.

El desarrollo de la API REST se organizará en cuatro sprints, cada uno con una duración de dos semanas. Durante cada sprint, se asignarán 30 horas para la implementación de tareas y funcionalidades específicas. La Tabla 10 presenta las fechas de inicio y finalización de cada sprint, junto con la duración total de cada actividad. Es importante destacar que el sprint 0 se detalló en la sección 2.1 de este capítulo.

Tabla 10: Detalle de los Sprints del Proyecto

Nro. Sprint	Fecha de inicio	Fecha fin	Duración (Horas)
Sprint 0	30/10/2023	12/11/2023	30
Sprint 1	13/11/2023	26/11/2023	30
Sprint 2	27/11/2023	10/12/2023	30
Sprint 3	11/12/2023	24/12/2023	30
Sprint 4	25/12/2023	07/01/2024	30

Durante los sprints, nos enfocaremos en completar estas las historias de usuario, asegurándonos de cumplir con todos los criterios de aceptación definidas. El equipo de desarrollo trabajará en colaboración con el Product Owner para garantizar que los requisitos de estas historias se cumplan de manera efectiva y dentro del plazo establecido.

2.2.1. Sprint 1

Para el sprint 1, nos enfocaremos en implementar las dos primeras historias de usuario, HU-01 y HU-02. La primera historia, "Insertar usuarios", tiene una alta prioridad y una estimación de 30 unidades de trabajo. En esta historia, se desarrollará un método para crear usuarios en la API REST, permitiendo la generación de varios registros de usuarios. Las pruebas de aceptación incluyen la validación de los campos, la generación de registros aleatorios de personas y su almacenamiento en la base de datos, así como la medición del tiempo de ejecución. La segunda historia, "Consultar usuarios", también tiene una alta prioridad y una estimación de 30 unidades de trabajo. En esta tarea, se implementará la funcionalidad para realizar consultas a la tabla de usuarios en la arquitectura REST, permitiendo la visualización de los datos. Las pruebas de aceptación incluirán la consulta de la tabla en diferentes cantidades de registros, desde 1 hasta 10,000, así como la medición del tiempo de ejecución de estas consultas.

- **Reunión de planificación del Sprint 1**

En la tabla a continuación se detallan los aspectos relevantes de la reunión de planificación del Sprint 1:

Tabla 11: Reunión de planificación – Sprint 1

Sprint 1	
Fecha de la reunión:	13 de noviembre del 2023
Presentes:	Scrum Master, Product Owner, Equipo de Desarrollo.
Objetivos:	Implementar un servicio para la generación aleatoria de usuarios. Desarrollar un servicio que permita consultar usuarios en diferentes escalas. Ejecutar los servicios de inserción y consulta de usuarios tanto en el entorno Localhost como en contenedores Docker.

Sprint Backlog – Sprint 1:

En la siguiente tabla se presenta el Sprint Backlog correspondiente al presente Sprint, detallando las diferentes tareas a desarrollar junto con su respectiva estimación de tiempo.

Tabla 12: Sprint Backlog – Sprint 1

Nro. HU	Nombre	Tarea	Horas
HU-01	Insertar usuarios	Implementar el endpoint de la API REST que maneje la solicitud de registro de usuarios, con métodos POST para enviar los datos del usuario.	6
		Integrar la lógica de medición del tiempo de ejecución para evaluar la eficiencia y el rendimiento del proceso de registro de usuarios	2
		Crear el componente controlador y añadir métodos POST para enviar los datos del usuario.	8
		Ejecución del servicio para verificar la funcionalidad de inserción en diferentes	10

		cantidades de registros. en ambiente Localhost y contenedores Docker.	
HU-02	Consultar usuarios	Implementar el endpoint de la API REST que maneje la solicitud de consulta de usuarios.	6
		Integrar la lógica de medición del tiempo de ejecución para evaluar la eficiencia y el rendimiento de la consulta, registrando los tiempos de manera adecuada.	2
		Desarrollar el controlador correspondiente para manejar la lógica de la consulta en el Backend, incluyendo la ejecución de la consulta SQL y la preparación de los datos para su visualización..	8
		Ejecución del servicio para verificar la funcionalidad de la consulta en diferentes cantidades de registros. en ambiente Localhost y contenedores Docker.	10

- **Reunión de revisión**

Después de finalizar el primer Sprint, se llevó a cabo la sesión de revisión con la participación del Scrum Master, el Product Owner y el Equipo de Desarrollo. Durante esta reunión, se verificó el cumplimiento de las historias de usuario de acuerdo con el incremento planificado del producto. La Tabla 13 presenta un resumen detallado de las tareas ejecutadas, comparando las estimaciones de tiempo con el tiempo real empleado en su ejecución.

Tabla 13: Reunión de revisión – Sprint 1

Responsable	Tarea	Tiempo estimado (horas)	Tiempo real (horas)	Estado
	Implementar el endpoint de la API REST que maneje la solicitud de registro de usuarios, con métodos POST para enviar los datos del usuario.	6	6	Realizado

	Integrar la lógica de medición del tiempo de ejecución para evaluar la eficiencia y el rendimiento del proceso de registro de usuarios	1	1	Realizado
Desarrollador	Crear el componente controlador y añadir métodos POST para enviar los datos del usuario.	8	8	Realizado
	Ejecución del servicio para verificar la funcionalidad de inserción en diferentes cantidades de registros. en ambiente Localhost y contenedores Docker.	8	8	Realizado
	Implementar el endpoint de la API REST que maneje la solicitud de consulta de usuarios.	6	6	Realizado
	Integrar la lógica de medición del tiempo de ejecución para evaluar la eficiencia y el rendimiento de la consulta, registrando los tiempos de manera adecuada.	2	2	Realizado
	Desarrollar el controlador correspondiente para manejar la lógica de la consulta en el Backend, incluyendo la ejecución de la consulta SQL y la preparación de los datos para su visualización.	8	8	Realizado
	Ejecución del servicio para verificar la funcionalidad de la consulta en diferentes cantidades de registros. en ambiente Localhost y contenedores Docker.	8	10	Realizado
	Planificación	1	1	Realizado

Reuniones Scrum	Revisión	1	1	Realizado
	Retrospectiva	1	1	Realizado
	Total	50	50	

Entregable (Incremento del producto)

En esta fase, se logró la implementación exitosa de los servicios para la inserción y consulta de usuarios en ambos ambientes.

Funcionalidad Servicio

Se procede a realizar pruebas de funcionalidad de los servicios de inserción y consulta de usuarios en REST mediante el cliente Postman.

- **Generar usuarios.**

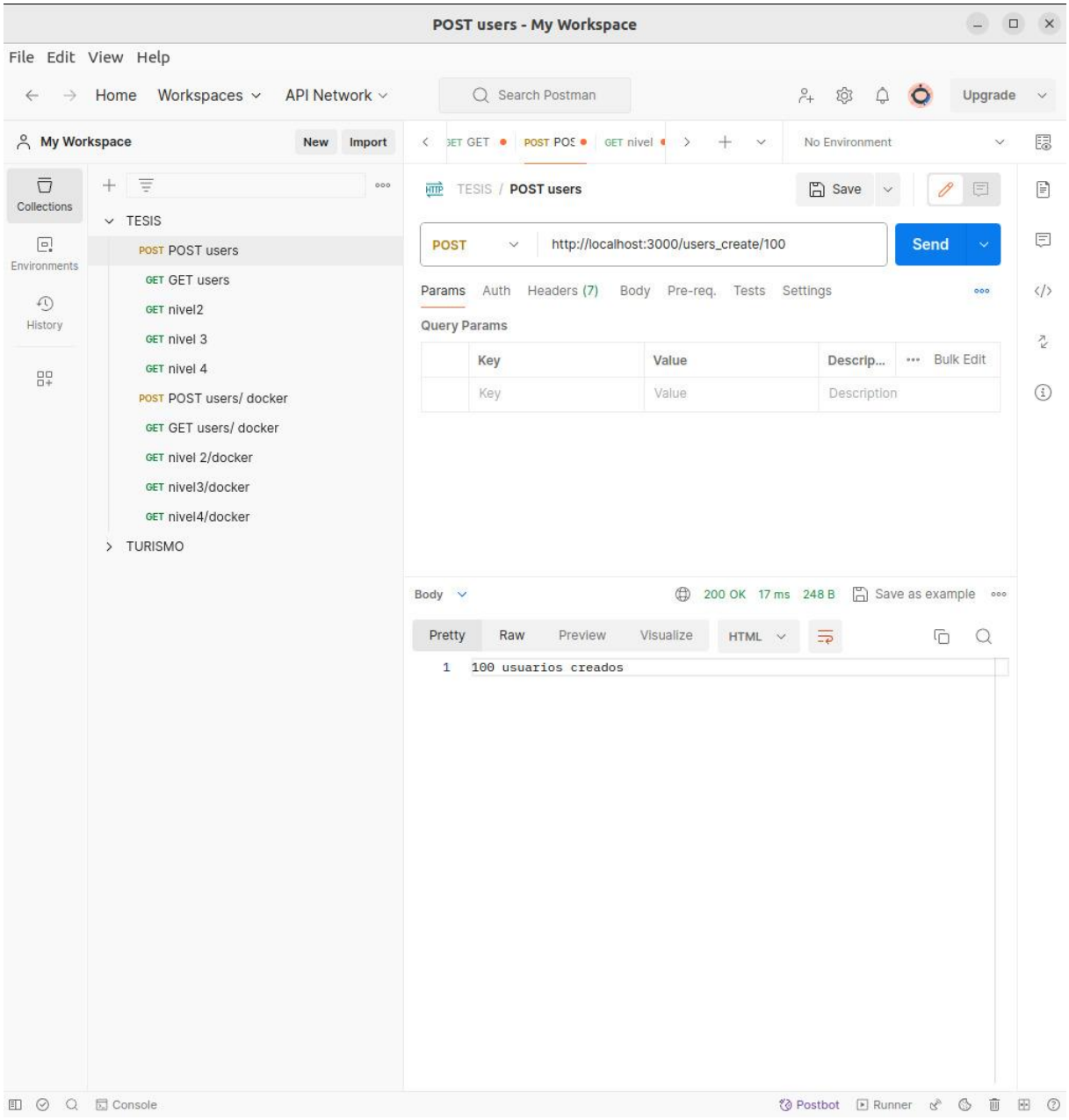


Figura 18: Petición para generar usuarios

- **Consultar usuarios.**

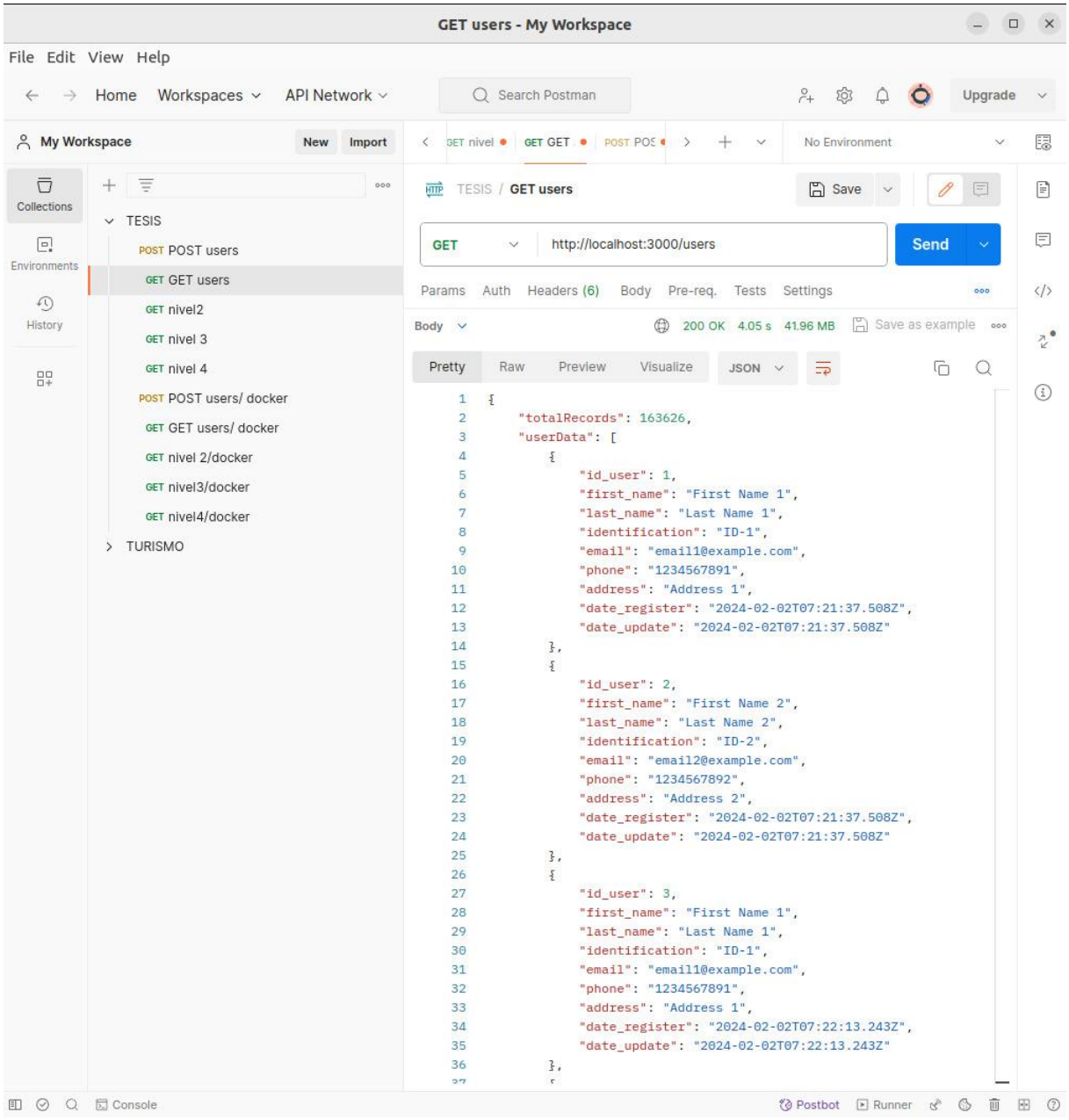


Figura 19: Petición para obtener usuarios.

2.2.2. Sprint 2

Para el sprint 2, nos concentraremos en implementar la tercera historia de usuario, HU-03. Esta historia, titulada "Consultar cuentas por usuario", tiene una prioridad alta y una estimación de 30 unidades de trabajo. En esta tarea, se desarrollará la funcionalidad para realizar consultas a una relación de tablas entre "users" y "account" en la arquitectura REST, lo que permitirá la visualización de los datos relacionados. Las pruebas de aceptación incluirán la realización de consultas a la tabla en la arquitectura REST y la visualización de los resultados para diferentes cantidades de registros, que van desde 1 hasta 10,000. Además,

se medirá el tiempo de ejecución de estas consultas para garantizar un rendimiento óptimo del servicio.

- **Reunión de planificación del Sprint 2**

En esta reunión, que tuvo lugar el 27 de noviembre de 2023, participaron el Scrum Master, el Product Owner y el equipo de desarrollo. Durante la sesión, se establecieron los siguientes objetivos para el Sprint 2:

Tabla 14: Reunión de planificación – Sprint 2

Sprint 2	
Fecha de la reunión:	27 de noviembre del 2023
Presentes:	Scrum Master, Product Owner, Equipo de desarrollo
Objetivos:	Implementar un servicio que permita realizar la consulta de cuentas por usuario en diferentes tamaños. Ejecutar el servicio de consulta de cuentas por usuarios en ambiente Localhost y contenedores Docker.

Sprint Backlog – Sprint 2

La Tabla 15 detalla el Sprint Backlog correspondiente al presente Sprint, donde se describen las tareas a desarrollar y su estimación de tiempo.

Tabla 15: Sprint Backlog – Sprint 2

Nro. HU	Nombre	Tarea	Horas
HU-03	Consultar cuentas por usuario	Implementar el endpoint de la API REST que maneje la solicitud de consulta de cuentas por usuario.	6
		Integrar la lógica de medición del tiempo de ejecución para evaluar la eficiencia y el rendimiento de la consulta.	2
		Desarrollar el controlador correspondiente para manejar la lógica de la consulta en el backend.	8
		Ejecución del servicio para verificar la funcionalidad de la consulta en diferentes cantidades de registros. en ambiente Localhost y contenedores Docker.	10

- **Reunión de revisión**

Tabla 16: Reunión de revisión – Sprint 2

Responsable	Tarea	Tiempo estimado (horas)	Tiempo real (horas)	Estado
Desarrollador	Implementar el endpoint de la API REST que maneje la solicitud de consulta de cuentas por usuario.	6	6	Realizado
	Integrar la lógica de medición del tiempo de ejecución para evaluar la eficiencia y el rendimiento de la consulta.	2	2	Realizado
	Desarrollar el controlador correspondiente para manejar la lógica de la consulta en el backend.	8	8	Realizado
	Ejecución del servicio para verificar la funcionalidad de la consulta en diferentes cantidades de registros. en ambiente Localhost y contenedores Docker.	8	10	Realizado
Reuniones Scrum	Planificación	1	1	Realizado
	Revisión	1	1	Realizado
	Retrospectiva	1	1	Realizado
	Total	27	27	

Entregable (Incremento del producto)

En este sprint se logró implementar el servicio de consulta de cuentas en ambiente Localhost como en contenedores Docker.

Funcionalidad Servicio

Se lleva a cabo la prueba de funcionalidad de los servicios de consulta utilizando el cliente Postman.

- **Consultar cuentas por usuarios.**

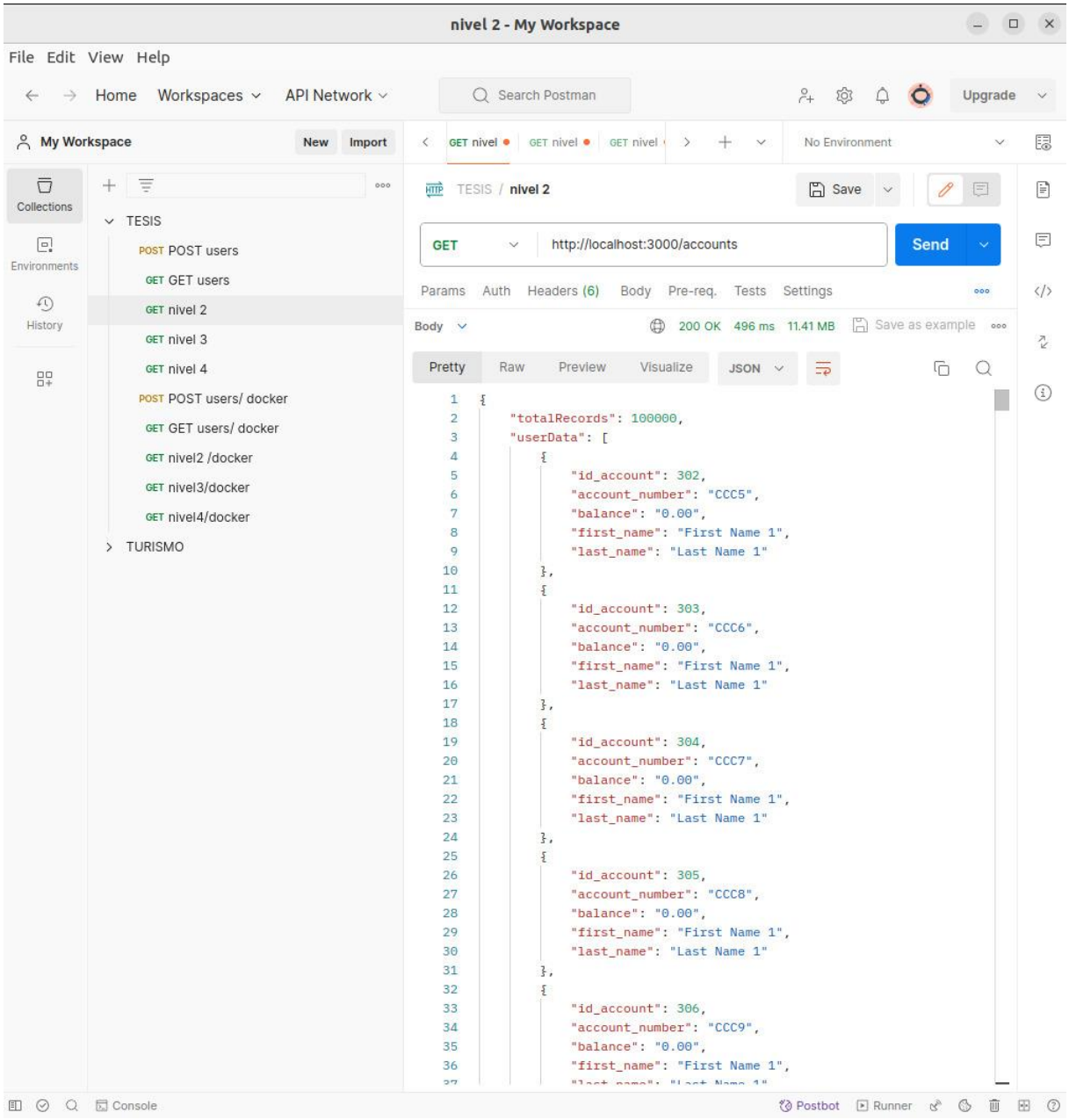


Figura 20: Petición para obtener cuentas por usuario.

2.2.3. Sprint 3

Para el sprint 3, nos enfocaremos en implementar la cuarta historia de usuario, HU-04. Esta historia, denominada "Consultar movimientos de una cuenta con tipo de movimiento", tiene una prioridad alta y una estimación de 30 unidades de trabajo. En esta tarea, se desarrollará la funcionalidad para realizar consultas a una relación de tablas entre "movements", "types_movement" y "account" en la arquitectura REST, lo que permitirá la visualización de los datos relacionados con los movimientos de una cuenta y su tipo de

movimiento asociado. Las pruebas de aceptación consistirán en la realización de consultas a la tabla en la arquitectura REST y la visualización de los resultados para diferentes cantidades de registros, que van desde 1 hasta 10,000. Asimismo, se medirá el tiempo de ejecución de estas consultas para garantizar un rendimiento óptimo del servicio.

- **Reunión de planificación**

Tabla 17: Reunión de planificación – Sprint 3

Sprint 3	
Fecha de la reunión:	11 de diciembre del 2023
Presentes:	Scrum Master, Product Owner, Equipo de Desarrollo.
Objetivos:	Implementar un servicio que permita realizar la consulta de movimientos de una cuenta con tipo de movimiento. Ejecutar el servicio de consulta de movimientos de una cuenta con tipo de movimiento en ambiente Localhost y contenedores Docker

Sprint Backlog – Sprint 3

La Tabla 18 presenta el Sprint Backlog detallando las diferentes tareas que se va a desarrollar.

Tabla 18: Sprint Backlog – Sprint 3

Nro. HU	Nombre	Tarea	Horas
HU-04	movimientos de una cuenta con tipo de movimiento	Implementar el endpoint de la API REST que maneje la solicitud de consulta de movimientos de una cuenta con tipo de movimiento.	6
		Integrar la lógica de medición del tiempo de ejecución para evaluar la eficiencia y el rendimiento de la consulta.	2
		Desarrollar el controlador correspondiente para manejar la lógica de la consulta en el backend.	8
		Ejecución del servicio para verificar la funcionalidad de la consulta en diferentes	10

	cantidades de registros. en ambiente Localhost y contenedores Docker.

a) Reunión de revisión

Tabla 19: Reunión de revisión – Sprint 3

Responsable	Tarea	Tiempo estimado (horas)	Tiempo real (horas)	Estado
Desarrollador	Implementar el endpoint de la API REST que maneje la solicitud de consulta de movimientos de una cuenta con tipo de movimiento.	6	6	Realizado
	Integrar la lógica de medición del tiempo de ejecución para evaluar la eficiencia y el rendimiento de la consulta.	2	2	Realizado
	Desarrollar el controlador correspondiente para manejar la lógica de la consulta en el Backend.	8	8	Realizado
	Ejecución del servicio para verificar la funcionalidad de la consulta en diferentes cantidades de registros. en ambiente Localhost y contenedores Docker.	8	10	Realizado
Reuniones Scrum	Planificación	1	1	Realizado
	Revisión	1	1	Realizado
	Retrospectiva	1	1	Realizado
	Total	27	27	

Entregable (Incremento del producto)

En este sprint se obtuvo como resultado la implementación del servicio de consulta de movimientos de una cuenta con tipo de movimiento tanto en ambiente Localhost como en contenedores Docker.

Funcionalidad Servicio

Se lleva a cabo la prueba de funcionalidad de los servicios de consulta utilizando el cliente Postman.

- **Consultar movimientos de una cuenta con tipo de movimiento.**

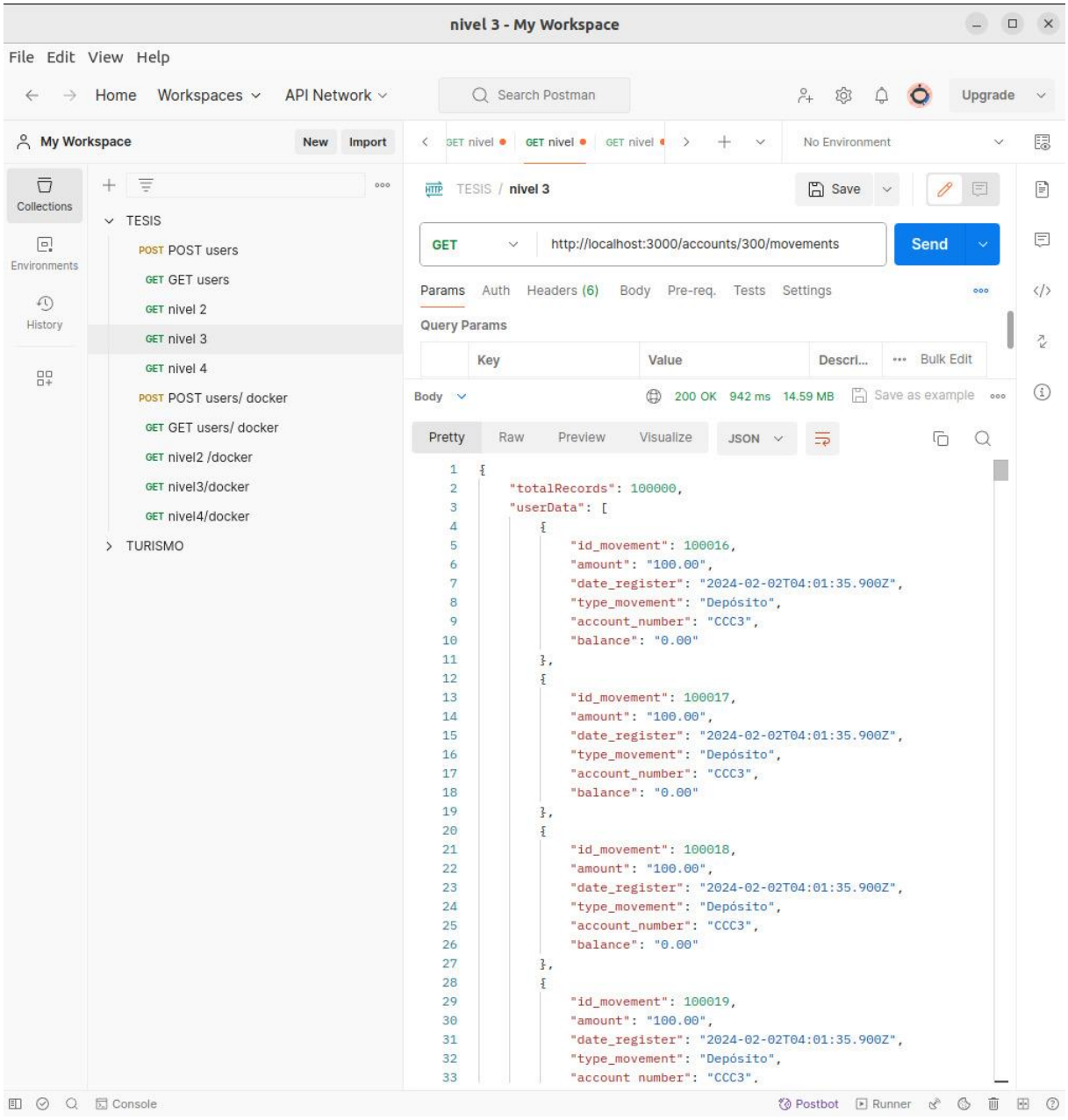


Figura 21: Petición para obtener movimientos de una cuenta con tipo de movimiento.

2.2.4. Sprint 4

En este sprint, nos enfocaremos en implementar la historia de usuario HU-MT-05, "Consultar todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta". Esta tarea de alta prioridad requiere 30 unidades de trabajo y consiste en desarrollar la funcionalidad para consultar una relación de tablas en la arquitectura REST, que incluye "movements", "types_movement", "account" y "types_account". Las pruebas de aceptación abarcan desde la consulta de la tabla hasta la visualización de los resultados en diferentes cantidades de registros, evaluando también el tiempo de ejecución de estas consultas.

- **Reunión de planificación**

Tabla 20: Reunión de planificación – Sprint 4

Sprint 4	
Fecha de la reunión:	25/12/2023
Presentes:	Scrum Master, Product Owner, Equipo de Desarrollo.
Objetivos:	Implementar un servicio que permita realizar la consulta de todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta. Ejecutar el servicio de consulta de todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta en ambiente Localhost y contenedores Docker

Sprint Backlog – Sprint 4

Tabla 21: Sprint Backlog – Sprint 4

Nro. HU	Nombre	Tarea	Horas
HU-05	Consultar todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta	Implementar el endpoint de la API REST que maneje la solicitud de consulta de todos los movimientos de un usuario con detalles de cuenta y tipo de cuenta.	6
		Integrar la lógica de medición del tiempo de ejecución para evaluar la eficiencia y el	2

	rendimiento de la consulta, registrando los tiempos de manera adecuada.	
	Desarrollar el controlador correspondiente para manejar la lógica de la consulta en el Backend, incluyendo la unión de las tablas y la preparación de los datos para su visualización.	8
	Ejecución del servicio para verificar la funcionalidad de la consulta en diferentes cantidades de registros. en ambiente Localhost y contenedores Docker.	10

a) Reunión de revisión

Tabla 22: Reunión de revisión – Sprint 4

Responsable	Tarea	Tiempo estimado (horas)	Tiempo real (horas)	Estado
Desarrollador	Implementar el endpoint de la API REST que maneje la solicitud de consulta de todos los movimientos de un usuario con detalles de cuenta y tipo de cuenta.	6	6	Realizado
	Integrar la lógica de medición del tiempo de ejecución para evaluar la eficiencia y el rendimiento de la consulta, registrando los tiempos de manera adecuada.	2	2	Realizado
	Desarrollar el controlador correspondiente para manejar la lógica de la consulta en el Backend, incluyendo la unión de las tablas y la preparación de los datos para su visualización.	8	8	Realizado

	Ejecución del servicio para verificar la funcionalidad de la consulta en diferentes cantidades de registros. en ambiente Localhost y contenedores Docker.	8	10	Realizado
	Planificación	1	1	Realizado
Reuniones Scrum	Revisión	1	1	Realizado
	Retrospectiva	1	1	Realizado
	Total	27	27	

Entregable (Incremento del producto)

En este sprint se obtuvo como resultado la implementación del servicio de todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta tanto en ambiente Localhost como en contenedores Docker.

Funcionalidad API – REST

Se lleva a cabo la prueba de funcionalidad de los servicios de consulta utilizando el cliente Postman.

- **Consultar todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta.**

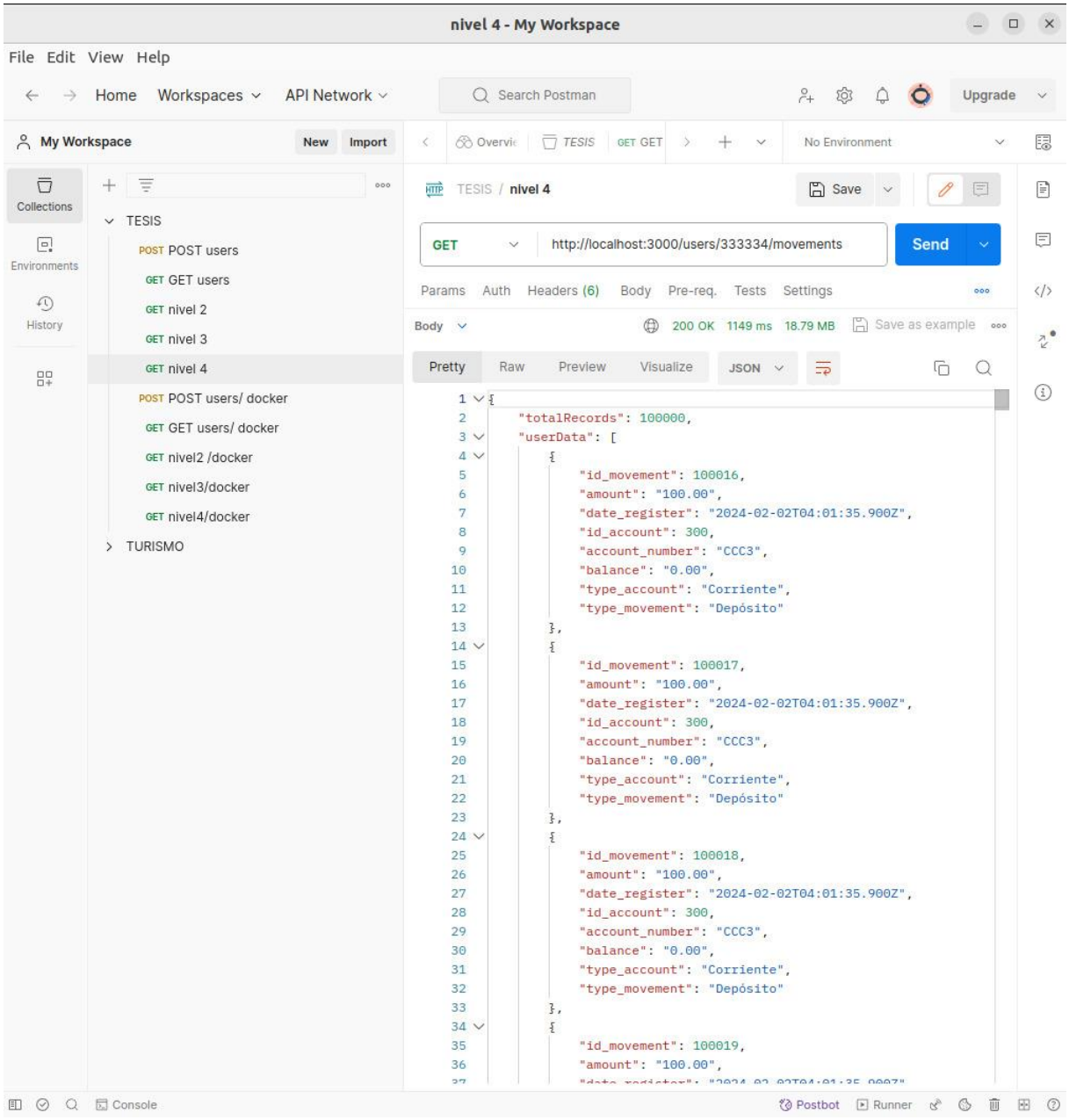


Figura 22: Petición para obtener todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta.

2.3. Pruebas de aceptación.

Se realizaron pruebas de aceptación ejecutando la aplicación tanto en el entorno Localhost como en los contenedores Docker. El objetivo principal de estas pruebas fue validar el correcto funcionamiento de las funcionalidades implementadas y asegurar que los servicios de la API REST cumplieran con los requisitos establecidos. En la Tabla 21 se presenta un conjunto de funcionalidades junto con su estado de aceptación por parte del Product Owner. Esta tabla proporciona una visión clara y concisa del nivel de aceptación asignado a cada funcionalidad por parte del responsable del producto.

Tabla 23: Pruebas de aceptación ambiente Localhost

Historia de Usuario	Nombre	Funcionalidad	Aceptación	
			SI	NO
HU-01	Insertar usuario	Generar usuarios aleatorios en ambiente Localhost.	X	
HU-02	Consultar usuarios	Consultar usuarios en ambiente Localhost.	X	
HU-03	Consultar cuentas por usuario	Consultar cuentas por usuario en ambiente Localhost.	X	
HU-04	Consultar movimientos de una cuenta con tipo de movimiento	Consultar movimientos de una cuenta con tipo de movimiento en ambiente Localhost	X	
HU-05	Consultar todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta	Consultar todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta en ambiente Localhost	X	

Tabla 24: Pruebas de aceptación contenedor Docker

Historia de Usuario	Nombre	Funcionalidad	Aceptación	
			SI	NO
HU-01	Insertar usuario	Generar usuarios aleatorios en contenedor Docker.	X	
HU-02	Consultar usuarios	Consultar usuarios en contenedor Docker.	X	
HU-03	Consultar cuentas por usuario	Consultar cuentas por usuario en contenedor Docker.	X	
HU-04	Consultar movimientos de una cuenta con tipo de movimiento	Consultar movimientos de una cuenta con tipo de movimiento en contenedor Docker.	X	
HU-05	Consultar todos los movimientos de un	Consultar todos los movimientos de un usuario con detalle de	X	

usuario con detalle de cuenta y tipo de cuenta	cuenta y tipo de cuenta en contenedor Docker.

2.4. Comparación de Rendimiento entre Localhost y Docker.

En esta sección, se presenta una comparación del rendimiento entre el ambiente Localhost y Docker para diferentes casos de uso (CU) en el contexto del experimento realizado. Los datos muestran el tiempo medio de respuesta para cada caso de uso en ambos entornos, calculado como la media de todos los registros de las 3 repeticiones en cada ambiente.

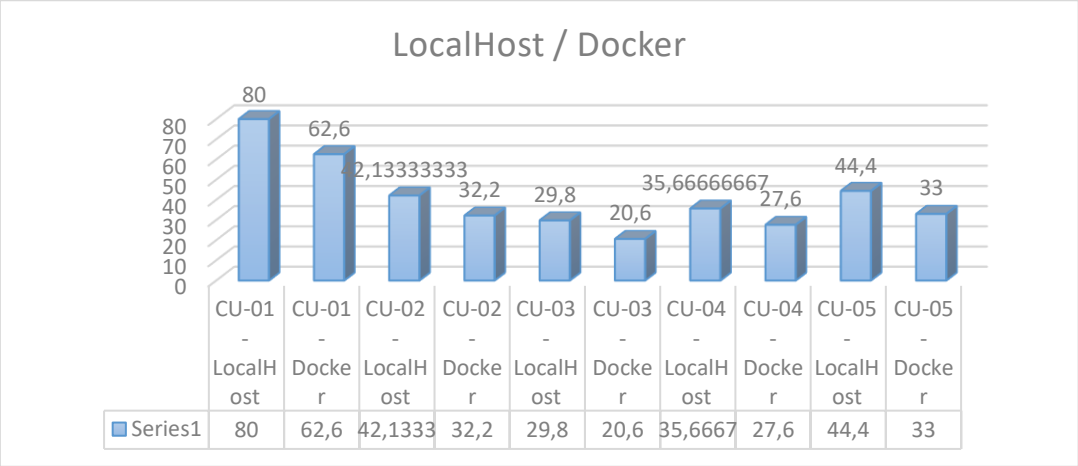


Figura 23: Comparación de Rendimiento entre Localhost y Docker.

Los valores proporcionados representan el tiempo medio de respuesta en milisegundos para cada caso de uso tanto en el ambiente Localhost como en Docker. Se calcularon como la media de todos los registros de las 3 repeticiones en cada ambiente. El detalle completo y análisis de estos registros se detalla en el CAPÍTULO 3.

CAPÍTULO 3

Validación de resultados

En este capítulo, se llevó a cabo el desarrollo de la investigación mediante un experimento controlado, siguiendo la guía de "*Experimentación en Ingeniería de Software*" de Wohlin (2012) como referencia. El experimento se enfocó en comparar el rendimiento de la base de datos PostgreSQL en entornos locales y en contenedores Docker. En este contexto, se planteó la siguiente pregunta de investigación: ***PI***: ¿Qué ambiente es más eficiente para la ejecución de bases de datos PostgreSQL en arquitecturas DevOps? Para medir la eficiencia

del rendimiento de estos dos ambientes se utilizó la métrica “tiempo medio de respuesta” de la norma ISO/IEC 25023.

En el contexto del desarrollo de software, es común encontrarnos con la ausencia de pruebas sólidas que respalden la adecuación, calidad, rendimiento, costos y riesgos de las tecnologías empleadas. Esta falta de evidencia puede generar incertidumbre y limitar la toma de decisiones fundamentadas, por lo tanto, en este estudio se utiliza experimentación de software como una herramienta para fundamentar empíricamente la respuesta a la pregunta de investigación planteada. A través de la experimentación, es posible obtener datos concretos que respalden la selección de tecnologías, procesos y enfoques de desarrollo, proporcionando una base sólida para la toma de decisiones informadas (Jedlitschka et al., 2008)**.**

3.1. Entorno Experimental

3.1.1. Objetivo del experimento

El objetivo principal de este experimento radica en comparar la eficiencia del rendimiento de la base de datos PostgreSQL en dos entornos distintos: Localhost y contenedores Docker. El enfoque se centra en analizar el tiempo medio de respuesta en cada entorno para evaluar su eficacia y determinar diferencias significativas en cuanto a rendimiento.

3.1.2. Factores y tratamientos

El factor para investigar es la base de datos PostgreSQL en ambos ambientes con la ayuda de servicios API. Los tratamientos que se emplearon fueron:

- Base de datos PostgreSQL en ambiente Localhost con servicios API REST.
- Base de datos PostgreSQL en contenedores Docker con servicios API REST.

3.1.3. Variables

A continuación, se observa la distribución de las variables utilizadas en el estudio experimental. La variable independiente se refiere a la configuración del entorno de base de datos PostgreSQL, distinguiendo entre el ambiente Localhost con servicios API REST y los contenedores Docker con servicios API REST. Por otro lado, la variable dependiente se define como la calidad del producto de software, siendo evaluada específicamente en función de la característica de eficiencia del rendimiento.

Tabla 25: Variables para la experimentación

Variable Independiente	El ambiente de ejecución de base de datos PostgreSQL en arquitecturas DevOps mediante API-REST
Variable Dependiente	La calidad del producto de software, tomando en cuenta la característica eficiencia del rendimiento.

Para evaluar la calidad del sistema y del producto de software, se utiliza la característica de "*Medidas de Eficiencia del Desempeño*" definida en la norma ISO/IEC 25023. Esta métrica proporciona un marco estándar para cuantificar la eficiencia operativa de un sistema informático, abarcando aspectos como la rapidez de respuesta, la utilización de recursos y la capacidad de procesamiento. En particular, se centra en métricas específicas como el tiempo medio de respuesta de las aplicaciones.

Métrica: Tiempo medio de respuesta

Se refiere al lapso que transcurre desde que se realiza una solicitud hasta su finalización, representando el tiempo promedio empleado en procesos asíncronos. En el marco de nuestro experimento, se empleó una función de medición específica para registrar y evaluar el tiempo de respuesta de las solicitudes, tanto en ambiente Localhost como en contenedores Docker.

Con esta métrica se responde a la pregunta: ¿Cuál es el tiempo medio empleado para completar un trabajo o un proceso asíncrono?

La ecuación para calcular esta métrica se define de la siguiente manera:

$$X = \sum_{i=1\ an} (B_i = A_i)/n$$

Donde:

$\boldsymbol{A_i}$ = Representa el tiempo de inicio del trabajo $\boldsymbol{i}$

$\boldsymbol{B_i}$ = Representa el tiempo necesario para completar el trabajo $\boldsymbol{i}$

$\boldsymbol{n}$ = Es el número total de mediciones realizadas.

3.1.4. Hipótesis

En esta sección, se formularon las preguntas de investigación en función de la pregunta principal de investigación, $\boldsymbol{PI}$.

PI_1 ¿Cuál de los entornos, ambiente local o contenedores Docker, es más eficiente para la ejecución de base de datos PostgreSQL?

Considerando esta interrogativa de investigación, se establece las siguientes hipótesis para el experimento:

- H_0: ***(Hipótesis nula):*** No hay diferencia en la medición de calidad del producto del software al ejecutar bases de datos en contenedores Docker o localhost.

- H_1: ***(Hipótesis alternativa 1):*** Existe diferencia en la medición de calidad del producto del software al ejecutar la base de datos en ambiente Localhost. La calidad del producto al ejecutar bases de datos en ambiente localhost es superior a la calidad del producto que produce en contenedor Docker.

- H_2: ***(Hipótesis alternativa 2):*** Existe diferencia en la medición de calidad del producto del software al desarrollar la base de datos en contenedor Docker. La calidad del producto al ejecutar bases de datos en contenedor Docker es superior a la calidad del producto que produce en ambiente Localhost.

3.1.5. Diseño

El experimento se ha concebido con el propósito de establecer un marco adecuado que facilite la comparación de la eficiencia entre la implementación de una base de datos en un entorno Localhost y su equivalente en un contenedor Docker, a través de la configuración de un entorno de laboratorio experimental. Para alcanzar este objetivo, se han delineado dos tareas experimentales distintas: la primera tarea se enfoca en el desarrollo de la base de datos en el contexto de un entorno Localhost, complementado con servicios API REST, mientras que la segunda tarea aborda el mismo desarrollo, pero dentro de un contenedor Docker, también con servicios API REST. Los detalles específicos de cada una de estas tareas, así como los casos de uso asociados, se encuentran detallados en la Tabla 26, que resalta cinco escenarios distintos que serán ejecutados en ambas configuraciones de entorno. Cada caso de uso será repetido en tres iteraciones con registros de datos variables. Para evaluar la eficiencia en cada caso, se utilizó la métrica descrita en la sección 3.1.3 del experimento, la cual facilitará la obtención de tiempos de respuesta precisos en cada ejecución.

Tabla 26: Diseño del experimento

Caso de Uso	Repeticiones	Servicios en Localhost	Servicios en Docker

CU - 01	3	Registros: 1, 10, 100, 1000, 10000.	Registros: 1, 10, 100, 1000, 10000.
CU - 02	3	Registros: 1, 10, 100, 1000, 10000.	Registros: 1, 10, 100, 1000, 10000.
CU - 03	3	Registros: 1, 10, 100, 1000, 10000.	Registros: 1, 10, 100, 1000, 10000.
CU - 04	3	Registros: 1, 10, 100, 1000, 10000.	Registros: 1, 10, 100, 1000, 10000.
CU - 05	3	Registros: 1, 10, 100, 1000, 10000.	Registros: 1, 10, 100, 1000, 10000.

Casos de Uso

En esta sección de la experimentación, se detalla los casos de uso que contiene los servicios API REST en cada uno de los ambientes. Se estableció cinco casos de uso con diferentes niveles de complejidad:

Caso de uso 1: Insertar usuarios

- Nivel de consulta :1
- Campos: **users**: id_user, first_name, last_name, identification, email, phone, address, date_register, date_update.

Caso de uso 2: Consultar usuarios

- Nivel de consulta: 1
- Campos: **users**: id_user, first_name, last_name, identification, email, phone, address, date_register, date_update.

Caso de uso 3: Consultar cuentas por usuario

- Nivel de consulta: 2
- Campos: **accounts:** id_account, account_number, balance, status, date_register, date_updated, id_type_account, id_user. **users**: id_user, first_name, last_name, identification, email, phone, address, date_register, date_update.

Caso de uso 4: Consultar movimientos de una cuenta con tipo de movimiento

- Nivel de consulta: 3

- Campos: **movements:** id_movement, amount, date_register, date_update, sender_id_account, receiver_id_account, id_type_movement. **types_movement**: id_type_movement, name, description,date_register, date_update. **accounts:** id_account, account_number, balance, status, date_register, date_updated, id_type_account, id_user. **users**: id_user, first_name, last_name, identification, email, phone, address, date_register, date_update.

Caso de uso 5: Consultar todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta.

- Nivel de consulta :4
- Campos: **movements:** id_movement, amount, date_register, date_update, sender_id_account, receiver_id_account, id_type_movement. **accounts:** id_account, account_number, balance, status, date_register, date_updated, id_type_account, id_user. **types_movement:** id_type_movement, name, description, date_register, date_update. **types_account:** id_type_account, name, description, date_register, date_update.

3.1.6. Tareas experimentales

En esta sección se detallan las tareas experimentales que se desarrollaron en el CAPÍTULO 2. Mediante estas tareas experimentales, se configuró un laboratorio experimental, como se ilustra en la Figura 18.

Tarea experimental 1: Desarrollo de la base de datos en ambiente Localhost con servicios API REST.

El objetivo de esta tarea es ejecutar los casos de uso en ambiente Localhost. Para lo cual, se debe desarrollar una aplicación cliente que implemente los casos de uso detallado en la sección 2.1 y su diseño de experimento detallada en la Tabla 26.

Tarea experimental 2: Desarrollo de la base de datos en contenedor Docker con servicios API REST.

El objetivo de esta tarea es ejecutar los casos de uso en contenedor Docker. Para lo cual, se debe desarrollar una aplicación cliente que implemente los casos de uso detallado en la sección 2.1 y su diseño de experimento detallada en la Tabla 26.

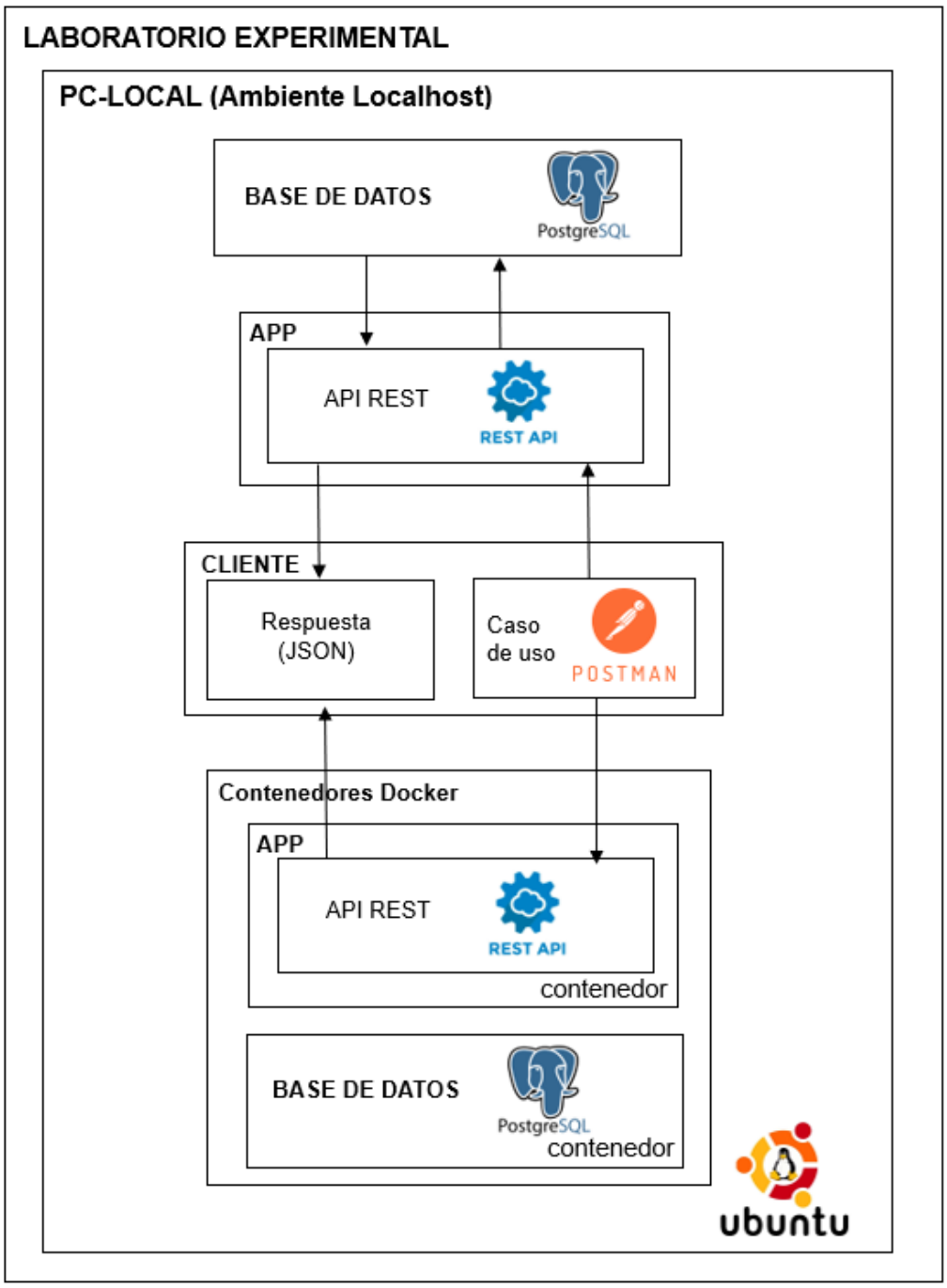


Figura 24: Arquitectura del laboratorio experimental.

3.1.7. Instrumentación

En esta sección se proporciona una descripción detallada de la instrumentación utilizada en el laboratorio computacional, que incluye la infraestructura, tecnologías y herramientas empleadas:

Descripción de PC-local:

- Sistema Operativo: Linux Ubuntu 22.04 64-bit

- Procesador: AMD Ryzen 5 2500U with Radeon Vega Mobile Gfx 2.00 GHz
- Memoria: RAM 16,0 GB

Ambiente de Desarrollo:

- Ambiente de desarrollo integrado IDE: Visual Studio Code v1.67.2
- Lenguaje de programación: Javascript
- Entorno de tiempo de ejecución de Javascript: NodeJS v12.22.9
- Librería npm para el API REST
- Controlador de base de datos (PostgreSQL)
- Aplicación cliente para el consumo del API REST: Postman v9.18.3

Recolección y análisis de datos: consta las siguientes aplicaciones.

- Microsoft Excel 365

3.1.8. Recolección de datos

Además, se describe el esquema de datos utilizado en el archivo Excel para el registro de los resultados, que se presenta en la Tabla 27.

Tabla 27: Estructura de recolección de datos

Campo	Descripción
Caso de uso	Tipo de caso de uso realizado.
Ambiente	Localhost o Docker.
Nivel	Nivel de consulta.
Repetición	Número de veces que se repitió la ejecución (de 1 a 3 veces).
Nro. Registros	Cantidad de registros consultados en el caso de uso.
Tiempo	Tiempo que tardó la ejecución del caso de uso, medido en milisegundos.

3.2. Ejecución del experimento

Los pasos diseñados en el entorno experimental, tal como se detallan en la sección 3.1, fueron seguidos para llevar a cabo la comparación de eficiencia entre el ambiente local y los contenedores Docker.

3.2.1. Explicación de la ejecución

Durante la ejecución del experimento, se llevaron a cabo tres repeticiones para cada cantidad de registros necesarios en cada caso de prueba, tanto en entornos locales como en contenedores Docker. Este enfoque se implementó con el fin de asegurar la obtención de mediciones precisas y representativas de los tiempos de respuesta.

En cada repetición, se ejecutó la consulta correspondiente de cada caso de uso con el número específico de registros y se registraron los tiempos de respuesta. Este proceso se repitió dos veces más, generando así tres conjuntos de tiempos de respuesta para cada cantidad de registros requeridos en el primer caso de prueba. Al realizar estas repeticiones, se generaron múltiples conjuntos de datos que facilitaron el cálculo de promedios. Las repeticiones se llevaron a cabo tanto en entornos locales como en el contenedor Docker, utilizando registros de 1, 10, 100, 1000 y 10000 registros.

3.2.2. Recolección de datos

Después de completar la ejecución de los casos de uso siguiendo el procedimiento establecido, se procedió a recopilar los resultados de los tiempos de respuesta y se ingresaron en una hoja de cálculo de Microsoft Excel 365 para su análisis. Los tiempos fueron registrados en milisegundos. A continuación se presenta la respuesta correspondiente al caso de uso 1.

Repeticiones para el caso de uso 1, ambiente localhost (1 registro):

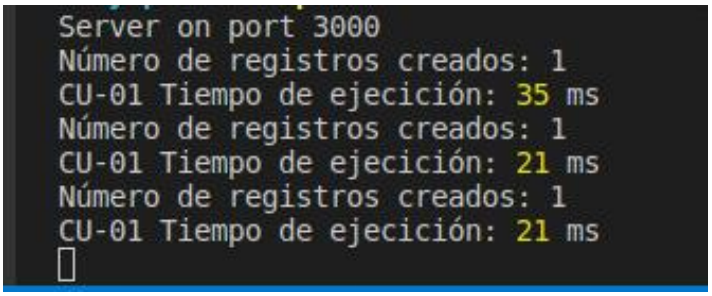


Figura 25: Ambiente localhost con tres repeticiones para el caso de uso 1 con 1 registro

Repeticiones para el caso de uso 1, Contenedor Docker (1 registro):

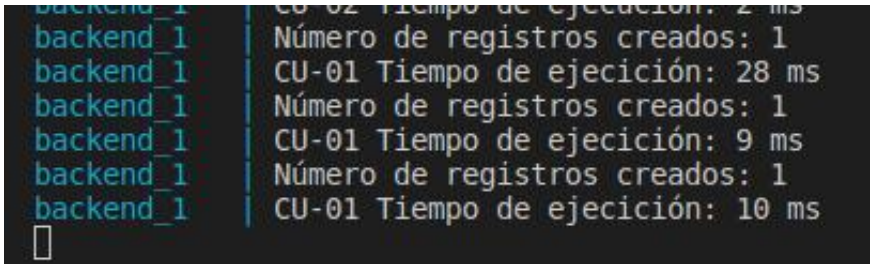


Figura 26: Contenedor Docker con tres repeticiones para el caso de uso 1 con 1 registro

Repeticiones para el caso de uso 1, ambiente localhost (10000 registros):

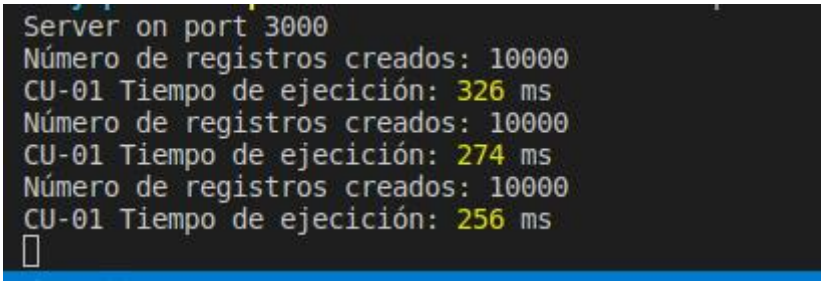


Figura 27: Ambiente localhost con tres repeticiones para el caso de uso 1 con 10000 registros

Repeticiones para el caso de uso 1, Contenedor Docker (10000 registros):

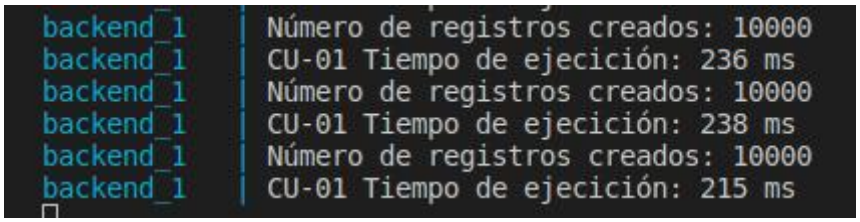


Figura 28: Contenedor Docker con tres repeticiones para el caso de uso 1 con 10000 registros

Luego de Obtener los datos de cada caso de uso con sus respectivos registros y repeticiones, se procedió a registrar los resultados en una hoja de Excel, tal y como se muestra en la Tabla.

Tabla 28: Registros recopilados del caso de uso 1 con 1 y 10000 registros

Nro.	Caso de Uso	Ambiente	Nivel	Repeticiones	Nro. Registros	Tiempo (ms)
1	CU-01	Localhost	1	1	1	35
2	CU-01	Localhost	1	2	1	21
3	CU-01	localhost	1	3	1	21
4	CU-01	Docker	1	1	1	28

5	CU-01	Docker	1	2	1	9
6	CU-01	Docker	1	3	1	10
7	CU-01	localhost	1	1	10000	326
8	CU-01	localhost	1	2	10000	274
9	CU-01	localhost	1	3	10000	256
10	CU-01	Docker	1	1	10000	236
11	CU-01	Docker	1	2	10000	238
12	CU-01	Docker	1	3	10000	215

3.1 Análisis de resultados

En esta parte del estudio se examinó el efecto de la base de datos PostgreSQL en el entorno Localhost y en los contenedores Docker, centrándose en la eficiencia del rendimiento utilizando la métrica del tiempo medio de respuesta conforme a la norma ISO/IEC 25023.

3.1.1. Análisis de la eficiencia del rendimiento

A continuación, se efectúa un análisis detallado sobre la eficiencia del rendimiento de cada caso de uso en los diversos entornos..

Caso de Uso 1

Para el primer caso de uso, "Insertar Usuarios", se compararon los tiempos promedio de respuesta entre Docker y Localhost. Los resultados muestran que, en Docker, el tiempo medio de respuesta fue de 62.6 segundos, mientras que en Localhost fue de 80 segundos. Esto representa una diferencia significativa de 17.4 segundos a favor de Docker. Al calcular el porcentaje de mejora, se observa que Docker fue un 21.75% más eficiente en comparación con Localhost para este caso de uso. Esto sugiere que Docker proporciona un entorno más eficiente para la inserción de usuarios en la base de datos PostgreSQL.

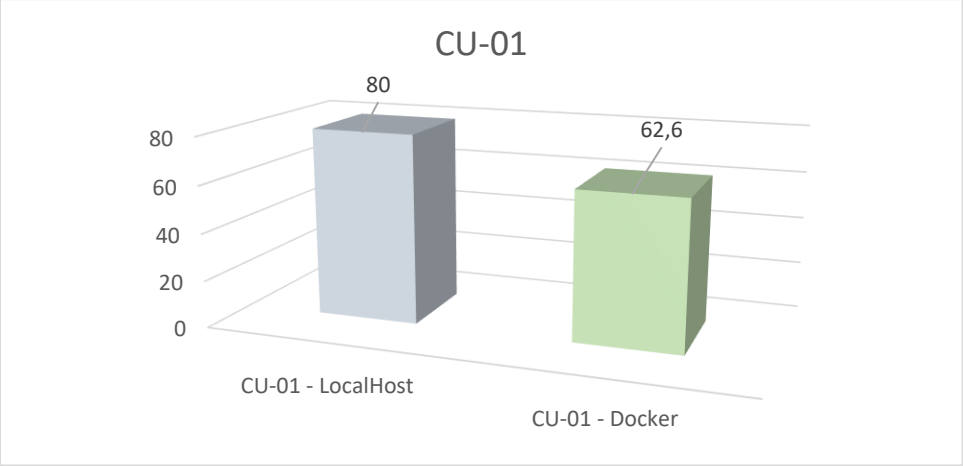


Figura 29: Valor medio de eficiencia caso de uso 1.

Caso de Uso 2

Para el segundo caso de uso, "Consultar Usuarios", se analizaron los tiempos promedio de respuesta entre Docker y Localhost. Se encontró que, en Docker, el tiempo medio de respuesta fue de 32.2 segundos, mientras que en Localhost fue de 42.13 segundos. Esto indica una diferencia de 9.93 segundos a favor de Docker. Calculando el porcentaje de mejora, Docker demostró ser un 23.56% más eficiente que Localhost en este caso. Estos resultados sugieren que Docker es más eficiente para la consulta de usuarios en la base de datos PostgreSQL.

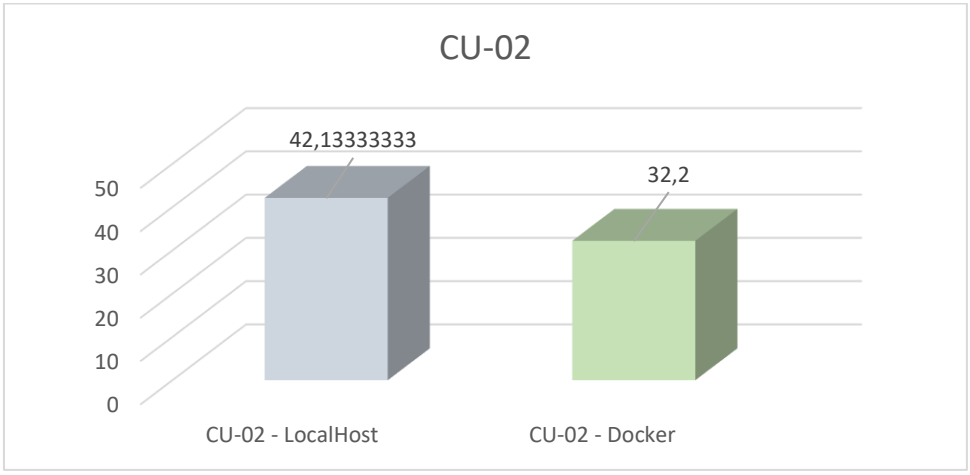


Figura 30: Valor medio de eficiencia caso de uso 2.

Caso de Uso 3

Para el tercer caso de uso, "Consultar Cuentas por Usuario", se analizaron los tiempos medios de respuesta entre Docker y Localhost. Se encontró que, en Docker, el tiempo medio de respuesta fue de 20.6 segundos, mientras que en Localhost fue de 29.8 segundos. Esto representa una diferencia de 9.2 segundos a favor de Docker. Al calcular el porcentaje de

mejora, se observa que Docker fue un 30.87% más eficiente en comparación con Localhost para este caso de uso. Esto indica que Docker proporciona un entorno más eficiente para consultar cuentas por usuario en la base de datos PostgreSQL.

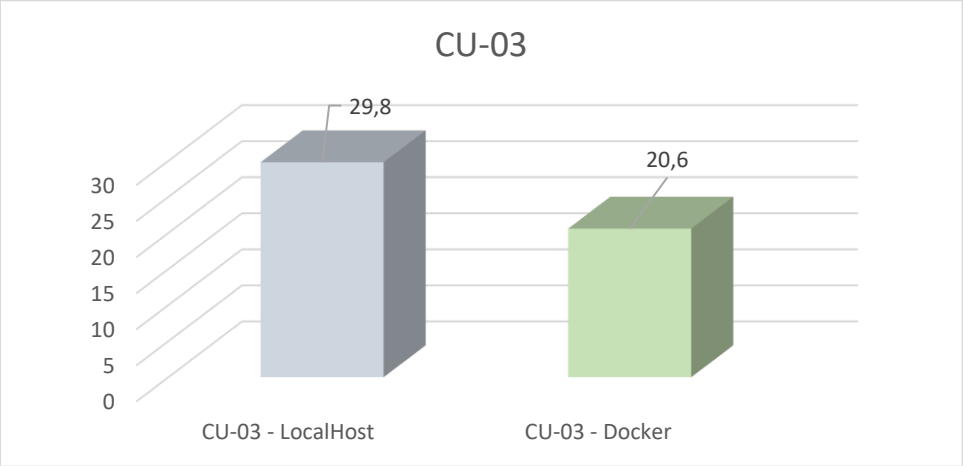


Figura 31: Valor medio de eficiencia caso de uso 3.

Caso de Uso 4

Para el cuarto caso de uso, "Consultar Movimientos de una Cuenta con Tipo de Movimiento", se analizaron los tiempos promedio de respuesta entre Docker y Localhost. Se encontró que, en Docker, el tiempo medio de respuesta fue de 27.6 segundos, mientras que en Localhost fue de 35.67 segundos. Esto indica una diferencia de 8.07 segundos a favor de Docker. Calculando el porcentaje de mejora, Docker demostró ser un 22.61% más eficiente que Localhost en este caso. Estos resultados sugieren que Docker es más eficiente para la consulta de movimientos de una cuenta con tipo de movimiento en la base de datos PostgreSQL.

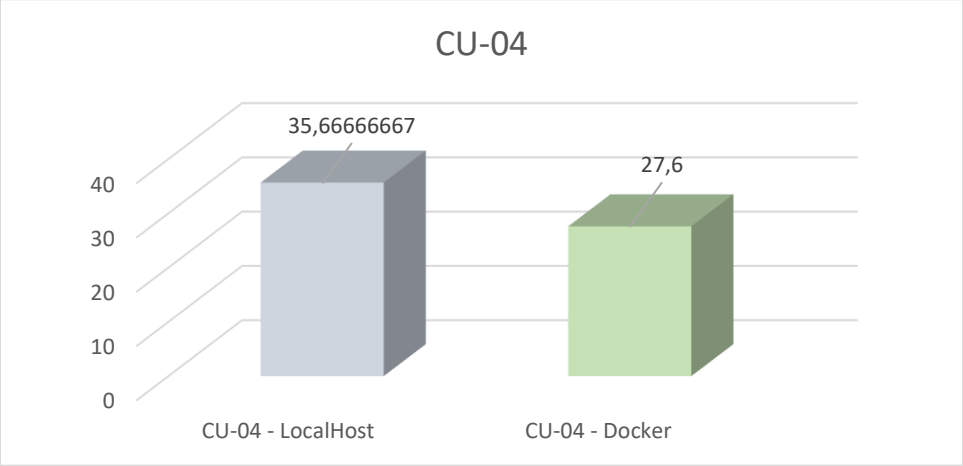


Figura 32: Valor medio de eficiencia caso de uso 4.

Caso de Uso 5

Para el quinto caso de uso, "Consultar Todos los Movimientos de un Usuario con Detalle de Cuenta y Tipo de Cuenta", se analizaron los tiempos medios de respuesta entre Docker y Localhost. Se encontró que, en Docker, el tiempo medio de respuesta fue de 33 segundos, mientras que en Localhost fue de 44.4 segundos. Esto representa una diferencia de 11.4 segundos a favor de Docker. Al calcular el porcentaje de mejora, se observa que Docker fue un 25.68% más eficiente en comparación con Localhost para este caso de uso. Esto sugiere que Docker proporciona un entorno más eficiente para consultar todos los movimientos de un usuario con detalle de cuenta y tipo de cuenta en la base de datos PostgreSQL.

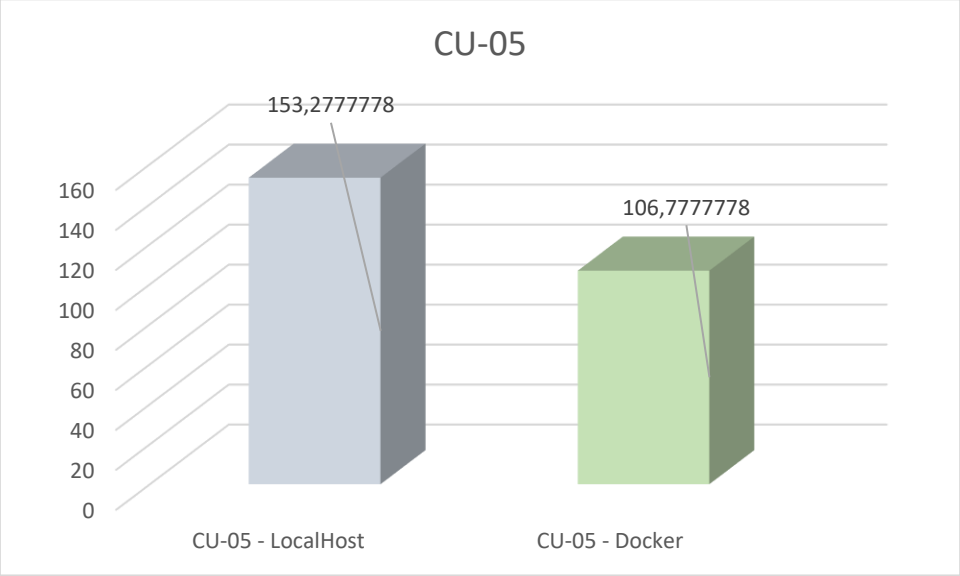


Figura 33: Valor medio de eficiencia caso de uso 5.

3.2 Análisis de impactos

Después de analizar los tiempos promedio de respuesta para cada caso de uso en los entornos Localhost y Docker, se han identificado importantes impactos en la eficiencia del despliegue de servicios. En general, se observa una notable mejora en el rendimiento al utilizar Docker en comparación con Localhost.

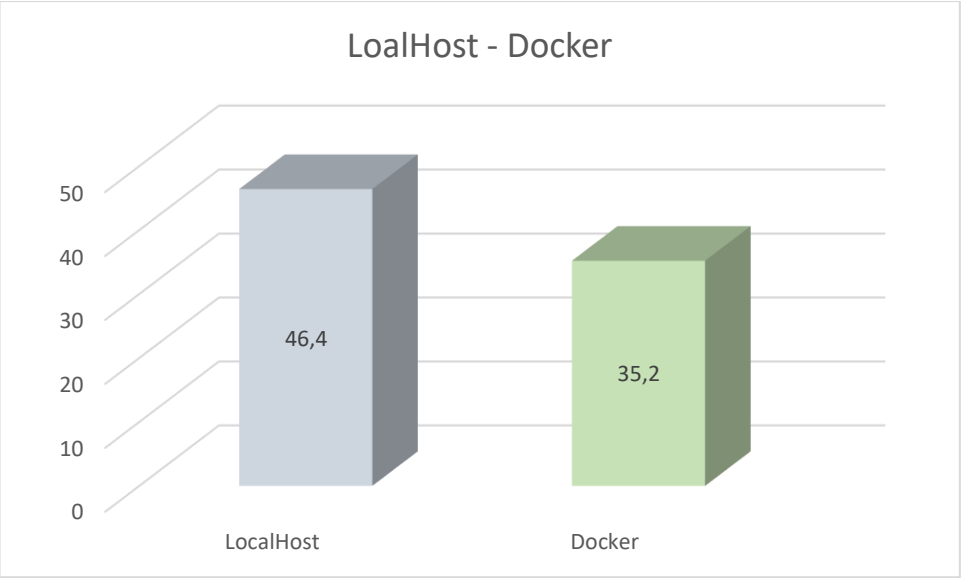


Figura 34: Comparativa de eficiencia en ambos ambientes.

Los tiempos promedio de respuesta para cada caso de uso en ambos entornos, se encontró que Docker demostró un rendimiento general más rápido en comparación con Localhost. En promedio, el tiempo de respuesta en Docker fue de aproximadamente 35.2 segundos, mientras que en Localhost fue de 46.4 segundos. Esto significa que Docker fue, en promedio, 11.2 segundos más rápido que Localhost. Expresado como un porcentaje de mejora, esto equivale a aproximadamente un 24.14% de eficiencia adicional en Docker en comparación con Localhost. Estos hallazgos sugieren que Docker ofrece una mejor eficiencia general en términos de tiempo de respuesta, lo que respalda su preferencia como entorno para el despliegue de los servicios analizados.

3.2.1. Resumen de análisis de impactos

En la Tabla 29 se muestra un resumen de los resultados detallados previamente para cada caso de uso, incluyendo sus impactos correspondientes.

Tabla 29: Resumen de análisis de impactos para cada caso de uso

Caso de uso	Localhost media (ms)	Porcentaje (%)	Docker media (ms)	Porcentaje (%)	Impacto	Diferencia(ms)
CU-01	80	100%	62.6	78.25%	21.75%	17.4
CU-02	42.13	100%	32.2	76.43%	23.56%	9.93

CU-03	29.8	100%	20.6	69.13%	30.87%	9.2
CU-04	35.67	100%	27.6	77.39%	22.61%	8.07
CU-05	44.4	100%	33	74.32%	25.68%	11.4

3.2.2. Análisis de resultados con estadística descriptiva

Distribución normal entorno Localhost

El análisis de los resultados de distribución y frecuencia proporciona información valiosa sobre cómo se distribuyen los datos en diferentes rangos y con qué frecuencia ocurren ciertos valores dentro de esos rangos. En el conjunto de datos proporcionado, se observa que la mayoría de los registros se concentran en los intervalos más bajos, con una mayor frecuencia en los intervalos de 20 y 40. Esto sugiere que la mayoría de los valores están agrupados en el extremo inferior de la escala de datos. Sin embargo, también se observa la presencia de algunos valores atípicos en los extremos superior e inferior, como se evidencia por la presencia de registros en los intervalos de 260, 280 y 300, así como en el intervalo de 0. Esto indica una cierta variabilidad en los datos y la posible presencia de valores extremos que pueden afectar la interpretación de las medidas estadísticas.

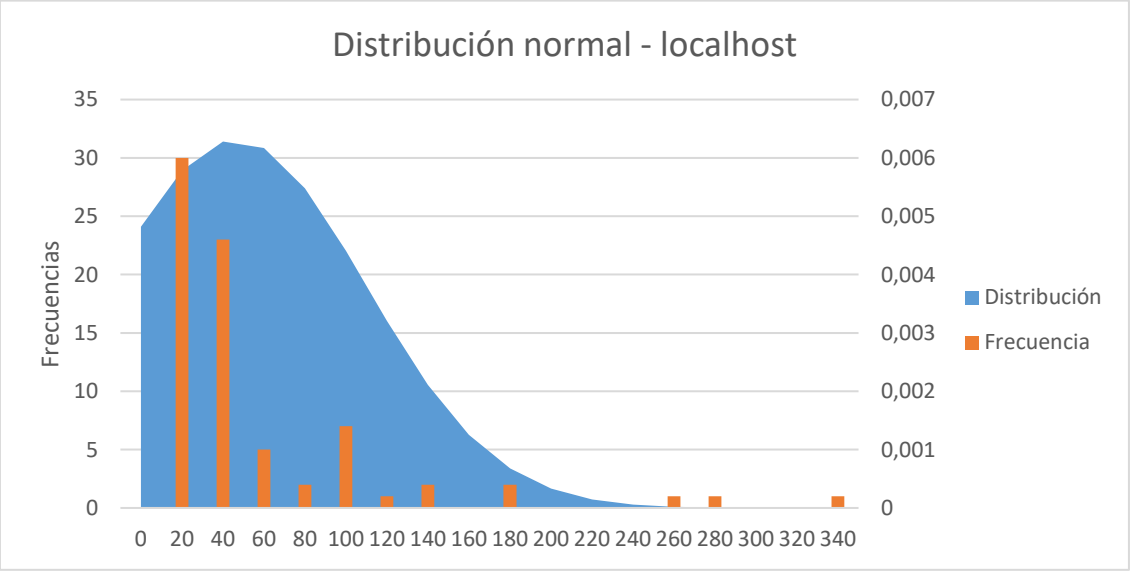


Figura 35: Distribución normal entorno Localhost.

Distribución normal contenedores Docker

El análisis de los resultados de distribución y frecuencia revela una distribución sesgada hacia los valores más bajos en el conjunto de datos proporcionado. La mayoría de

los registros se concentran en los intervalos inferiores, particularmente en los intervalos de 20 y 40, con frecuencias de 47 y 10 respectivamente. Esto sugiere que la mayoría de los valores se encuentran en el extremo inferior de la escala de datos, lo que podría indicar una tendencia hacia valores más bajos en la variable medida. Además, se observa que la frecuencia disminuye considerablemente a medida que aumentan los intervalos, con pocos o ningún registro en los intervalos superiores. Esto indica una distribución sesgada hacia la izquierda, donde la cola de la distribución se encuentra en el extremo inferior de la escala. Sin embargo, también se identifican algunos valores atípicos en los intervalos superiores, como se evidencia por la presencia de registros en los intervalos de 220, 240 y 260. En resumen, el análisis de distribución y frecuencia proporciona información sobre cómo se distribuyen los datos y destaca la presencia de valores atípicos que pueden influir en la interpretación de los resultados.

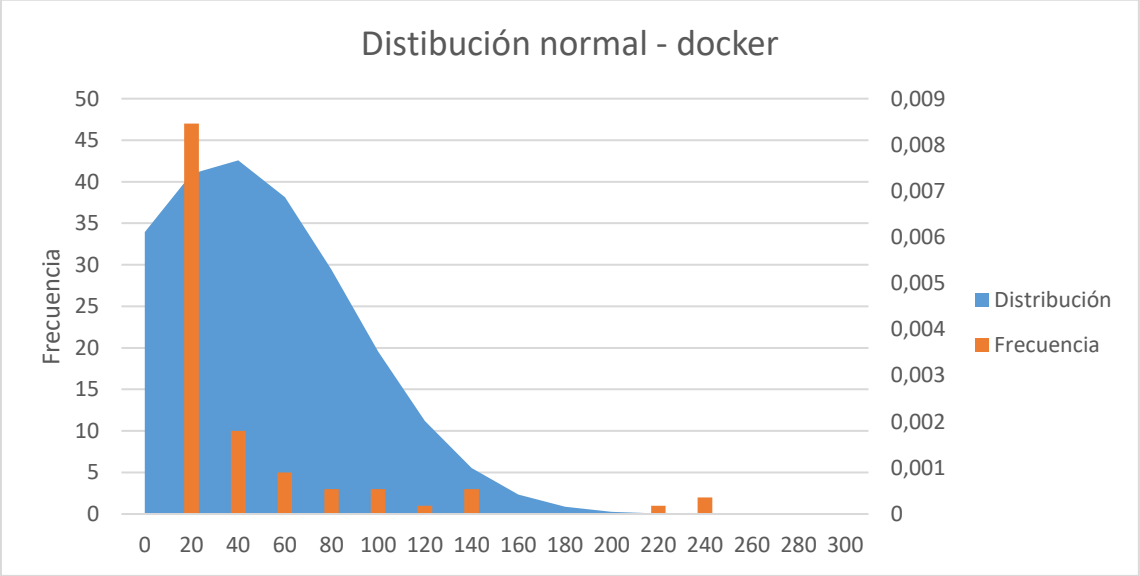


Figura 36: Distribución normal contenedor Docker.

Para los dos conjuntos de datos proporcionados, se observa una distribución sesgada hacia los valores más bajos, con la mayoría de los registros concentrados en los intervalos inferiores. En el primer conjunto de datos, la mayor frecuencia se encuentra en los intervalos de 20 y 40, con 47 y 10 registros respectivamente. Esto sugiere una tendencia hacia valores más bajos en la variable medida, con poca variabilidad en los intervalos superiores. En el segundo conjunto de datos, se presenta una distribución similar, con la mayoría de los registros también concentrados en los intervalos inferiores. Los intervalos de 20 y 40 tienen las frecuencias más altas, con 47 y 10 registros respectivamente. Sin embargo, en este conjunto de datos, la frecuencia disminuye más rápidamente a medida que aumentan los intervalos, lo que indica una distribución más sesgada hacia la izquierda. Además, se identifican algunos valores atípicos en los intervalos superiores, lo que sugiere una presencia

de variabilidad en los datos. En resumen, ambos conjuntos de datos exhiben una distribución sesgada hacia los valores más bajos, con poca variabilidad en los intervalos superiores, pero con la presencia de valores atípicos en el segundo conjunto de datos.

CONCLUSIONES

- La construcción de la base teórica mediante una revisión bibliográfica y conceptualización de ambientes locales y contenedores Docker, facilitó el aprendizaje de esta nueva tecnología. De igual manera se determinó que, REST es la arquitectura factible para el desarrollo de servicios, junto con la base de datos PostgreSQL que está dentro de la lista de tecnologías más comunes que se ejecutan en Docker, llevando el tercer puesto siendo la base de datos relacional de código abierto que ha ido subiendo de rango cada año.
- La implementación del marco de trabajo Scrum en el desarrollo de la instrumentación tecnológica del experimento de software de este proyecto proporcionó una estructura sólida y flexible para la gestión de los equipos y la entrega de incrementos de producto. A lo largo del proceso, se observó cómo Scrum fomenta la colaboración, la transparencia y la adaptabilidad, permitiendo al equipo responder de manera ágil a los cambios en los requisitos y las prioridades del proyecto. Además, la definición clara de roles, eventos y artefactos en Scrum facilitó la comunicación y la coordinación entre los miembros del equipo, lo que contribuyó a un ambiente de trabajo productivo y enfocado en los objetivos del sprint. En resumen, la adopción de Scrum demostró ser una herramienta efectiva para la entrega incremental de valor, promoviendo la calidad del producto y la satisfacción del cliente.
- El diseño experimental propuesto en el capítulo 3 para comparar la eficiencia del rendimiento de servicios API REST en ambiente local contra servicios API REST en contenedores Docker en términos de la calidad de software, se contesta la siguiente pregunta de investigación $\boldsymbol{PI}$: ¿Qué ambiente es más eficiente para la ejecución de bases de datos PostgreSQL en arquitecturas DevOps? Para ello, se implementó 5 casos de uso en los diferentes ambientes y se midió la eficiencia del rendimiento utilizando la métrica del tiempo medio de respuesta de la norma ISO/IEC 25023 de la característica "*Medidas de eficiencia del desempeño*".
- Basándonos en esta caracterización, hemos podido responder a la pregunta derivada de la investigación: $\boldsymbol{PI}(\mathbf{1})$ ¿Cuál de los entornos, ambiente local o contenedores Docker, es más eficiente para la ejecución de base de datos PostgreSQL? Tras realizar las tareas experimentales y analizar los resultados, se ha confirmado que el tiempo medio de respuesta en la arquitectura Docker es menor que en el entorno Localhost.
- El análisis comparativo entre los tiempos de respuesta en los entornos Localhost y Docker revela consistentemente un mejor rendimiento en Docker en todos los casos de uso evaluados. Los tiempos de respuesta promedio fueron notablemente inferiores en Docker en comparación con Localhost, mostrando mejoras significativas que

oscilan entre el 21.75% y el 30.87%. Estos hallazgos sugieren que Docker proporciona un entorno más eficiente para el despliegue de aplicaciones que requieren tiempos de respuesta rápidos, lo que podría tener un impacto positivo en la experiencia del usuario y la eficiencia general del sistema.

RECOMENDACIONES

- Hacer uso de la documentación establecida en Docker para conocer sus complementos, librerías y otras dependencias requeridas para implementar Docker, con el fin de evitar el mal funcionamiento y posibles conflictos en la comunicación de contenedores.
- Para obtener resultados precisos, se recomienda desplegar la aplicación en un entorno de producción, donde esté empaquetado todo el proyecto. Esto permitirá medir con precisión los tiempos de ejecución de los servicios. Es importante tener en cuenta que los tiempos de ejecución pueden variar según los recursos disponibles en el ordenador utilizado.
- Además, es importante tener en cuenta que los resultados pueden variar en un entorno de desarrollo más complejo y con una mayor cantidad de registros. El rendimiento y los tiempos de ejecución pueden estar influenciados por la complejidad de la aplicación y la cantidad de datos procesados. Por lo tanto, se recomienda realizar pruebas exhaustivas en diferentes escenarios y condiciones para evaluar adecuadamente el rendimiento de la aplicación en situaciones más realistas.

BIBLIOGRAFÍA

Alberca, J. (2023). *Beneficios de aplicar DevOps en una organización ~ José Alberca*. http://www.jalberca.com/2023/01/beneficios-de-aplicar-devops-en-una.html

Arjona Quijano, C. (2019). *Docker: Creando entorno de desarrollo*.

Azure. (2019). *¿Qué es DevOps? Explicación de DevOps | Microsoft Azure*. Azure. https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-devops/#tools

Baxendale, P., & Codd, E. F. (1970). *Information Retrieval A Relational Model of Data Large Shared Data Banks*.

Benítez, M. (2015). *Curso de Introducción a la Administración de Bases de Datos - Miguel Ángel Benítez, Ángel Arias - Google Libros*.

Buchanan, I. (2019). *Comparación de contenedores y máquinas virtuales*.

Castillo, J. D. S., Martínez, A., Quesada-López, C., & Jenkins, M. (2020). Caracterización de las prácticas de DevOps en organizaciones que desarrollan software: Un mapeo sistemático de literatura. *Revista Ibérica de Sistemas e Tecnologias de Informação*, *E28*, 83–96. https://www.proquest.com/scholarly-journals/caracterización-de-las-prácticas-devops-en/docview/2388305192/se-2?accountid=10218

Cuesta, B., & González, S. (2016). *Introducción a docker*.

CUSTOM. (2019). *Diferencias entre contenedores de servicios y máquinas virtuales*.

De La Torre, C. (2022). *Containerized Docker Application Lifecycle with Microsoft Platform and Tools*.

De Toro, Á. (2022). *¿Qué es Scrum? Conoce el Framework que agiliza el Trabajo en Equipo*. https://www.escueladenegociosydireccion.com/revista/business/scrum-framework-agiliza-trabajo-equipo/

Departamento de ciencias de la computación e I.A. (2013). Introducción a las bases de datos. *Universidad de Granada*, 1–36.

Domínguez, J. (2020). *Cliente PSQL de Postrgres*. http://creativecommonsvenezuela.org.ve

Funes, F. (2018). *Análisis de solución eficiente para escalabilidad de sistemas online mediante orquestación de contenedores*.

García, F. (2021). *SLR, mappings y meta-análisis*. Zenodo. https://zenodo.org/records/4700155#.YbxKj2hByMo

Helma, S. (2010, February). *Programación de Bases de Datos con MYSQL y PHP - Helma Spona - Google Libros*.

IBM. (2018). *Consultas de lenguaje - Documentación de IBM*.

Jedlitschka, A., Ciolkowski, M., & Pfahl, D. (2008). Reporting experiments in software engineering. *Guide to Advanced Empirical Software Engineering*, 201–228.

https://doi.org/10.1007/978-1-84800-044-5_8

Lucateli Bernardi, T., Lazaretti Zanatta, A., Giacomelli Beux, J., Biduski, D., & Andrei Bellei, É. (2017). Learning by Doing em Fábrica de Software: Relato de uma Experiência no Mestrado Profissional em Computação Aplicada. *Anais Da Escola Regional de Engenharia de Software (ERES)*, 203–212. https://sol.sbc.org.br/index.php/eres/article/view/10098

Luz, J. (2021). *Trabajo final de grado GRADO EN INGENIER ´ IA INFORM ATICA Facultad de Matem ´ aticas e Inform ´ atica Universidad de Barcelona on de datos desde una API p ´ Caso de uso - consumo y visualizaci ´ COVID-19 Director :*

Mamani Rodríguez, Z. E., Del Pino Rodríguez, L., & Gonzales Suarez, J. C. (2020). Arquitectura basada en Microservicios y DevOps para una ingeniería de software continua. *Industrial Data*, *23*(2), 141–149. https://doi.org/10.15381/idata.v23i2.17278

Nakai, H., Tsuda, N., Honda, K., Washizaki, H., & Fukazawa, Y. (2016). Initial Framework for Software Quality Evaluation Based on ISO/IEC 25022 and ISO/IEC 25023. *Proceedings - 2016 IEEE International Conference on Software Quality, Reliability and Security-Companion, QRS-C 2016*, 410–411. https://doi.org/10.1109/QRS-C.2016.66

Narvaez, J. (2014). *GUÍA DE LAS MEJORES PRÁCTICAS ADMINISTRATIVAS, SEGURIDAD Y ALTA DISPONIBILIDAD, CASO DE ESTUDIO: POSTGRESQL*.

Objetivos de Desarrollo Sostenible | Programa De Las Naciones Unidas Para El Desarrollo. (n.d.). Retrieved March 4, 2024, from https://www.undp.org/es/sustainable-development-goals

Osorio Rivera, F. L. (2008). Base de datos relacionales. Teoría y práctica. *Https://Catalogo.Itm.Edu.Co/Gpd-Bases-de-Datos-Relacionales.Html*, 245.

Pacheco, J. (2018). *ESTUDIO COMPARATIVO ENTRE UNA ARQUITECTURA CON MICROSERVICIOS Y CONTENEDORES DOCKERS Y UNA ARQUITECTURA TRADICIONAL (MONOLÍTICA) CON COMPROBACIÓN APLICATIVA*.

Pacheco Laje, J. L. (2018). *ESTUDIO COMPARATIVO ENTRE UNA ARQUITECTURA CON MICROSERVICIOS Y CONTENEDORES DOCKERS Y UNA ARQUITECTURA TRADICIONAL (MONOLÍTICA) CON COMPROBACIÓN APLICATIVA*.

PIÑEIRO GOMEZ, J. M. (2013). *Bases de datos relacionales y modelado de datos*.

Riti, P. (2018). Introduction to DevOps. *Pro DevOps with Google Cloud Platform*, 1–18. https://doi.org/10.1007/978-1-4842-3897-4_1

Rodríguez, T. (2019). *De Docker a Kubernetes: entendiendo qué son los contenedores y por qué es una de las mayores revoluciones de la industria del desarrollo*. Xataka. https://www.xataka.com/otros/docker-a-kubernetes-entendiendo-que-contenedores-que-mayores-revoluciones-industria-desarrollo

Sánchez, J. (2004). *Principios sobre Bases de Datos Relacionales*.

Silva, J. A. (2021). *Análisis de contenedores Docker y sus implicaciones de seguridad*.

Simbaña Alarcón, M. A. (2016). *Estudio del contenedor Cloud Docker y propuesta de implementación para la plataforma Cloud FICA*.

Takeuchi, H., & Nonaka, I. (1986). *The New New Product Development Game*. https://hbr.org/1986/01/the-new-new-product-development-game

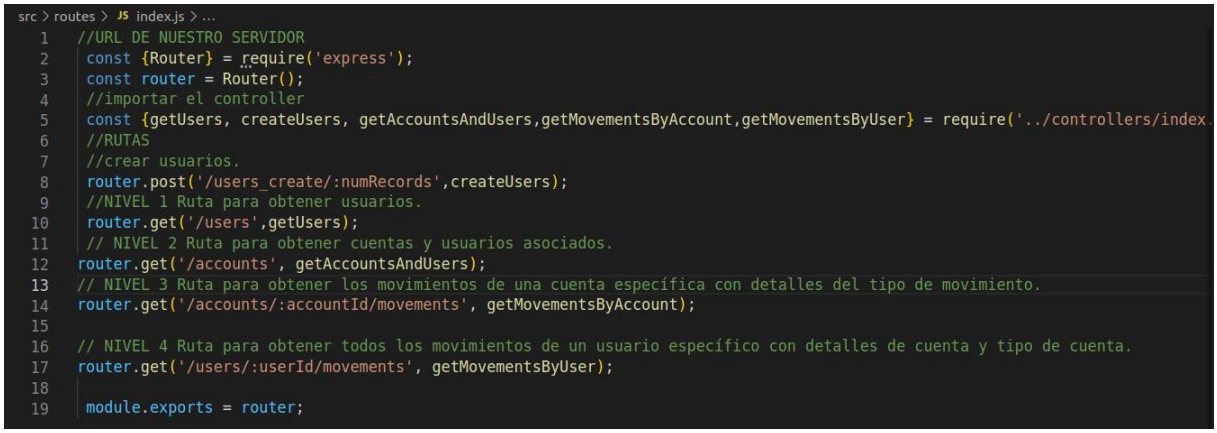
TIC. (2019, July). *Base de datos*.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., & Wesslén, A. (2012). Experimentation in Software Engineering. In *IEEE Transactions on Software Engineering: Vol. SE-12* (Issue 7). https://doi.org/10.1109/TSE.1986.6312975

Zeron, L. (2019). *Consecuencias de no actualizar su tecnología IT - Data System*. Data Blog. https://www.reparacionordenadoresmadrid.org/consecuencias-de-no-actualizar-su-tecnologia-it.html

ANEXOS

ANEXO A: Estructura de URL de los servicios para las 5 peticiones.



```
src > routes > JS index.js > ...
1   //URL DE NUESTRO SERVIDOR
2   const {Router} = require('express');
3   const router = Router();
4   //importar el controller
5   const {getUsers, createUsers, getAccountsAndUsers,getMovementsByAccount,getMovementsByUser} = require('../controllers/index.
6   //RUTAS
7   //crear usuarios.
8   router.post('/users_create/:numRecords',createUsers);
9   //NIVEL 1 Ruta para obtener usuarios.
10  router.get('/users',getUsers);
11  // NIVEL 2 Ruta para obtener cuentas y usuarios asociados.
12  router.get('/accounts', getAccountsAndUsers);
13  // NIVEL 3 Ruta para obtener los movimientos de una cuenta específica con detalles del tipo de movimiento.
14  router.get('/accounts/:accountId/movements', getMovementsByAccount);
15
16  // NIVEL 4 Ruta para obtener todos los movimientos de un usuario específico con detalles de cuenta y tipo de cuenta.
17  router.get('/users/:userId/movements', getMovementsByUser);
18
19  module.exports = router;
```

ANEXO B: Estructura de Controller para la inserción de usuarios.

```
// Crear usuarios de forma automática
const createUsers = async (req, res) => {
    try {
        const { numRecords } = req.params; // Obtener el número de registros desde los parámetros de la URL
        const insertQuery = 'INSERT INTO users(first_name, last_name, identification, email, phone, address) VALUES ';
        const values = [];

        const startTime = new Date(); // Registro de la hora de inicio

        for (let i = 0; i < numRecords; i++) {
            const identification = `ID-${i + 1}`.substring(0, 15); // Limitar la longitud de la identificación a 15 caracteres
            values.push(`('First Name ${i + 1}', 'Last Name ${i + 1}', '${identification}', 'email${i + 1}@example.com', '123456789${i + 1}', 'Address ${i
        }
        const query = insertQuery + values.join(', ');

        const response = await pool.query(query);

        const endTime = new Date(); // Registro de la hora de finalización
        const elapsedTime = endTime - startTime; // Calcula el tiempo de ejecución en milisegundos

        console.log('Número de registros creados:', numRecords);
        console.log('CU-01 Tiempo de ejecición:', elapsedTime, 'ms'); // Imprimir por consola el número de registros y el tiempo de ejecución

        res.send(`${numRecords} usuarios creados`);
    } catch (error) {
        console.error('Error al crear usuarios automáticamente:', error);
        res.status(500).json({ message: 'Error interno del servidor' });
    }
};
```

ANEXO C: Función controller para obtener la lista usuarios.

```
// Ver usuarios
const getUsers = async (req, res) => {
    try {
        const startTime = new Date(); // Registro de la hora de inicio

        const response = await pool.query('SELECT * FROM users');
        const userData = response.rows;
        const totalRecords = userData.length;

        const endTime = new Date(); // Registro de la hora de finalización
        const elapsedTime = endTime - startTime; // Calcula el tiempo transcurrido en milisegundos

        // Imprimir el número total de registros en la consola del servidor
        console.log('Número total de registros:', totalRecords);
        console.log('CU-02 Tiempo de ejecución:', elapsedTime, 'ms'); // Imprime el tiempo transcurrido en la consola

        res.status(200).json({ totalRecords, userData });
    } catch (error) {
        console.error('Error al obtener usuarios:', error);
        res.status(500).json({ message: 'Error interno del servidor' });
    }
}
```

ANEXO D: Función controller para obtener cuentas y usuarios asociados.

```
// Función para obtener cuentas y usuarios asociados (NIVEL 2)
const getAccountsAndUsers = async (req, res) => {
    try {
        const startTime = new Date(); // Registro de la hora de inicio
        const response = await pool.query('SELECT accounts.id_account, accounts.account_number, accounts.balance, users.first_name, users.last_name FROM ac
        const userData = response.rows;
        const totalRecords = userData.length;

        const endTime = new Date(); // Registro de la hora de finalización
        const elapsedTime = endTime - startTime; // Calcula el tiempo transcurrido en milisegundos

        console.log('Número total de registros:', totalRecords);
        console.log('CU-03 Tiempo de ejecución:', elapsedTime, 'ms'); // Imprime el tiempo transcurrido en la consola

        res.status(200).json({ totalRecords, userData });
    } catch (error) {
        console.error('Error al obtener cuentas y usuarios:', error);
        res.status(500).json({ message: 'Error interno del servidor' });
    }
};
```

ANEXO E: Función controller para obtener los movimientos de una cuenta específica con detalles del tipo de movimiento.

```
// Función para obtener los movimientos de una cuenta especifica con detalles del tipo de movimiento (NIVEL 3)
const getMovementsByAccount = async (req, res) => {
    try {
        const startTime = new Date(); // Registro de la hora de inicio
        const { accountId } = req.params; // Obtén el ID de cuenta de los parámetros de la URL
        const response = await pool.query('SELECT movements.id_movement, movements.amount, movements.date_register, types_movement.name AS type_movement, a
        const userData = response.rows;
        const totalRecords = userData.length;

        const endTime = new Date(); // Registro de la hora de finalización
        const elapsedTime = endTime - startTime; // Calcula el tiempo transcurrido en milisegundos

        console.log('Número total de registros:', totalRecords);
        console.log('CU-04 Tiempo de ejecución:', elapsedTime, 'ms'); // Imprime el tiempo transcurrido en la consola

        res.status(200).json({ totalRecords, userData });
    } catch (error) {
        console.error('Error al obtener movimientos de cuenta:', error);
        res.status(500).json({ message: 'Error interno del servidor' });
    }
};
```

ANEXO F: Función controller para obtener todos los movimientos de un usuario específico con detalles de cuenta y tipo de cuenta.

```
// Función para obtener todos los movimientos de un usuario específico con detalles de cuenta y tipo de cuenta (NIVEL 4)
const getMovementsByUser = async (req, res) => {
    try {
        const startTime = new Date(); // Registro de la hora de inicio
        const { userId } = req.params; // Obtén el ID de usuario de los parámetros de la URL
        const response = await pool.query('SELECT movements.id_movement, movements.amount, movements.date_register, accounts.id_account, accounts.account_n
        const userData = response.rows;
        const totalRecords = userData.length;

        const endTime = new Date(); // Registro de la hora de finalización
        const elapsedTime = endTime - startTime; // Calcula el tiempo transcurrido en milisegundos

        console.log('Número total de registros:', totalRecords);
        console.log('CU-05 Tiempo de ejecución:', elapsedTime, 'ms'); // Imprime el tiempo transcurrido en la consola

        res.status(200).json({ totalRecords, userData });
    } catch (error) {
        console.error('Error al obtener movimientos por usuario:', error);
        res.status(500).json({ message: 'Error interno del servidor' });
    }
};
```

ANEXO G: Conexión a la base de dato en Localhost.

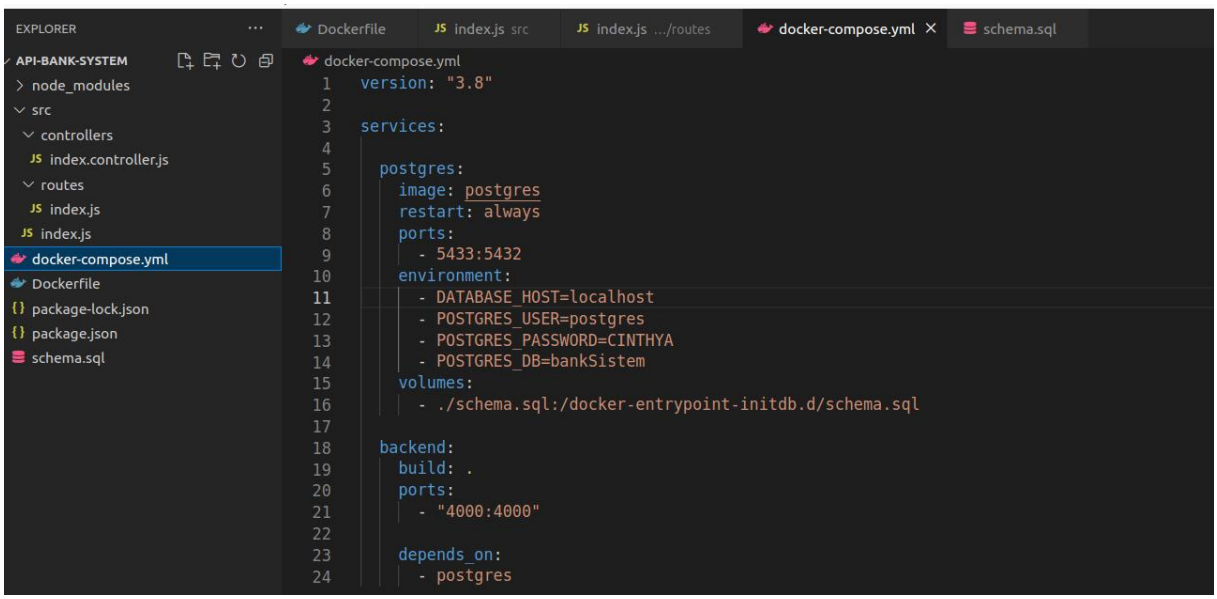
```
const pool = new Pool({
    host: 'localhost',
    user: 'postgres',
    password: 'CINTHYA',
    database: 'bankSystem',
    port: '5432'
});
```

ANEXO H: Conexión a la base de dato en contenedor Docker.

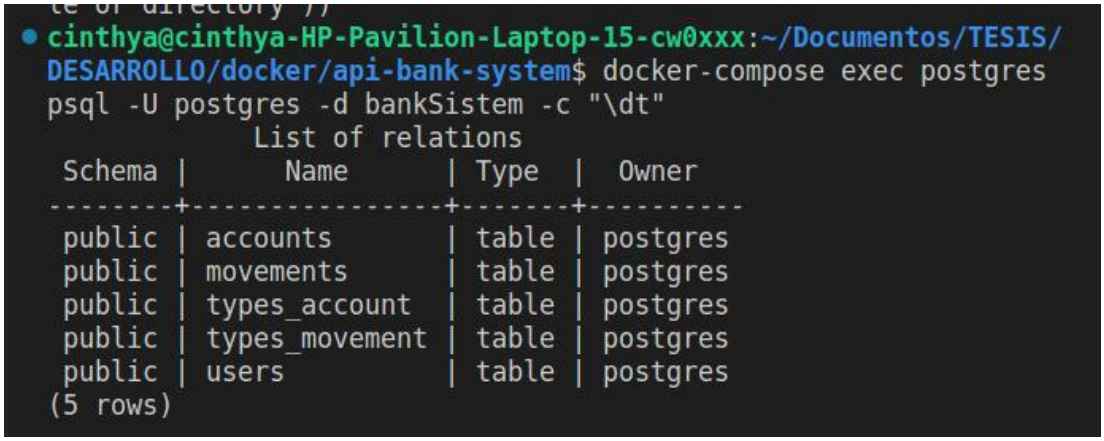
```
ckerfile | JS index.js src | JS index.js .../routes | docker-compose.yml | JS index.controller.js × | schem
controllers > JS index.controller.js > pool
const { Pool } = require('pg');

const pool = new Pool({
    host: 'postgres',        // Nombre del servicio de la base de datos en Docker Compose
    user: 'postgres',        // Usuario de la base de datos
    password: 'CINTHYA'      // Contraseña de la base de datos
ntos/TESIS/DESARROLLO/docker/api-bank-system/src/routes
    port: '5432'             // Puerto expuesto del contenedor
});
```

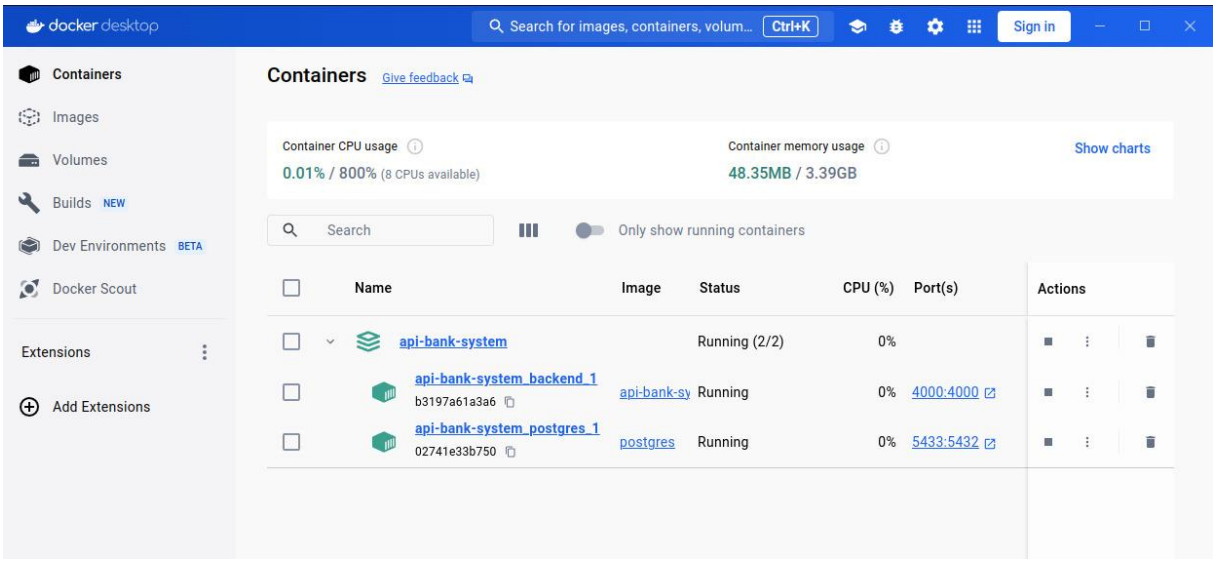
ANEXO I: Docker-compose en proyecto.



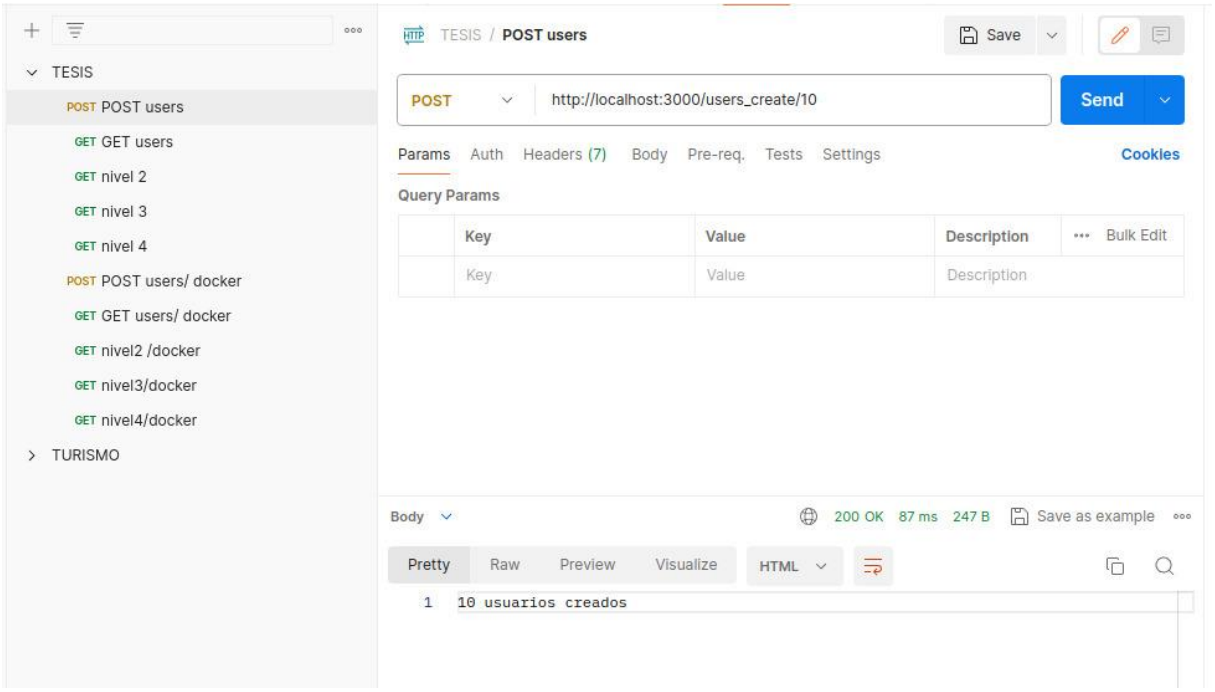
ANEXO J: Ingreso a la Base de datos dockerizada.



ANEXO K: Docker desktop de proyecto dockerizado.



ANEXO L: Estructura de peticiones para cada caso de uso.



ANEXO M: Modelo de ejecución y respuesta al realizar peticiones en ambiente Localhost.

```
cinthya@cinthya-HP-Pavilion-Laptop-15-cw0xxx:~/Documentos/TESIS/DESARROLLO/api-bank-system$ npm run dev

> api-bank-system@1.0.0 dev
> nodemon src/index.js

[nodemon] 3.0.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node src/index.js`
body-parser deprecated undefined extended: provide extended option src/index.js:6:17
Server on port 3000
Número de registros creados: 10
CU-01 Tiempo de ejecición: 56 ms
Número de registros creados: 10
CU-01 Tiempo de ejecición: 20 ms
Número de registros creados: 10
CU-01 Tiempo de ejecición: 8 ms
```

ANEXO N: Modelo de ejecución y respuesta al realizar peticiones en contenedor Docker.

```
cinthya@cinthya-HP-Pavilion-Laptop-15-cw0xxx:~/Documentos/TESIS/DESARROLLO/docker/api-bank-system$ docker-compose up
Starting api-bank-system_postgres_1 ... done
Starting api-bank-system_backend_1  ... done
Attaching to api-bank-system_postgres_1, api-bank-system_backend_1
postgres_1  |
postgres_1  | PostgreSQL Database directory appears to contain a database; Skipping initialization
postgres_1  |
postgres_1  | 2024-02-29 01:50:39.999 UTC [1] LOG:  starting PostgreSQL 16.1 (Debian 16.1-1.pgdg120+1) on x86_64-pc-linux-gnu, compil
(Debian 12.2.0-14) 12.2.0, 64-bit
postgres_1  | 2024-02-29 01:50:39.999 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
postgres_1  | 2024-02-29 01:50:39.999 UTC [1] LOG:  listening on IPv6 address "::", port 5432
postgres_1  | 2024-02-29 01:50:40.009 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
postgres_1  | 2024-02-29 01:50:40.021 UTC [29] LOG:  database system was shut down at 2024-02-29 01:50:35 UTC
postgres_1  | 2024-02-29 01:50:40.030 UTC [1] LOG:  database system is ready to accept connections
backend_1   |
backend_1   | > api-bank-system@1.0.0 start /app
backend_1   | > node src/index.js
backend_1   |
backend_1   | Thu, 29 Feb 2024 01:50:40 GMT body-parser deprecated undefined extended: provide extended option at src/index.js:6:17
backend_1   | Server on port 4000
backend_1   | Número total de registros: 100000
backend_1   | CU-05 Tiempo de ejecición: 44 ms
backend_1   | Número total de registros: 100000
backend_1   | CU-05 Tiempo de ejecición: 9 ms
backend_1   | Número total de registros: 100000
backend_1   | CU-05 Tiempo de ejecición: 8 ms
```

ANEXO O: Registro de recopilación de datos del Caso de Uso 1

Nro.	Caso de Uso	Ambiente	Nivel	Repeticiones	Nro. Registros	Tiempo (ms)
1	CU-01	Localhost	1	1	1	35
2	CU-01	Localhost	1	2	1	21
3	CU-01	Localhost	1	3	1	21
4	CU-01	Localhost	1	1	10	30
5	CU-01	Localhost	1	2	10	8
6	CU-01	Localhost	1	3	10	8
7	CU-01	Localhost	1	1	100	37
8	CU-01	Localhost	1	2	100	11

9	CU-01	Localhost	1	3	100	11
10	CU-01	Localhost	1	1	1000	59
11	CU-01	Localhost	1	2	1000	51
12	CU-01	Localhost	1	3	1000	52
13	CU-01	Localhost	1	1	10000	326
14	CU-01	Localhost	1	2	10000	274
15	CU-01	Localhost	1	3	10000	256
16	CU-01	Docker	1	1	1	28
17	CU-01	Docker	1	2	1	9
18	CU-01	Docker	1	3	1	10
19	CU-01	Docker	1	1	10	24
20	CU-01	Docker	1	2	10	8
21	CU-01	Docker	1	3	10	9
22	CU-01	Docker	1	1	100	13
23	CU-01	Docker	1	2	100	12
24	CU-01	Docker	1	3	100	14
25	CU-01	Docker	1	1	1000	38
26	CU-01	Docker	1	2	1000	39
27	CU-01	Docker	1	3	1000	46
28	CU-01	Docker	1	1	10000	236
29	CU-01	Docker	1	2	10000	238
30	CU-01	Docker	1	3	10000	215

ANEXO P: Registro de recopilación de datos del Caso de Uso 2

Nro.	Caso de Uso	Ambiente	Nivel	Repeticiones	Nro. Registros	Tiempo (ms)
1	CU-02	Localhost	1	1	1	32
2	CU-02	Localhost	1	2	1	4

3	CU-02	Localhost	1	3	1	1
4	CU-02	Localhost	1	1	10	28
5	CU-02	Localhost	1	2	10	2
6	CU-02	Localhost	1	3	10	1
7	CU-02	Localhost	1	1	100	36
8	CU-02	Localhost	1	2	100	7
9	CU-02	Localhost	1	3	100	8
10	CU-02	Localhost	1	1	1000	74
11	CU-02	Localhost	1	2	1000	26
12	CU-02	Localhost	1	3	1000	13
13	CU-02	Localhost	1	1	10000	169
14	CU-02	Localhost	1	2	10000	140
15	CU-02	Localhost	1	3	10000	91
16	CU-02	Docker	1	1	1	18
17	CU-02	Docker	1	2	1	2
18	CU-02	Docker	1	3	1	2
19	CU-02	Docker	1	1	10	2
20	CU-02	Docker	1	2	10	2
21	CU-02	Docker	1	3	10	3
22	CU-02	Docker	1	1	100	4
23	CU-02	Docker	1	2	100	21
24	CU-02	Docker	1	3	100	10
25	CU-02	Docker	1	1	1000	47
26	CU-02	Docker	1	2	1000	16
27	CU-02	Docker	1	3	1000	40
28	CU-02	Docker	1	1	10000	128
29	CU-02	Docker	1	2	10000	93
30	CU-02	Docker	1	3	10000	95

ANEXO Q: Registro de recopilación de datos del Caso de Uso 3

Nro.	Caso de Uso	Ambiente	Nivel	Repeticiones	Nro. Registros	Tiempo (ms)
1	CU-03	Localhost	2	1	1	35
2	CU-03	Localhost	2	2	1	2
3	CU-03	Localhost	2	3	1	1
4	CU-03	Localhost	2	1	10	33
5	CU-03	Localhost	2	2	10	4
6	CU-03	Localhost	2	3	10	2
7	CU-03	Localhost	2	1	100	36
8	CU-03	Localhost	2	2	100	5
9	CU-03	Localhost	2	3	100	3
10	CU-03	Localhost	2	1	1000	46
11	CU-03	Localhost	2	2	1000	25
12	CU-03	Localhost	2	3	1000	23
13	CU-03	Localhost	2	1	10000	97
14	CU-03	Localhost	2	2	10000	88
15	CU-03	Localhost	2	3	10000	47
16	CU-03	Docker	2	1	1	14
17	CU-03	Docker	2	2	1	1
18	CU-03	Docker	2	3	1	1
19	CU-03	Docker	2	1	10	11
20	CU-03	Docker	2	2	10	18
21	CU-03	Docker	2	3	10	1
22	CU-03	Docker	2	1	100	23
23	CU-03	Docker	2	2	100	3
24	CU-03	Docker	2	3	100	5
25	CU-03	Docker	2	1	1000	34

26	CU-03	Docker	2	2	1000	14
27	CU-03	Docker	2	3	1000	16
28	CU-03	Docker	2	1	10000	72
29	CU-03	Docker	2	2	10000	51
30	CU-03	Docker	2	3	10000	45

ANEXO R: Registro de recopilación de datos del Caso de Uso 4

Nro.	Caso de Uso	Ambiente	Nivel	Repeticiones	Nro. Registros	Tiempo (ms)
1	CU-04	Localhost	3	1	1	35
2	CU-04	Localhost	3	2	1	3
3	CU-04	Localhost	3	3	1	2
4	CU-04	Localhost	3	1	10	32
5	CU-04	Localhost	3	2	10	3
6	CU-04	Localhost	3	3	10	3
7	CU-04	Localhost	3	1	100	37
8	CU-04	Localhost	3	2	100	6
9	CU-04	Localhost	3	3	100	5
10	CU-04	Localhost	3	1	1000	65
11	CU-04	Localhost	3	2	1000	24
12	CU-04	Localhost	3	3	1000	13
13	CU-04	Localhost	3	1	10000	131
14	CU-04	Localhost	3	2	10000	95
15	CU-04	Localhost	3	3	10000	81
16	CU-04	Docker	3	1	1	2
17	CU-04	Docker	3	2	1	2
18	CU-04	Docker	3	3	1	1
19	CU-04	Docker	3	1	10	21

20	CU-04	Docker	3	2	10	2
21	CU-04	Docker	3	3	10	3
22	CU-04	Docker	3	1	100	18
23	CU-04	Docker	3	2	100	6
24	CU-04	Docker	3	3	100	6
25	CU-04	Docker	3	1	1000	40
26	CU-04	Docker	3	2	1000	16
27	CU-04	Docker	3	3	1000	15
28	CU-04	Docker	3	1	10000	120
29	CU-04	Docker	3	2	10000	96
30	CU-04	Docker	3	3	10000	66

ANEXO S: Registro de recopilación de datos del Caso de Uso 5

Nro.	Caso de Uso	Ambiente	Nivel	Repeticiones	Nro. Registros	Tiempo (ms)
1	CU-05	Localhost	4	1	1	37
2	CU-05	Localhost	4	2	1	5
3	CU-05	Localhost	4	3	1	3
4	CU-05	Localhost	4	1	10	38
5	CU-05	Localhost	4	2	10	4
6	CU-05	Localhost	4	3	10	3
7	CU-05	Localhost	4	1	100	35
8	CU-05	Localhost	4	2	100	10
9	CU-05	Localhost	4	3	100	10
10	CU-05	Localhost	4	1	1000	86
11	CU-05	Localhost	4	2	1000	36
12	CU-05	Localhost	4	3	1000	27
13	CU-05	Localhost	4	1	10000	166

14	CU-05	Localhost	4	2	10000	117
15	CU-05	Localhost	4	3	10000	89
16	CU-05	Docker	4	1	1	14
17	CU-05	Docker	4	2	1	8
18	CU-05	Docker	4	3	1	7
19	CU-05	Docker	4	1	10	11
20	CU-05	Docker	4	2	10	13
21	CU-05	Docker	4	3	10	11
22	CU-05	Docker	4	1	100	6
23	CU-05	Docker	4	2	100	13
24	CU-05	Docker	4	3	100	15
25	CU-05	Docker	4	1	1000	65
26	CU-05	Docker	4	2	1000	10
27	CU-05	Docker	4	3	1000	9
28	CU-05	Docker	4	1	10000	46
29	CU-05	Docker	4	2	10000	132
30	CU-05	Docker	4	3	10000	135