Migración de la plataforma COMA desplegada para cada escuela a un solo servidor y realización de actividades de mantenimiento

Juan David Escalante Pinilla y Daniel Mauricio Pérez Bolívar

Trabajo de Grado para Optar al Título de Ingeniero de Sistemas

Director

Luis Ignacio González Ramírez

Magíster en Informática

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería de Sistemas e Informática

Ingeniería de Sistemas

Bucaramanga

2025

Dedicatoria

A mis padres, Juan Carlos y Luisa Esmeralda, cuya dedicación y amor incondicional han sido mi mayor fuente de inspiración y motivación durante este recorrido. A ustedes les debo la vida y la increíble formación que, con esfuerzo y sacrificio, me dieron como persona.

Al amor de mi vida, Karla Valentina, quien ha estado a mi lado desde los inicios de mi formación como profesional. Tu paciencia, apoyo y amor constante me han permitido fijar un objetivo de vida juntos.

A mis hermanas, Andrea y Natalia, quienes son un gran ejemplo a seguir. Me han motivado a siempre dar lo mejor de mí, sin importar la situación.

Juan David

Primero, a Dios por estar siempre presente en mi vida, bendiciéndome y guiándome sin desampararme en ningún momento. Gracias por darme la sabiduría, paciencia y entendimiento necesarios para culminar con éxito mi proceso educativo.

A mi familia, en especial a mis padres, Wilson Pérez y María Bolívar; a mi abuelo, Alfonso Bolívar; a mi abuela, Francisca Salamanca; y a mi hermano, Camilo Pérez, quienes han sido mi mayor motivación para seguir adelante. Su amor incondicional, apoyo constante y sacrificios han sido pilares fundamentales en mi vida. Su presencia en los momentos más difíciles de mi carrera y de mi vida ha sido invaluable. Gracias a ellos, he logrado llegar hasta aquí y cumplir esta meta. No sé qué sería de mí sin ellos. Los amo con todo mi corazón.

A mis amigos más cercanos, quienes también me han apoyado y acompañado a lo largo de este camino. De cada uno de ellos he aprendido valiosas lecciones y he compartido momentos inolvidables. Gracias por permitirme conocer a personas tan maravillosas.

Daniel Pérez

Agradecimientos

A mi director, el profesor Luis Ignacio, por su invaluable apoyo, guía y confianza durante mi formación como profesional. Esta experiencia ha sido bastante enriquecedora, donde no solo logré crecer profesionalmente, sino como persona. Estoy infinitamente agradecido de haber tenido esta oportunidad de aprendizaje bajo su liderazgo.

A mis compañeros del grupo de innovación y desarrollo CALUMET por su constante apoyo, colaboración y amistad. Sus aportes fueron esenciales para hacer realidad este trabajo, y gracias a ustedes el grupo se convirtió en un lugar de confianza, risas y aprendizaje continuo. Espero siempre estar en contacto con ustedes.

A mi compañera de vida, quien siempre ha estado junto a mi frente a cualquier situación. Cada día te agradezco por siempre estar a mi lado en los momentos difíciles, y por celebrar cada logro juntos.

A mi familia, por estar presentes frente a cualquier adversidad. Su paciencia y apoyo emocional me ha permitido seguir adelante y cumplir mis objetivos.

A mi gran amigo Daniel por el apoyo y escucha en los momentos más cruciales. Este trabajo es fruto no solo por nuestro trabajo colaborativo; también por los consejos y disposición para seguir adelante frente a cualquier reto.

A mis amigos Óscar, Santiago y Laura, quienes desde quinto semestre me han brindado apoyo, confianza y sabiduría. Gracias por ser una parte esencial de este camino, tanto en los buenos momentos como en los desafíos.

Juan David

Agradezco a mis padres, abuelos y hermano, quienes han sido mi fuente inagotable de amor, apoyo y motivación. Gracias a ellos he podido llegar tan lejos y convertirme en la persona que soy hoy.

A mi director, Luis Ignacio Gonzales, por su apoyo y guía desde mi ingreso al grupo. Su enseñanza no solo me ha permitido crecer como profesional, sino también como ser humano.

A mis compañeros del grupo CALUMET, quienes con sus aportes contribuyeron significativamente a la realización de este trabajo y convirtieron al grupo en un lugar lleno de alegría y compañerismo.

A mi compañero de trabajo y gran amigo Juan, quien ha sido una fuente constante de sabiduría y apoyo, tanto en lo académico como en lo personal.

A Santiago, Óscar y Laura, una persona muy especial para mí, quienes estuvieron a mi lado a lo largo de toda mi carrera. Juntos compartimos alegrías y tristezas, avanzando paso a paso y conquistando pequeños logros hasta llegar a este momento.

A todos ellos, les expreso mis más sinceros agradecimientos.

Daniel Pérez

Tabla de Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Lista de Apéndices

pág.

Glosario

Bash: intérprete de comandos utilizado en sistemas operativos Unix y Linux. Permite la ejecución de comandos para interactuar con el sistema.

CALUMET: grupo de innovación y desarrollo de la Universidad Industrial de Santander.

COMA: acrónimo de la plataforma web Comunidad Académica.

Cron: administrador de tareas de Linux que permite ejecutar comandos en un momento de terminado.

Crontab: archivo de texto donde se listan todas las tareas que deben ejecutarse en el cron y en qué momento deben hacerse.

Elise: framework de estilos creado por el grupo CALUMET.

Git: sistema de control de versiones distribuido ampliamente usado para el seguimiento de cambios en archivos y colaboración en proyectos de desarrollo de software.

GitHub: plataforma de desarrollo colaborativo para alojar proyectos en la nube que usan Git como sistema de control de versiones.

Repositorio: espacio de almacenamiento utilizado para guardar código fuente y su historial de cambios en proyectos de software.

Servidor web: ordenador que almacena, procesa y entrega archivos de sitios web a los usuarios desde un navegador. Utilizan el protocolo HTTP.

Resumen

Título: Migración de la plataforma COMA desplegada para cada escuela a un solo servidor y realización de actividades de mantenimiento[*]

Autor: Juan David Escalante Pinilla y Daniel Mauricio Pérez Bolívar[**]

Palabras Clave: Servidor centralizado, comunidad académica, contenedores, migración, optimización, eficiencia, COMA, administración óptima, mantenimiento.

Descripción:

La plataforma Comunidad Académica (COMA) consiste en una serie de portales web disponibles para las escuelas y facultades de la Universidad Industrial de Santander (UIS). Gracias a sus amplias funcionalidades, esta es utilizada por numerosos usuarios, facilitando la unión e integración de la comunidad universitaria. Sin embargo, debido a que cada portal estaba desplegado en servidores obsoletos, el tiempo de respuesta no era el esperado. Además, su naturaleza descentralizada dificultaba su gestión y administración.

Este proyecto se fundamenta en la necesidad de mejorar la experiencia de usuarios y facilitar la administración de la plataforma. El objetivo es migrar los diversos portales a un servidor centralizado utilizando la herramienta Docker. Se incluyen también tareas de mantenimiento realizadas en COMA en módulos de trabajos de grado, aula virtual, entre otros. Se destaca, en particular, el desarrollo de un algoritmo para la actualización automática de bases de datos.

Se detalla la arquitectura original de los servidores, el proceso de selección de la tecnología a utilizar y la propuesta de arquitectura desarrollada e implementada por los autores. Asimismo, se describe el proceso llevado a cabo mediante la metodología de prototipos evolutivos, hasta alcanzar la arquitectura deseada, junto con su posterior despliegue en producción.

Además, se aborda el desarrollo de herramientas diseñadas para el mantenimiento de los portales en esta nueva arquitectura, así como las pruebas realizadas para evaluar su estabilidad y las mejoras obtenidas en comparación con la arquitectura anterior. Finalmente, se presentan las conclusiones del trabajo y se proponen líneas de trabajo futuro derivadas de la migración de los servidores.

---

[*] Trabajo de Grado

[**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática. Ingeniería de Sistemas. Director: Luis Ignacio González Ramírez. Magíster en Informática.

Abstract

Title: Migration of the COMA platform deployed for each school to a single server and carrying out maintenance activities[*]

Author(s): Juan David Escalante Pinilla y Daniel Mauricio Pérez Bolívar[**]

Key Words: Centralized server, academic community, containers, migration, optimization, efficiency, COMA, optimal management, maintenance.

Description:

The Comunidad Académica (COMA) platform consists of a series of web portals available for the schools and faculties of the Industrial University of Santander (UIS). Thanks to its extensive functionalities, it is widely used by numerous users, facilitating the integration of the university community. However, since each portal was hosted on outdated servers, the response times were not as expected. Additionally, their decentralized nature made management and administration difficult.

This project is based on the need to improve user experience and simplify the platform's administration. The objective is to migrate the various portals to a centralized server using Docker. It also includes maintenance tasks performed on COMA in modules such as degree work, virtual classrooms, among others. Notably, the development of an algorithm for the automatic updating of internal databases is highlighted.

The original server architecture, the selection process of the technology to be used, and the proposed architecture developed and implemented by the authors are detailed. Furthermore, the process carried out using the evolutionary prototyping methodology is described, from archiving the desired architecture to its subsequent deployment in production.

In addition, the development of tools designed for maintaining the portals in this new architecture is addressed, as well as the tests conducted to evaluate its stability and the improvements achieved compared to the previous architecture. Finally, conclusions are presented, alongside with proposals for future work stemming from the server migration.

---

[*] Degree Work

[**]Faculty of Physical-Mechanical Engineering. School of Systems Engineering and Informatics. Systems Engineering. Advisor: Luis Ignacio González Ramírez. Master in Computer Science.

Introducción

La plataforma web COMA (Comunidad Académica) consiste en una serie de portales web disponibles para las escuelas y facultades de la Universidad Industrial de Santander (UIS). Esta desempeña un papel fundamental en el contexto de la universidad gracias a la amplitud de servicios que ofrece. Tanto administrativos y profesores, como estudiantes y egresados hacen uso de la plataforma, convirtiéndola en un medio digital que permite unir e integrar a la comunidad universitaria.

El impacto positivo de la plataforma en la universidad implica el cumplimiento de requerimientos que lleven al correcto y ágil funcionamiento de los portales. El grupo de innovación y desarrollo CALUMET, perteneciente a la Escuela de Ingeniería de Sistemas e Informática de la UIS, es autor y responsable de proporcionar mantenimiento, soporte, nuevas funcionalidades y servicios a los diversos portales web que componen el sistema según las necesidades de los usuarios.

Previo a la realización de este proyecto, COMA se encontraba desplegado en diversos servidores en la nube de Azure, uno para cada escuela, los cuales contaban con especificaciones técnicas obsoletas que afectaban negativamente su funcionamiento y tiempo de respuesta. Además, su estructura totalmente descentralizada dificulta su gestión y actualización, lo que incrementa el tiempo y esfuerzo requerido para desplegar los cambios desarrollados.

En este contexto, surge la necesidad de migrar la plataforma a un servidor centralizado con especificaciones pertinentes, con el fin de mejorar el tiempo de respuesta de los diferentes portales y facilitar su administración. Durante este trabajo, se realizó la migración completa de la plataforma, partiendo de una investigación previa y análisis de las diferentes alternativas que

permitan cumplir este objetivo. Adicionalmente, se explican las diversas labores de mantenimiento realizadas en la plataforma por los autores, junto con la descripción de la nueva herramienta utilizada para desplegar actualizaciones y administrar la plataforma web. Finalmente, se describe una validación mediante pruebas de rendimiento, estrés y resistencia, además de la entrega de la documentación que detalla las acciones a tomar para realizar la administración y actualización de los diversos portales de acuerdo con la nueva arquitectura de COMA.

1. Planteamiento y Justificación del Problema

1.1 Definición de la Situación Actual

Actualmente, los diversos portales web de la plataforma COMA se encuentran desplegados en servidores independientes en la nube de Azure, cada uno con especificaciones técnicas limitadas, como 2GB de RAM y un procesador de un sólo núcleo y un hilo de ejecución. Esto representa una limitación considerable en el rendimiento de la plataforma, provocando problemas de saturación en ciertos portales, lo que se traduce en tiempos de respuesta lentos y tiempos de carga prolongados, afectando negativamente la capacidad de la plataforma para cumplir con las expectativas y necesidades de los usuarios.

Adicionalmente, existen desafíos en la administración, mantenimiento y realización de copias de seguridad de los portales web. Actualmente se utiliza una herramienta interna llamada "Cóndor", desarrollada por el grupo CALUMET, que presenta ciertas limitaciones. Por ejemplo, para cada actualización de la plataforma, es necesario listar manualmente todos los archivos modificados que se van a desplegar, lo cual es propenso a errores. Asimismo, el proceso de realizar copias de seguridad locales de cada portal web es manual y repetitivo, ya que requiere acceder por SSH a cada servidor individualmente. Esta tarea implica que el administrador del sistema debe estar presente y atento para ingresar las contraseñas en cada servidor cuando la línea de comandos lo solicita.

Finalmente, los requerimientos de los usuarios son cambiantes, lo que implica una atención pronta frente a solicitudes de nuevos servicios, cambios o solución de problemas para aumentar el tiempo de vida útil de la plataforma.

1.2 Justificación

La plataforma COMA es una herramienta web que permite la integración entre los miembros de la comunidad universitaria. En ella existen diversos servicios a disposición de la comunidad como la publicación de la hoja de vida, difusión de noticias y eventos, compartir documentos, envío de correos, aula virtual, sistema de gestión de trabajos de grado, entre muchos otros; que contribuyen al cumplimiento de la misión institucional de la UIS. Debido al impacto positivo de la plataforma en los diferentes entes universitarios, se deben constantemente realizar labores de mantenimiento y mejoras para satisfacer a los usuarios, aumentando así el tiempo de vida útil del software. A pesar del soporte constante que se brinda a COMA, en ocasiones se presentan cambios externos en políticas organizacionales que afectan servicios específicos. Estos desafíos requieren de una respuesta ágil, destacando la importancia de adaptarse a un entorno cambiante.

Desde la creación del proyecto en el año 2003, los servicios a disposición de los usuarios han ido aumentando, llevando a que la mayoría de las escuelas opten por tener su propio sitio, logrando la creación de más de 30 portales, cada uno desplegado en servidores independientes de la plataforma Azure. Esta fragmentación conlleva a un desaprovechamiento de recursos, debido a que algunas escuelas utilizan el portal con mayor frecuencia y demanda que otras, resultando en tiempos de respuesta desiguales. Además, la estructura descentralizada dificulta la tarea de administración y despliegue de actualizaciones, pues esto implica un proceso manual. Los servidores utilizados previamente no lograban satisfacer las demandas de rendimiento y capacidad requeridas para su uso frecuente, lo que generaba frustración en los usuarios al no percibir un tiempo de respuesta adecuado.

La necesidad de realizar mantenimiento a COMA y migrar los portales a un servidor centralizado radica en la importancia de brindar un servicio de calidad a los usuarios, dónde se garantice una experiencia uniforme y eficiente para todos los miembros de la comunidad universitaria, además de facilitar su mantenimiento.

2. Objetivos

2.1 Objetivo General

Migrar la plataforma COMA a un único servidor para optimizar el uso de los recursos y facilitar su administración. Además, atender las labores de mantenimiento para hacer más eficiente y funcional los servicios a los usuarios.

2.2 Objetivos Específicos

Migrar la plataforma COMA a un servidor centralizado utilizando contenedores para su despliegue a partir de un análisis de la arquitectura de la plataforma web y las diversas alternativas.

Optimizar el rendimiento y uso de recursos de las plataformas en el nuevo servidor.

Facilitar la administración y mantenimiento de los diversos portales web por parte del grupo CALUMET, integrando herramientas y procesos que faciliten el despliegue de actualizaciones.

Realizar labores de mantenimiento, corrección de errores y mejoras en la plataforma COMA con el fin de atender a los requerimientos de los usuarios.

Desarrollar e implementar un algoritmo de actualización automática para las bases de datos de COMA, con el fin de disminuir tareas manuales y tediosas.

3. Marco de Referencia

En el contexto de la migración de plataformas a un servidor, es fundamental comprender las tecnologías y herramientas que impulsan este proceso de transformación digital, así como conocer las tecnologías utilizadas en el desarrollo de la plataforma COMA que son empleadas en su mantenimiento y adición de funcionalidades. En este marco teórico, se profundizará en la comprensión de las tecnologías y herramientas clave, así como en su integración y aplicación efectiva.

3.1 Base de Datos Relacional

La base de datos usada en COMA es una base de datos relacional. Esta organiza los datos en tablas compuestas por filas y columnas, donde cada fila representa un registro único y cada columna representa un atributo (*¿Qué Es Una Base De Datos Relacional?*, s.f.). Las relaciones entre las tablas se establecen mediante claves primarias y foráneas. Esto permite consultas complejas y relaciones entre los datos, lo que lo hace ideal para una amplia gama de aplicaciones.

3.1.1 MySQL

La base de datos utilizada en la plataforma es MySQL, más exactamente la versión 8.3.0. Este es un sistema de gestión de base de datos relacional que se utiliza para almacenar y administrar datos de manera estructurada (*MySQL :: Developer Zone.*, s.f.). Proporciona un método eficiente para organizar y acceder a grandes volúmenes de información de manera rápida y segura (Vanier et al., 2018). Dentro del proyecto se utilizan principalmente 2 grandes bases de datos, la primera es una base de datos común para todas las escuelas esta es llamada poseidon, la segunda es única para escuela llamada diamante.

3.2 Java

Lenguaje de programación utilizado para la parte lógica del proyecto COMA, el cual es un lenguaje de programación de alto nivel y orientado a objetos, desarrollado por Sun Microsystems (ahora propiedad de Oracle Corporation) (*Java Documentation - Get Started*, s.f.). Además del lenguaje en sí, Java incluye un entorno de ejecución que permite la portabilidad de las aplicaciones desarrolladas en Java a diferentes plataformas (Schildt, 2021). Por medio de este se realizan las conexiones a la base y el CRUD del proyecto, así como sirve los datos que se muestran en la página web.

3.3 JSP

JavaServer Pages es una tecnología utilizada para crear páginas web dinámicas en Java. Permite la separación del código Java y el contenido HTML, lo que facilita el desarrollo de aplicaciones web escalables y mantenibles. Por medio de esta tecnología se crean archivos de peticiones los cuales implementan los métodos de las clases creadas en Java; estas son enviadas desde el navegador por medio de Javascript (*Tecnología JSP (JavaServer Pages)*, s.f.).

3.4 Apache Tomcat

Servidor web de código abierto desarrollado por la Apache Software Foundation que implementa las tecnologías de Java Servlet, JavaServer Pages (JSP) y WebSockets, entre otras, para permitir la ejecución de aplicaciones web Java [8]. Funciona como un contenedor de servlets que gestiona las solicitudes de los clientes y las dirige a los componentes adecuados para su procesamiento. Tomcat es ampliamente utilizado para el desarrollo y la implementación de aplicaciones web dinámicas y escalables, ofreciendo una plataforma robusta y confiable para ejecutar aplicaciones Java en entornos de producción. En este servidor se encuentra desplegado

actualmente cada una de las plataformas de las escuelas, la versión 9.0.95 será utilizada en la migración (*Apache Tomcat*®, s.f.).

3.5 Máquina Virtual

Representación virtual, o emulación, de una computadora física. La virtualización permite crear varias máquinas virtuales, cada una con su propio sistema operativo y aplicaciones en una única máquina física. Una máquina virtual no puede interactuar directamente con un sistema físico. En cambio, necesita una capa de software ligera, llamada hipervisor, para coordinarse con el hardware físico subyacente. El hipervisor asigna recursos informáticos físicos, como procesadores, memoria y almacenamiento, a cada máquina virtual. Mantiene cada máquina virtual separada de las otras para que no interfieran entre sí (*¿Qué Es Una Máquina Virtual (VM)?*, 2024).

3.6 Kubernetes

Plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa. Tiene un ecosistema grande y en rápido crecimiento. Kubernetes ofrece un entorno de administración centrado en contenedores; este orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo de los usuarios no tengan que hacerlo. Esto ofrece la simplicidad de las Plataformas como Servicio (PaaS) con la flexibilidad de la Infraestructura como Servicio (IaaS) y permite la portabilidad entre proveedores de infraestructura (*¿Qué Es Kubernetes?*, 2022).

3.7 Docker

Plataforma de código abierto que permite empaquetar, distribuir y ejecutar aplicaciones dentro de contenedores. Los contenedores son entornos ligeros y portátiles que encapsulan todo lo necesario para ejecutar una aplicación, incluyendo el código, las bibliotecas y las dependencias,

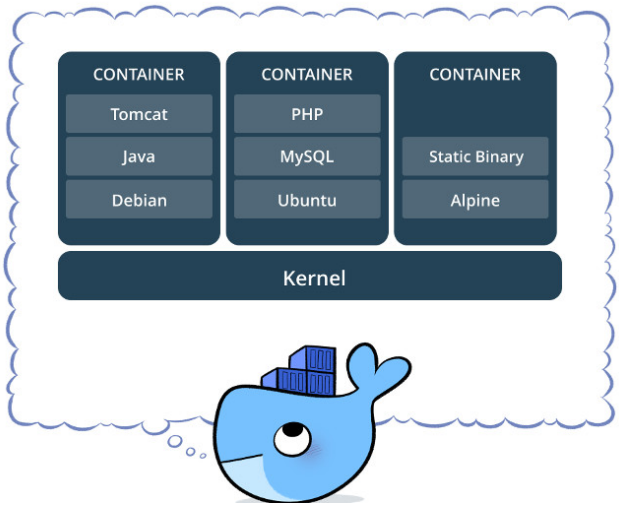
de manera que la aplicación pueda ejecutarse de manera consistente en cualquier entorno donde Docker esté instalado (Jangla, 2018).

El funcionamiento de los contenedores se basa en la tecnología de virtualización a nivel de sistema operativo. A diferencia de las máquinas virtuales tradicionales, que incluyen un sistema operativo completo y a menudo son más pesadas en términos de recursos, los contenedores comparten el mismo núcleo del sistema operativo del host, lo que los hace más eficientes en términos de uso de recursos y más rápidos para iniciar y detener.

Los contenedores se crean a partir de imágenes, que son plantillas de solo lectura que contienen el sistema operativo, las bibliotecas y las dependencias necesarias para ejecutar una aplicación. Estas imágenes se pueden compartir y reutilizar a través de Docker Hub u otros registros de contenedores. Cuando se ejecuta un contenedor a partir de una imagen, se agrega una capa de escritura encima de la imagen base, lo que permite que el contenedor sea modificable y que los cambios realizados durante la ejecución se almacenen de manera persistente.

Figura 1

Arquitectura básica de Docker



Nota. El gráfico representa la manera en que Docker aísla los contenedores. Tomado de *Servidores Admin*, 2020. (https://www.servidoresadmin.com/que-es-docker-y-como-funciona-introduccion-a-docker).

3.8 Nginx

Es un software de servidor web de código abierto que también puede funcionar como proxy inverso, balanceador de carga y servidor de caché. Fue diseñado para manejar conexiones concurrentes de manera eficiente, lo que lo hace ideal para sitios web con alto tráfico. NGINX se utiliza principalmente para servir contenido web estático, mejorar el rendimiento de aplicaciones, distribuir el tráfico entre varios servidores y asegurar la entrega de contenido rápido y confiable. Su capacidad para manejar múltiples tareas simultáneamente lo hace popular en entornos modernos de desarrollo y despliegue de aplicaciones web. (*¿Qué Es NGINX Y Cómo Funciona? NGINX Explicado Para Principiantes*, 2018)

3.9 Reverse Proxy

Es un servidor que actúa como intermediario entre los clientes y uno o más servidores de backend, recibiendo las solicitudes de los usuarios y redirigiéndolas al servidor correspondiente antes de devolver la respuesta. Ofrece beneficios como balanceo de carga, que distribuye las solicitudes entre varios servidores para mejorar la eficiencia; seguridad, al ocultar la identidad de los servidores de backend; almacenamiento en caché, que reduce la carga y mejora el tiempo de respuesta; y manejo de SSL/TLS, facilitando la gestión de certificados y encriptación. En resumen, un *reverse proxy* optimiza el rendimiento y la seguridad en un entorno de red. (*What Is a Reverse Proxy? | Reverse Proxy Server - Kemp*, s.f.)

3.10 Certificado SSL/TLS de Let's Encrypt

Un certificado SSL/TLS es un objeto digital que permite a los sistemas verificar la identidad y, posteriormente, establecer una conexión de red cifrada con otro sistema mediante el protocolo Secure Sockets Layer/Transport Layer Security (SSL/TLS). Los certificados se emiten con un sistema criptográfico conocido como infraestructura de claves públicas (PKI) (*¿Qué Es Un Certificado SSL? - Explicación Del Certificado SSL/TLS*, s.f.). Este certificado es otorgado Let's Encrypt, una autoridad de certificación gratuita, automatizada, y abierta traída por la organización sin ánimos de lucro Internet Security Research Group (ISRG). El objetivo de Let's Encrypt y el protocolo ACME es hacer posible configurar un servidor HTTPS y permitir que este genere automáticamente un certificado válido para navegadores, sin ninguna intervención humana. (*Cómo Funciona*, 2019)

3.11 Certbot

Es una herramienta de código abierto y gratuita que facilita la obtención y renovación de certificados SSL/TLS para asegurar sitios web, utilizando el servicio de Let's Encrypt. Su propósito es ayudar a los administradores de servidores a implementar HTTPS de forma fácil y automática, lo que mejora la seguridad de las comunicaciones en línea. Certbot simplifica el proceso de configuración y renovación de certificados, permitiendo que los sitios web puedan ofrecer conexiones cifradas, protegiendo la privacidad de los usuarios y garantizando una navegación segura (*About Certbot*, s.f.).

3.12 Github Actions

Es una plataforma de automatización que permite crear flujos de trabajo personalizados para automatizar tareas en los proyectos de desarrollo. Integrada en GitHub, permite configurar procesos como la integración continua (CI) y la entrega continua (CD), que ayudan a compilar,

probar y desplegar el código de forma automática. GitHub Actions se basa en "workflows" (flujos de trabajo), que son conjuntos de tareas definidas en archivos YAML y se ejecutan en respuesta a eventos específicos, como un push, una pull request o a una programación personalizada. Esto facilita la automatización de tareas repetitivas, la mejora de la calidad del código y el despliegue rápido de aplicaciones. (*Entender Las GitHub Actions - Documentación De GitHub*, s.f.)

3.13 Node.js

Es un entorno de ejecución de JavaScript de código abierto y multiplataforma, que permite ejecutar código JavaScript en el servidor, fuera del navegador. Basado en el motor V8 de Google Chrome, Node.js se caracteriza por su arquitectura orientada a eventos y su modelo de ejecución asíncrono, lo que lo hace muy eficiente para desarrollar aplicaciones escalables y de alto rendimiento. Esto es ideal para crear aplicaciones web, APIs, servidores y aplicaciones en tiempo real como chats y juegos. Node.js permite a los desarrolladores usar JavaScript tanto en el frontend como en el backend, facilitando la creación de aplicaciones completas usando un solo lenguaje (*Qué Es NodeJS Y Para Qué Sirve*, 2019).

3.14 JMeter

Apache JMeter es una herramienta de código abierto diseñada para pruebas de rendimiento y carga, enfocada en analizar y medir el comportamiento de servicios y aplicaciones web bajo diferentes condiciones de carga. Permite simular múltiples usuarios concurrentes y recopila datos sobre la velocidad, estabilidad y capacidad de respuesta del sistema, ayudando a identificar posibles cuellos de botella. Su interfaz es flexible y soporta varios protocolos como HTTP, FTP, y bases de datos JDBC, lo que la hace versátil para distintas pruebas. (*Apache JMeter*, s.f.)

3.15 Pruebas de Software

Son un proceso de evaluación que asegura que una aplicación cumple con sus funciones esperadas, ayudando a evitar errores y a mejorar el rendimiento. Estas pruebas son especialmente efectivas cuando se realizan de manera continua, lo cual permite detectar problemas desde el diseño hasta el despliegue en producción. Existen diversos tipos de pruebas, como pruebas unitarias, de rendimiento, de seguridad, y de usabilidad, cada una adaptada a aspectos específicos de la calidad y confiabilidad del software. (*¿Qué Son Las Pruebas De Software?*, 2024)

4. Metodología

Para garantizar el éxito en la realización del proyecto y cumplir los objetivos, se decide por utilizar como metodología la estrategia de migración de aplicaciones de IBM, teniendo en cuenta que la migración de la plataforma COMA va desde la nube hacia un servidor centralizado. El patrón utilizado para la migración es el de cambio de plataforma, también conocido como realojamiento, el cual implica hacer ciertos cambios en la aplicación web para que pueda obtener mejor beneficio de la nueva arquitectura planteada (IBM, s.f.). Esta estrategia consta de cuatro etapas o fases principales: identificación y descubrimiento, la evaluación de candidatos, la planeación y desarrollo de la nueva arquitectura y la migración de la plataforma.

4.1 Fase de Identificación y Descubrimiento

Esta fase está orientada a conocer y analizar la arquitectura de la aplicación web COMA, partiendo de la realización de actividades de mantenimiento. Con ello se obtiene un mayor conocimiento sobre cómo está estructurada la aplicación web y sus dependencias, para realizar un análisis de la misma y evaluar la viabilidad del proyecto.

4.2 Fase de Evaluación de Candidatos

Esta fase consiste en evaluar la complejidad, la criticidad, el impacto y establecer las posibles alternativas de solución para una migración exitosa. Entre ellas, se encuentran máquinas virtuales, Docker y Kubernetes. Al final de esta fase, se decidió que Docker es la tecnología adecuada para la migración de COMA a un servidor centralizado.

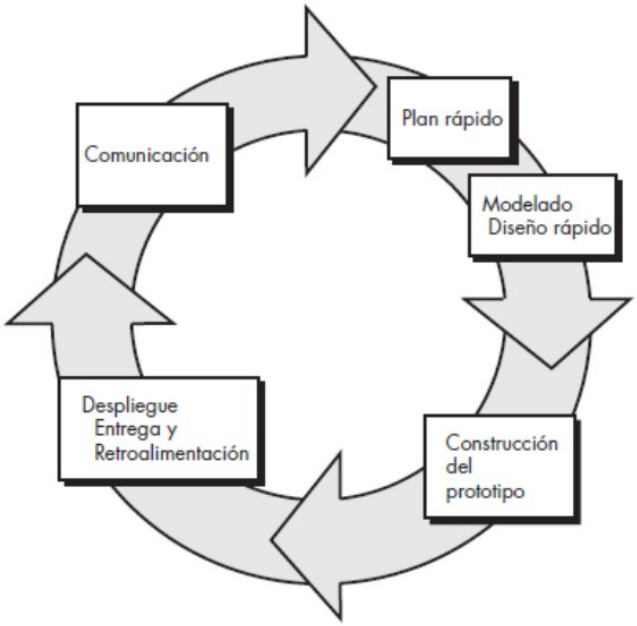
4.3 Fase de Planificación y Desarrollo

Después de elegir el candidato adecuado, se procede a planificar el desarrollo de la nueva arquitectura. Para esto, se decide utilizar el modelo de prototipo evolutivo, el cual se caracteriza

por permitir desarrollar versiones cada vez más completas de un software o sistema (Celi-Párraga et al., 2023, 24).

Figura 2

Modelo de Prototipo Evolutivo



Nota. El gráfico representa las fases del modelo de prototipo evolutivo. Adaptado de *Ingeniería del Software I: Requerimientos y Modelado del Software* (p. 25), 2023.

> Según Celi et al. (2023), el modelo inicia con comunicación, la cual consiste en definir los objetivos generales del software, identificar requerimientos y detectar áreas que requieran una mayor definición. Se planea rápidamente una iteración para hacer el prototipo, y se lleva a cabo el modelado en forma de un "diseño rápido", el cual lleva a la construcción de un prototipo. Este se entrega y es evaluado por los participantes, que dan retroalimentación para mejorar los requerimientos. La iteración ocurre a medida que el prototipo es afinado para satisfacer las necesidades de distintos participantes, y permite a los desarrolladores entender mejor qué es lo que se va a hacer (p. 24-25).

En este sentido, se realizaron diversos prototipos que llevaron a una arquitectura sólida y estable para la plataforma COMA utilizando contenedores. Adicionalmente, también se detalla la herramienta creada para facilitar la administración de la nueva arquitectura.

IBM recomienda que durante la fase de planificación y desarrollo se realicen pruebas de manera constante para verificar la integridad de datos y validar que estos se encuentren en la ubicación correcta, por lo que se crea y ejecuta un plan de pruebas donde se compare la arquitectura descentralizada con la nueva arquitectura. Finalmente, se realiza el despliegue de la nueva arquitectura en producción.

4.4 Fase de Migración

Esta fase consiste en la migración de la plataforma a producción a partir de la arquitectura obtenida de la fase anterior. Se detallan las particularidades que se tienen en cuenta para que la migración de arquitectura sea exitosa, además de documentos finales que detallan dicha arquitectura. Según IBM, es esencial monitorear constantemente la nueva arquitectura en producción para atender cualquier eventualidad.

5. Desarrollo del proyecto

5.1 Fase de Identificación y Descubrimiento

5.1.1 Realización de Actividades de Mantenimiento

Para obtener conocimiento sobre la arquitectura de la aplicación web COMA, se realizaron diversas labores de mantenimiento, la cual consiste en correcciones de errores, mejoras y nuevas funcionalidades en los diferentes módulos de la plataforma COMA. A continuación, se mencionan dichas labores realizadas durante el tiempo de colaboración en CALUMET por parte de los autores.

En el módulo de trabajos de grado, se implementaron correcciones y se añadieron nuevas funcionalidades. Entre las principales mejoras se destacan:

- La inclusión de la posibilidad de visualizar los proyectos asignados a los profesores durante la misma sesión del comité de trabajos de grado.
- La optimización de consultas que causaban demoras excesivas al cargar la visualización de las solicitudes presentadas al comité. Esta corrección redujo el tiempo de respuesta de 20 segundos a menos de un segundo.
- La incorporación de una opción para deshabilitar campos en el formulario del tema de trabajo de grado, permitiendo que las escuelas soliciten deshabilitarlos según sus necesidades. Esto responde a una solicitud de la Escuela de Ingeniería Industrial, que buscaba agilizar el proceso al reducir los campos necesarios en el formulario.
- La adición de una funcionalidad que permite asignar un único calificador para un trabajo de grado en la sesión de asignación de calificadores, debido a otra solicitud

de la Escuela de Ingeniería Industrial, ya que anteriormente el sistema obligaba a asignar siempre dos calificadores.

- El envío de correos a los autores de un trabajo de grado cuando el director toma una decisión sobre una solicitud. Esto permite que los autores estén informados del estado de sus solicitudes tras ser evaluadas por el director.
- La optimización de una consulta que lista los profesores cátedra en el menú de asignar calificadores a un proyecto. Antes mostraba todos los profesores sin importar si son activos en la universidad o no; ahora solo muestra aquellos que son activos, o que han sido creados manualmente por el administrador del sistema.

En el servicio de aula virtual, se destacan las siguientes correcciones y mejoras:

- Se corrige la obtención de la IP en la que un estudiante está respondiendo un examen o un taller para la vista del profesor. Este servicio había dejado de funcionar, y es necesario para que el profesor tenga certeza de que los estudiantes están respondiendo desde un computador del aula de clase.
- Se arregla un problema en la vista del estudiante donde la semana actual no coincide con la configurada por el profesor. Esto es importante debido a que las actividades mostradas para el estudiante, deben corresponder a la semana actual del curso.

En cuanto a ciertas funcionalidades administrativas de COMA, debido a actualizaciones de las plataformas de Google que requieren atención de los desarrolladores, fue necesario realizar los siguientes cambios:

- Se actualizó un fragmento de código relacionado con Google Analytics para permitir el monitoreo y la generación de estadísticas sobre las visitas a las páginas de las escuelas.

- Se realizó una actualización del método de autenticación de envío de correos a través de la plataforma de Google, pues la compañía anunció que, a partir del verano de 2024, se desactivará el acceso a aplicaciones menos seguras, es decir, aquellas que no son de Google y pueden acceder estas cuentas con el método de autenticación básica de usuario y contraseña (Google, s.f.). La plataforma COMA utilizaba este método poco seguro, surgiendo la necesidad de actualizarlo a OAuth, el cual permite que las aplicaciones accedan a las cuentas con una llave digital, sin necesidad de utilizar la contraseña.

Finalmente, se desarrolló un algoritmo de actualización automática para las bases de datos *diamante* de las escuelas usando la base datos *division*.

La Universidad Industrial de Santander (UIS) cuenta con una base de datos denominada *vista*, que almacena información sobre estudiantes, egresados, matrículas, horarios y docentes. Estos datos también están replicados en las bases de datos *diamante* de cada escuela, las cuales necesitan ser actualizadas con frecuencia para garantizar la sincronización entre ambas.

Para facilitar este proceso, se utiliza una base de datos intermedia llamada *division,* también única para cada escuela, la cual contiene la misma información de *vista*, filtrada según los programas académicos de cada escuela. Cada noche, en un horario específico, se actualiza el contenido de *division*, para posteriormente realizar una actualización interna en *diamante* a partir de esta base de datos.

El proceso de actualización de *diamante* se realizaba de manera manual. Los administradores debían ingresar a un módulo para revisar, a través de una tabla, el conteo de datos en la base de datos y evaluar si es conveniente proceder con la actualización. Posteriormente, las tablas se actualizaban de forma manual, una a una, siguiendo un orden específico. Este proceso

presenta múltiples complicaciones, siendo la principal la decisión de la División de Tecnologías de la Información y la Comunicación (DTIC) de la universidad sobre restringir el acceso a *vista* en horario de 12:00 a.m. a 7:00 a.m. Como consecuencia, las actualizaciones de las bases de datos solo podían realizarse durante este periodo, obligando al administrador a ingresar durante dicha franja y seguir una serie de pasos para actualizar correctamente las bases de datos. Con estas restricciones, las actualizaciones se realizaban con poca frecuencia durante el semestre académico, lo que ocasionaba problemas para los estudiantes de simultaneidad, ya que sus datos no se veían reflejados en las bases de datos de COMA, generando inconsistencias y una experiencia negativa para los usuarios.

Por este motivo se decidió realizar un algoritmo que, a partir de los datos presentes en *division*, tenga el criterio para decidir cuándo actualizar las bases de datos y notifique al administrador sobre la decisión tomada acerca de dicha actualización.

Para la implementación, modificó la tabla TP_Escuelas de la base de datos *poseidon*, que es compartida por todas las escuelas. Esta tabla contiene información sobre cada escuela y su estado. Para el algoritmo, se crearon estos dos campos:

- Accion: indica el estado de la actualización.
- ActAutom: señala si la actualización automática se llevó a cabo.

Debido a las necesidades identificadas, se definieron tres estados para el campo *Accion*:

- Forzado: se utiliza cuando es necesario realizar la actualización independientemente de los cálculos que realice el algoritmo.
- Detenido: en este estado, las bases de datos no se actualizarán bajo ninguna circunstancia hasta que se dé una nueva orden.

- Automático: permite que la actualización se lleve a cabo según el criterio del algoritmo.

Para el campo *ActAutom*, se definieron dos estados:

- Activo: indica que la última actualización automática se realizó exitosamente.
- Inactivo: indica que la última actualización automática no se realizó.

El algoritmo funciona de la siguiente manera:

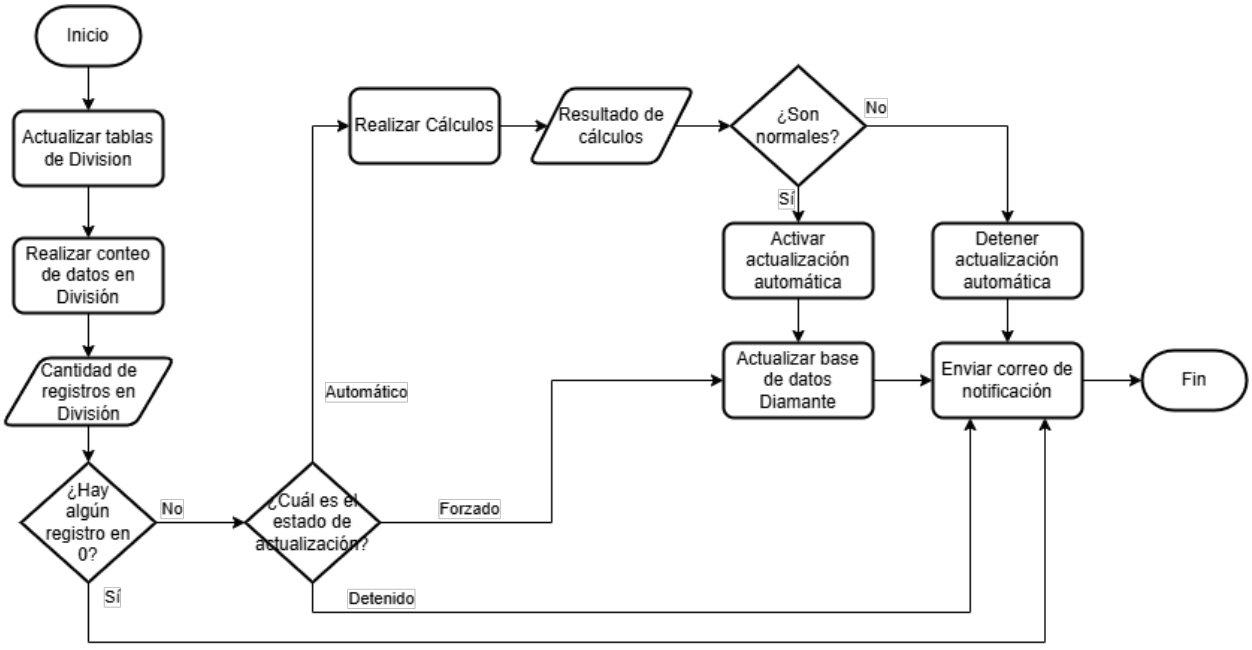
- Inicialmente, se realiza el conteo de los registros de las tablas existentes en la base *division*. Este conteo se agrega a la tabla TB_ContaDivision en la base de datos *diamante*, la cual almacena los registros de los últimos 30 días.
- El algoritmo verifica que ninguno de los campos del último registro tenga valores en 0. Si algún campo tiene un valor de 0, se envía un correo al administrador notificando el motivo por el cual no se realizó la actualización; si todos los campos tienen valores válidos, se continua con el proceso.
- Se verifica el estado del campo *Acción* y, dependiendo de su valor, se ejecutan diferentes acciones: si el estado es Detenido, no se realiza la actualización; si es Forzado, el algoritmo actualiza las bases de datos y cambia el estado a Automático para asegurar que esta acción solo se ejecute una vez; y si el estado es Automático, el algoritmo calcula el promedio de los últimos cinco registros en la tabla TB_ContaDivision y verifica si dicho promedio se encuentra dentro de un margen de error específico (30% en este caso). Si está dentro del margen, los datos se consideran normales y se procede con la actualización; de lo contrario, no se realiza.

- Por último, independientemente de la acción realizada, se envía un correo al administrador notificando los resultados y las decisiones tomadas frente a la actualización.

En la Figura 3 se puede observar el flujo del algoritmo realizado.

Figura 3

Diagrama de flujo del algoritmo de actualización automática



Como este proceso debe realizarse para todos los portales, no es conveniente realizarla en un mismo instante de tiempo. Se decidió distribuir la actualización de los portales según facultad. Por ejemplo, las bases de datos de los portales de las escuelas pertenecientes a la facultad de ingenierías Fisicomecánicas inicia a las 2:00 a.m., mientras que las de la facultad de ingenierías fisicoquímicas inician a las 3:00 a.m. Adicionalmente, a partir de la hora de inicio, los portales actualizan sus bases de datos uno a uno, distribuido cada diez minutos.

Finalmente, se creó un nuevo servicio en la plataforma COMA para que el administrador del sistema pueda monitorear y cambiar el estado de actualización de todas las escuelas. En la Figura 4 se observa la interfaz de este servicio.

Figura 4

Nuevo servicio de administración de actualización de bases de datos

Partiendo de estas labores de mantenimiento, se obtiene un mejor conocimiento sobre la arquitectura de la plataforma web COMA para proceder a realizar un análisis exhaustivo de la misma.

5.1.2 Análisis de la Arquitectura de COMA

A partir de la realización de actividades de mantenimiento, se observa que plataforma web COMA tiene una arquitectura monolítica en la que el backend y el frontend están integrados en un único proyecto y servidor. Utiliza Tomcat como servidor web, encargado de servir las páginas al cliente y gestionar las peticiones al backend. Los lenguajes de programación empleados incluyen Java, JavaServer Pages (JSP), JavaScript, HTML y CSS.

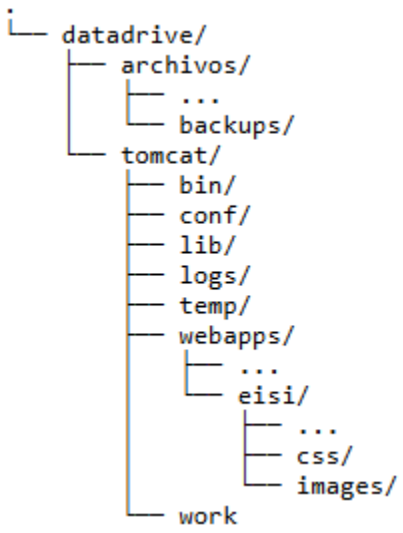
El proyecto COMA implementa principalmente tres bases de datos SQL en MySQL: *diamante, division* y *poseidon.* Las dos primeras son únicas para cada organización académica,

mientras que la última es una base de datos compartida. El detalle del contenido de cada base de datos se explicó en la sección de actividades de mantenimiento, específicamente en el desarrollo del algoritmo de actualización automática.

Previo a la realización de este proyecto, la plataforma COMA estaba desplegada en varios servidores de Azure, uno para cada organización académica. En la Figura 5 se describe la estructura de carpetas en cada servidor, siendo la misma para cada portal web.

Figura 5

Estructura de carpetas de COMA en los servidores de Azure



Dentro de la carpeta webapps se aloja la aplicación web, servida por el servidor Tomcat. El contexto del proyecto COMA es la carpeta eisi, que contiene todo el código base de la plataforma web, incluyendo tanto el backend como el frontend. Además, dentro de esta carpeta, existen directorios específicos para cada organización académica, donde se almacenaban imágenes, colores de los portales y ciertos archivos. Sin embargo, almacenar imágenes y archivos únicos de cada escuela directamente dentro de la estructura del código base de la plataforma limita

la posibilidad de aprovechar ciertas funcionalidades de Tomcat sobre el despliegue de nuevas versiones de la aplicación web, lo que dificulta su gestión y mantenimiento.

Por otro lado, la carpeta archivos almacena documentos y otros elementos subidos por los usuarios. Aunque estos archivos no eran servidos directamente por Tomcat, debido a su ubicación fuera de la carpeta de instalación del servidor, podían ser accedidos mediante la librería ServletOutputStream de Java, que permite enviar datos binarios, como archivos, al cliente.

5.1.3 Viabilidad del Proyecto

Según IBM (s.f.), cuando las organizaciones evalúan la viabilidad de una migración, se consideran los siguientes aspectos.

5.1.3.1 Complejidad

La complejidad de la migración de la plataforma COMA radica en varios factores clave. En primer lugar, COMA ha sido desarrollado internamente por el grupo CALUMET, lo que significa que su director posee un conocimiento profundo sobre su arquitectura. Además, el mantenimiento y análisis realizado permitió comprender las diversas funcionalidades y dependencias que han de ser consideradas para la migración. A pesar de que COMA lleva casi 20 años en desarrollo, ha sido objeto de constantes mejoras y refactorizaciones que han modernizado su arquitectura. Gracias a esto, la plataforma utiliza tecnologías estándar como Tomcat y Java, lo que facilita la actualización de sus versiones para garantizar mayor seguridad y estabilidad. El uso de estas tecnologías minimiza los riesgos asociados a la compatibilidad e integración con la nueva arquitectura.

5.1.3.2 Criticidad

La comunidad universitaria depende constantemente de los distintos portales web de COMA, por lo que ofrecer una buena experiencia de usuario es fundamental. Sin embargo, su arquitectura descentralizada ha afectado negativamente esta experiencia, dificultando tanto el acceso como la implementación de actualizaciones y mejoras.

En este sentido, la migración de COMA es crucial para optimizar su rendimiento y garantizar su sostenibilidad a largo plazo. La transición a un servidor centralizado no solo mejorará la experiencia del usuario, sino que también facilitará la gestión y mantenimiento de la plataforma, asegurando su estabilidad y continuidad operativa.

5.2 Fase de Evaluación de Candidatos

Una vez analizada la arquitectura de COMA, se procede a describir las diferentes herramientas disponibles para la nueva arquitectura, las cuales sirven como alternativas de solución a la migración.

5.2.1 Máquinas Virtuales

Las máquinas virtuales son paquetes de software que proveen una emulación completa de dispositivos de hardware de bajo nivel como CPU, disco y dispositivos de red. Además, pueden incluir una pila de software complementaria que se ejecuta en el hardware emulado. La combinación de estos paquetes produce una instantánea completamente funcional de un sistema informático. (Buchanan, s.f.). En otras palabras, son una abstracción de hardware que permite ejecutar múltiples servidores en una sola máquina.

Entre sus principales ventajas se encuentra el aislamiento total de la máquina, por lo que se ejecuta como un sistema independiente. En este sentido, las máquinas virtuales fueron creadas

principalmente para ejecutar múltiples sistemas operativos en una misma máquina. (*Docker Vs VM – Difference Between Application Deployment Technologies*, s.f.)

Sin embargo, las máquinas virtuales utilizan muchos recursos de la máquina en la que se encuentran alojadas, pues no solo tienen una copia de todo un sistema operativo, sino también del hardware que necesita para ejecutarse. Esto incrementa rápidamente el uso de RAM y ciclos de CPU (Jangla, 2018, 4).

5.2.2 Docker

Plataforma de código abierto que permite empaquetar, distribuir y ejecutar aplicaciones dentro de contenedores. Los contenedores son entornos ligeros y portátiles que encapsulan todo lo necesario para ejecutar una aplicación, incluyendo el código, las bibliotecas y las dependencias, de manera que la aplicación pueda ejecutarse de manera consistente en cualquier entorno donde Docker esté instalado (Jangla, 2018).

Un contenedor consiste en un entorno de ejecución: una aplicación, con sus dependencias y otros binarios, y archivos de configuración necesarios, agrupados en un solo paquete (Jangla, 2018, 2). Aunque a cada contenedor se le puede asignar una cantidad de uso de recursos específica, tiene la ventaja de poder utilizar dinámicamente los recursos del host según las necesidades. El uso de contenedores permite también aislar aplicaciones o servicios.

5.2.3 Kubernetes

Kubernetes es una plataforma de código abierto que orquesta sistemas de tiempo de ejecución de contenedores en un clúster de recursos de red. Kubernetes por dentro puede o no usar Docker (Campbell, s.f.). Facilita la implementación y gestión de sistemas distribuidos complejos. Kubernetes es especialmente útil cuando se requiere de una escalabilidad dinámica y una gestión automatizada de contenedores a gran escala, pues fue diseñado para ejecutar aplicaciones en un

cluster, es decir, en un grupo de máquinas o nodos. Con Kubernetes, es posible garantizar la resiliencia y la alta disponibilidad, reduciendo tiempos de inactividad y mantener la continuidad del servicio en varias situaciones críticas.

5.2.4 Selección de Tecnología

De las tres soluciones posibles, se opta por Docker como la opción más adecuada, dado que la arquitectura de COMA es monolítica. Como cada portal estaba alojado en un único servidor, y en este proyecto no se cambia la estructura de la aplicación web, no es necesario utilizar varios nodos o clústeres para cada portal, por lo que se descarta el uso del modo Swarm de Docker, una herramienta de orquestación de contenedores similar a Kubernetes. Por las mismas razones, también se descarta Kubernetes.

Además, los contenedores presentan ventajas significativas frente a las máquinas virtuales, ya que son más ligeros y gestionan los recursos del servidor de forma dinámica. Esto permite maximizar el uso de los recursos del nuevo servidor, dado que los portales con mayor demanda podrán acceder a los recursos necesarios sin las limitaciones rígidas de asignación propias de las máquinas virtuales.

5.3 Fase de Planificación y Desarrollo

Una vez elegida la tecnología, inicia el proceso de desarrollo de la nueva arquitectura con la metodología de prototipo evolutivo. En total se realizaron 4 prototipos para llegar a la arquitectura final, la cual se espera que sea lo suficientemente robusta para satisfacer a los usuarios con un tiempo de respuesta adecuado. Los dos primeros prototipos se realizaron de manera local en un computador presente en el grupo CALUMET, mientras que los dos finales se realizaron en el nuevo servidor, el cual fue entregado al grupo por la Escuela de Ingeniería de Sistemas e

Informática. A continuación, se detalla todo el proceso de desarrollo y los cuatro prototipos creados, para la migración de la plataforma COMA.

5.3.1 Prototipo inicial: adaptación de COMA a Docker

En primera instancia, es necesario adaptar la plataforma web al entorno de Docker. Para ello, se partió de un análisis de la arquitectura de COMA en los servidores de Azure, observando sus especificaciones, versiones del servidor web Tomcat y la base de datos, estructura de archivos, entre otros aspectos. Se decidió por conservar la estructura de archivos descrita en la Figura 5.

Durante la fase de mantenimiento, y en base al feedback recibido por los desarrolladores del grupo CALUMET, incluidos los autores, se definieron unas versiones de las tecnologías utilizadas en COMA que son compatibles con la arquitectura actual de la aplicación web, sin necesidad de realizar cambios significativos. La Tabla 1 muestra esta comparación de versiones.

Tabla 1

Versiones de tecnologías utilizadas en COMA

Tecnología	Versión en servidor de Azure	Versión en el nuevo servidor
Tomcat	8.0.12	9.0.95
MySQL	5.1.32	5.7.44
Java JDK	7	21 (LTS)

Nota. En esta tabla se observa una comparación entre las versiones de las tecnologías utilizadas en COMA.

En este sentido, se opta por utilizar las versiones mencionadas anteriormente. La decisión de optar por Docker como alternativa de solución facilita escoger la versión establecida, ya que

Docker cuenta con un repositorio de imágenes llamado Docker Hub, las cuales son plantillas que contienen lo suficiente para correr una aplicación en un contenedor. Así, se utilizaron las imágenes de Docker correspondientes a estas versiones.

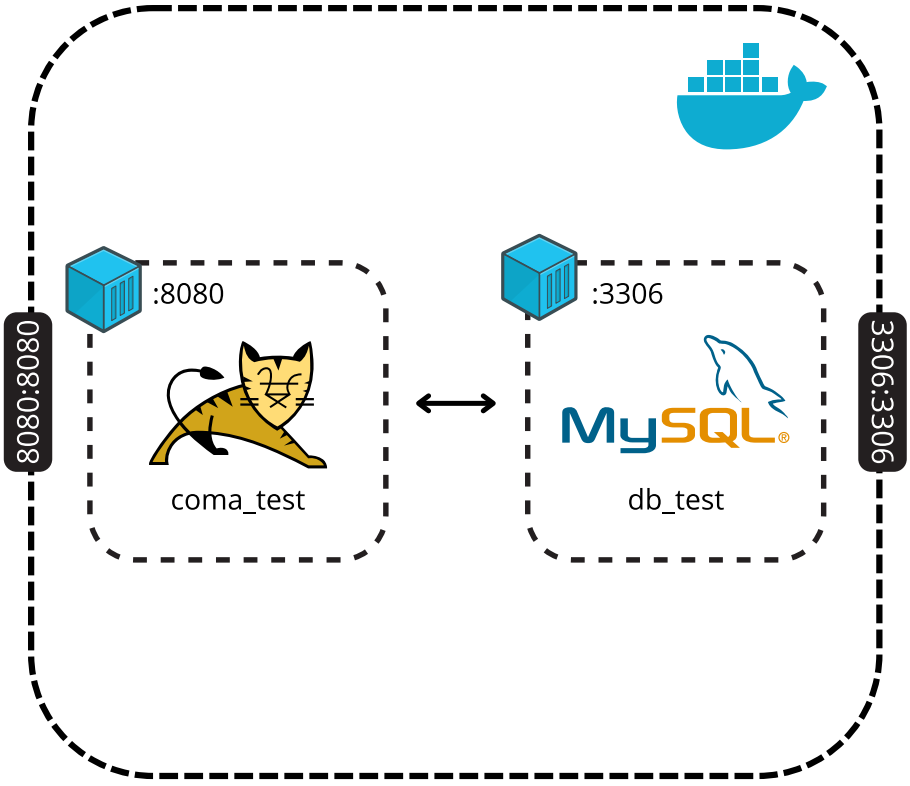
Para crear los contenedores, se decidió utilizar Docker Compose, una herramienta que permite definir y correr aplicaciones con varios contenedores. Además, simplifica el control de toda la aplicación, facilitando la administración de servicios, redes y volúmenes en un archivo de configuración de tipo YAML, el cual, con un solo comando, inicia los servicios descritos en este (*Docker Compose*, s.f.).

Para este prototipo se creó un contenedor de MySQL y un contenedor de Tomcat. Para que ambos contenedores puedan conectarse únicamente entre sí, se creó una red de Docker específicamente para COMA. En este sentido, si se crea algún contenedor que no esté dentro de dicha red, no podrá acceder a estos contenedores. Las redes de Docker tienen su propio DNS interno, por lo que, para la comunicación entre contenedores, se utiliza el nombre del contenedor como alias. Es decir, para acceder a la base de datos desde COMA, no se utiliza la IP asignada, sino el nombre del contenedor de la base de datos en el bloque de código donde se guarda la dirección de acceso. En cuanto a las bases de datos, se importaron una copia de *diamante* y *poseidon* en el contenedor de MySQL.

Cuando se compila una aplicación web en Java, este genera un archivo con extensión WAR que contiene todo el proyecto compilado. Este archivo, cuando se agrega a la carpeta webapps de Tomcat, automáticamente realiza el despliegue de la aplicación. Para este prototipo, se añadió el proyecto compilado en un volumen de Docker dentro de la carpeta webapps. La Figura 6 muestra la arquitectura del este primer prototipo.

Figura 6

Prototipo inicial de COMA en Docker



Este prototipo no requirió de cambios significativos en COMA, a excepción del host donde están alojadas las bases de datos. Se comprueba la ejecución del proyecto ingresando en un navegador a la IP donde se esté ejecutando Docker (en este caso, localhost), en el puerto 8080, y agregando el contexto /eisi en la URL.

5.3.2 Segundo prototipo: creación de dos portales web

Tras adaptar COMA en Docker, surgió la necesidad de agregar un segundo portal web en la arquitectura. Durante el inicio de este prototipo, se tienen dos consideraciones:

- Se decide crear un contenedor de MySQL exclusivo para *poseidon*, considerando que todos los portales web acceden a esta base de datos.

- Se plantea que cada portal web tendrá su propio contenedor de MySQL paralelo, el cual va a contener las bases de datos *diamante* y *division*. Esto se hace con el objetivo de aislar las bases de datos y facilitar la diferenciación por escuela.

Se identificó un problema durante la construcción del prototipo. En COMA hay tres clases de Java que tienen información diferente para cada organización:

- Parametrizacion.java: Almacena en una lista parámetros esenciales para la plataforma, como el nombre de la escuela, identificadores de programas académicos, correos electrónicos asociados a cada portal, entre otros.
- SuperParametrizacion.java: Contiene los datos necesarios para conectar con las bases de datos *diamante* y *poseidon*, como el usuario, host, puerto, nombre de la base de datos y credenciales de acceso.
- ConexionesDivisionServidor.java: Gestiona la conexión y las consultas hacia la base de datos *division*, incluyendo los parámetros de conexión correspondientes.

De permanecer estas diferencias, cada portal web tendría una base de código diferente, lo que lo que complica la gestión de versiones y las actualizaciones. Además, sería necesario garantizar que los valores específicos no se modifiquen accidentalmente durante una actualización. Para abordar este problema, se optó utilizar variables de entorno, lo que permiten crear una configuración flexible y reutilizable para los diferentes portales de COMA. Estas variables son declaradas en el archivo de configuración de Docker Compose, utilizado para la creación de los contenedores de Tomcat.

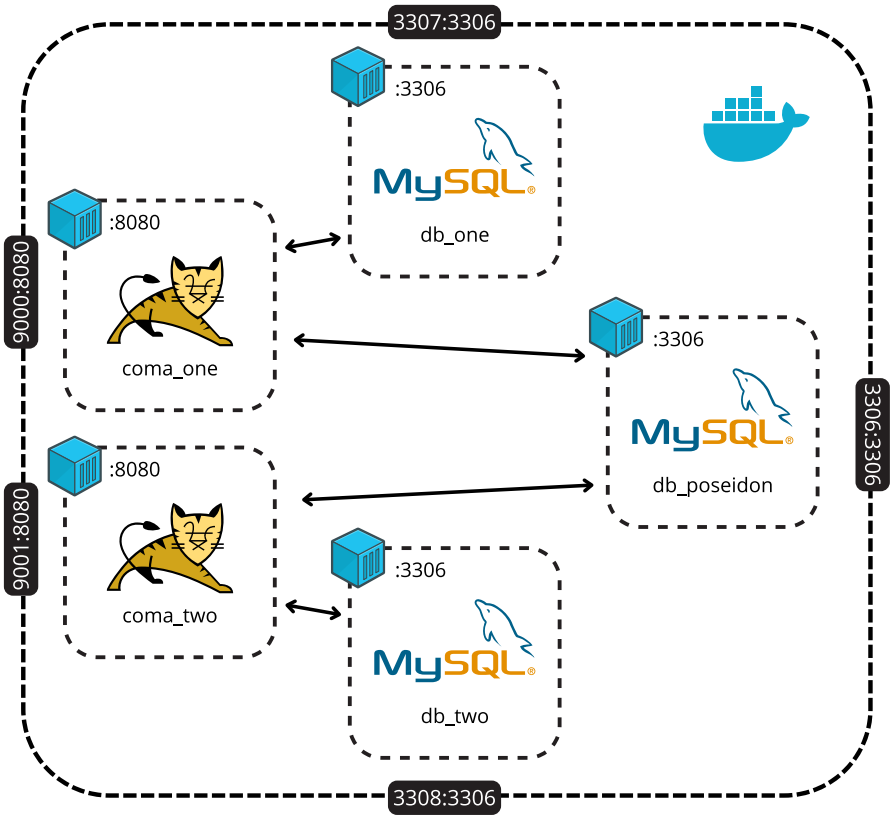
Las clases mencionadas fueron adaptadas para obtener los parámetros necesarios directamente de las variables de entorno, en lugar de utilizar cadenas de texto estáticas. Además, para mantener compatibilidad con entornos de desarrollo locales, se mantienen los valores

estáticos dado el caso que dicha variable no exista. Esta solución simplifica la gestión de los portales, reduce el riesgo de errores durante actualizaciones y mantiene compatibilidad con entornos de desarrollo.

En la Figura 7 se observa la arquitectura definida para este prototipo.

Figura 7

Segundo prototipo de COMA en Docker



Cada contenedor de Tomcat tiene expuesto un puerto diferente en el host para poder acceder a ambos portales de manera independiente. Además, para poder acceder las bases de datos, también se exponen puertos a cada contenedor de las bases de datos.

5.3.3 Tercer prototipo: simulación de dos portales web completos

Para este prototipo, la Escuela de Ingeniería de Sistemas e Informática (EISI) de la Universidad Industrial de Santander (UIS) entregó el nuevo servidor al grupo CALUMET. Además, en respuesta a una solicitud por del grupo, la División de Tecnologías de la Información y la Comunicación (DTIC) de la universidad asignó dos alias de prueba que apuntan al nuevo servidor:

- eisidev01.uis.edu.co
- eisidev02.uis.edu.co

A partir de esto, se realizó una simulación de dos portales web completos correspondientes a las escuelas de Ingeniería de Petróleos y Geología, en el nuevo servidor.

Para la simulación de estos portales, se replicaron archivos, imágenes colores y otros elementos característicos de ambas escuelas para verificar su acceso y funcionamiento. Adicionalmente, se realizaron cambios en COMA frente a como se obtienen los colores de cada portal, pues estos estaban distribuidos en varios archivos; ahora, existe un único archivo CSS que guarda los colores de los portales en variables globales que pueden ser accedidas por los demás CSS, lo cual facilita su mantenimiento y edición.

En términos de arquitectura, se definieron varios cambios importantes. En primer lugar, se decidió utilizar un único contenedor de MySQL para las bases de datos de las escuelas pertenecientes a una facultad, en lugar de asignar un contenedor a cada organización académica. Esta decisión busca optimizar recursos y simplificar la administración. Como parte de este cambio, se modificaron los nombres de las bases de datos para incluir el nombre corto de cada escuela. Por ejemplo, la base de datos *diamante* pasó a denominarse *diamante_corto*, y la base de datos *división*

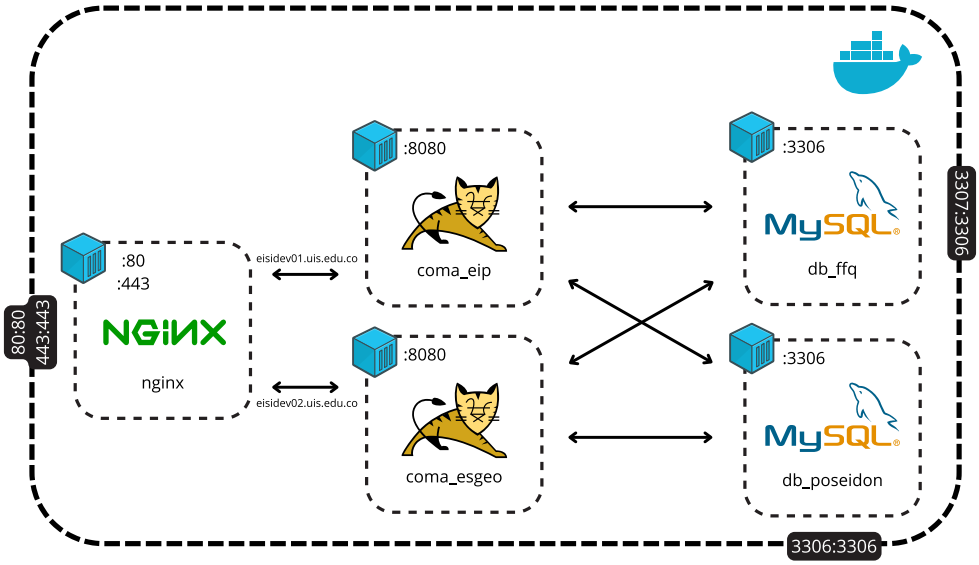
fue renombrada como *división_corto*. Estos cambios implican almacenar los nombres de las bases de datos en variables de entorno para mantener una misma base de código.

Además, se creó un contenedor de Nginx que funciona como reverse proxy. A partir de un archivo de configuración en Nginx, se definen los alias que apuntan al servidor, junto a donde se espera redirigir dicho alias. Como este contenedor fue creado dentro de la red de COMA en Docker, basta con agregar el nombre del contenedor destino. Así, Nginx se encarga de redirigir las solicitudes realizadas desde ciertas URL (o alias) hacia los contenedores correspondientes definidos en los archivos de configuración de cada portal. El alias eisidev01.uis.edu.co redirigió a petróleos, mientras que eisidev02.uis.edu.co a geología.

Finalmente, en este prototipo se implementó una mejora clave en los portales: la incorporación de certificados SSL/TLS para garantizar la seguridad en las conexiones. Estos certificados permiten establecer una comunicación encriptada entre los clientes y el servidor, protegiendo la transferencia de datos sensibles y aumentando la confianza de los usuarios en la plataforma. Para generar y gestionar los certificados, se utilizó un contenedor llamado *certbot*, el cual permite gestionar certificados otorgados por la organización Let’s Encrypt. Gracias a certbot es posible automatizar el proceso de creación, renovación y configuración de los certificados, además de su fácil integración con Nginx y Docker. Este contenedor no es permanente; se crea cuando es necesario (es decir, para la creación o renovación de certificados), y se elimina después de su uso. Su configuración está en un archivo de Docker Compose, el cual es llamado cada 12 horas para intentar renovar los certificados, según las recomendaciones de Let’s Encrypt. Los certificados solo se renovarán cuándo el tiempo de expiración sea menor a 30 días. En la Figura 8 se observa la arquitectura propuesta en este prototipo.

Figura 8

Tercer prototipo de COMA en Docker



5.3.4 Cuarto prototipo: arquitectura final

5.3.4.1 Finalización del prototipo

Como prototipo final, se adoptó la arquitectura previamente descrita, añadiendo un contenedor adicional. En el módulo de aula virtual de COMA existe un chat en tiempo real, el cual utiliza un framework creado en CALUMET llamado Realtime. Este utiliza Node.js y estaba instalado en cada servidor de Azure para ejecutar este servicio.

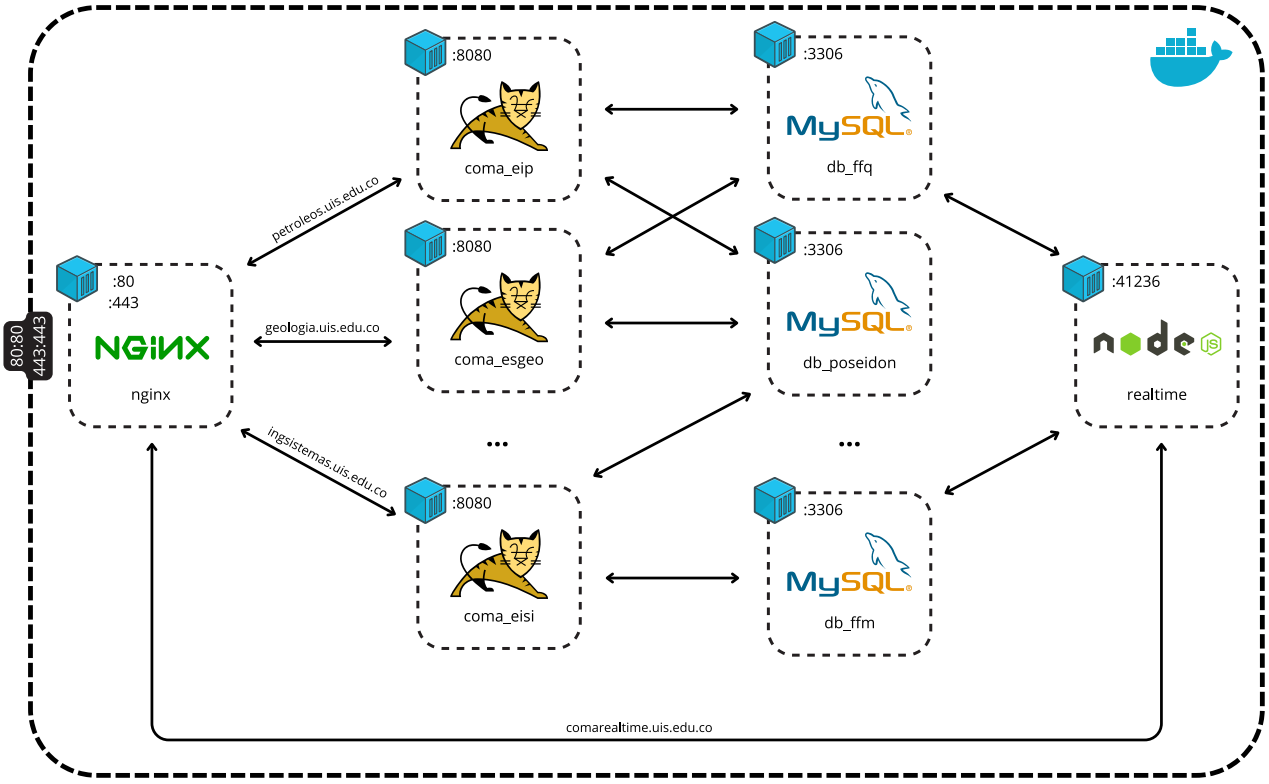
Para integrar este módulo en la nueva arquitectura, se decidió crear un único contenedor dedicado exclusivamente a responder solicitudes del chat de todas las escuelas. Esto implicó cambios en el código de Realtime para que pueda recibir y distinguir las solicitudes de todas las escuelas, además de guardar las conexiones a las bases de datos. Como está en constante comunicación con la base de datos *diamante* de cada escuela, el nuevo contenedor fue integrado dentro de la red de COMA en Docker. La decisión de no instalar Realtime en cada contenedor reduce el uso de recursos en cada portal.

Adicionalmente, como método de seguridad, se configuró un usuario para el acceso a la base de datos desde la aplicación web con permisos limitados a SELECT, INSERT, UPDATE, DELETE, SHOW y LOCK VIEW, siendo los dos últimos necesarios para la creación de copias de las bases de datos de manera automática.

Una vez definida la arquitectura completa, se llevó a cabo una prueba piloto de todos los portales web en el nuevo servidor, donde se realizaron copias de las bases de datos y los archivos (incluyendo imágenes y otros recursos) de cada portal web para posteriormente desplegar los treinta y dos portales simultáneamente. Esta prueba tuvo una duración aproximada de una semana, durante la cual se monitoreó constantemente el comportamiento de la plataforma y el consumo de recursos. Tras completar la prueba, se determinó que la arquitectura era estable.

Figura 9

Arquitectura final de COMA en Docker



5.3.4.2 Herramienta de Administración

En el grupo CALUMET existía una herramienta de administración llamada *Cóndor*, desarrollada en Node.js, versión 0.10. Esta herramienta permitía ejecutar acciones en los servidores de cada escuela por medio de un comando, utilizando dos archivos de configuración. El primero contiene una lista de las escuelas, junto con todos sus datos necesarios, para posteriormente acceder a ellas mediante SSH. El segundo se usa como archivo de configuración y en él se definen las acciones a ejecutar. En este archivo se configuran las rutas de origen y el destino, es decir, la ruta local y remota donde está instalado el proyecto para subir archivos; la lista de escuelas a las que se aplicarán las acciones y las acciones específicas, detallando las rutas de los archivos desde el directorio de instalación, comandos y queries que se usarán según corresponda.

Las acciones que podía realizar *Cóndor* incluyen:

- Ejecutar queries en las bases de datos *diamante* y *poseidon*.
- Crear copias de seguridad de la base de datos *diamante*, *poseidon* y *division*.
- Ejecutar comandos de Linux en los servidores de cada escuela.
- Copiar y eliminar archivos en el servidor de cada escuela.

En la Figura 10 se muestra un ejemplo del archivo de configuración principal. En él se aprecian las diferentes secciones separadas por comentarios y con una palaba clave al inicio para reconocer la acción.

Figura 10

Archivo de configuración principal de Cóndor

```
# ------------------------------------------------------------------------ #

DESTINATION /datadrive/tomcat/webapps/eisi/
ORIGIN /home/romel/projects/eisi/build/web/
MYSQL_DIAMANTE_BACKUP true
MYSQL_POSEIDON_BACKUP false

# ------------------------------------------------------------------------ #

SERVERS

sistemas
civil

# ------------------------------------------------------------------------ #

COMMANDS

ls -l /datadrive/tomcat/webapps/eisi


# ------------------------------------------------------------------------ #

MYSQL_DIAMANTE_QUERIES

INSERT INTO TB_FileManager
  (Codigo, Ruta, Lectura, Agregar, ModificarBorrar, SubExploracion)
  VALUES ("1234", "/eisi/Calumet/temp/", 1, 0, 0);
# ------------------------------------------------------------------------ #

MYSQL_POSEIDON_QUERIES

#
# ------------------------------------------------------------------------ #

DELETE_FILES

/Scripts/Estandar/Elise/


# ------------------------------------------------------------------------ #

ADD_FILES

/WEB-INF/classes/muisca/acreditacion/ModeloAcreditacion.class
```

La forma en que se realizaban actualizaciones al sistema consistía en añadir uno a uno la ruta de los archivos que han sido modificados en la sección de *ADD_FILES*. Esto es un proceso tedioso y propenso a errores, el cual generó varias discrepancias en las versiones desplegadas de cada portal web, donde algunos estaban desactualizados por problemas de actualizaciones, o porque dichos portales fueron apagados temporalmente. Además, configurar todas las acciones desde un mismo archivo puede llevar a confusiones sobre donde se agrega cada elemento.

Con la migración a la nueva arquitectura con contenedores, el software *Cóndor* quedó obsoleto. Por esta razón, se planteó rediseñar la herramienta adaptándola a la arquitectura Docker, mejorando y ampliando las funcionalidades del sistema anterior. Así nació el software *Súper*

Cóndor, que conserva ciertas similitudes con su predecesor, pero se ha modernizado significativamente.

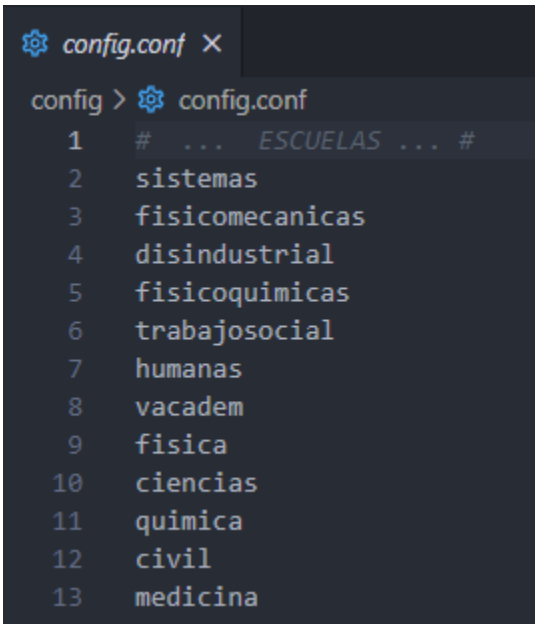
El nuevo proyecto, desarrollado en Node.js actualizado a la versión 20 (LTS), utiliza tres archivos de configuración principales, cambiando el método de acceso al servidor. Ahora se utiliza un par de llaves SSH, privada y pública, generadas mediante un comando, con la posibilidad de agregar una contraseña a la misma. La llave pública se almacena dentro del servidor, mientras que la privada se utiliza para acceder al servidor de manera segura. Este enfoque no solo limita el número de equipos que pueden acceder al servidor, sino que también agrega una capa adicional de seguridad al requerir tanto la clave privada como la contraseña para establecer la conexión.

El primer archivo contiene una lista en formato JSON con todos los contenedores y los datos necesarios para su correcto uso. Estos datos incluyen nombre clave (nombre del portal web abreviado), nombre del contenedor, nombre del contenedor de base de datos, nombres de los esquemas de las bases de datos y contraseñas de base de datos.

El segundo archivo contiene una lista con los nombres clave de cada contenedor. A partir de esta lista, se seleccionan las escuelas a las que se aplicarán las acciones. En la Figura 11 se observa un fragmento del archivo, el cual contiene un total de treinta y dos nombres clave. Para seleccionar la escuela a ejecutar la acción se debe dejar sin comentar el nombre clave de dicho portal.

Figura 11

Archivo de selección de escuelas en Super Cóndor



Por último, el archivo de configuración de rutas que contiene la contraseña y la ruta de la clave privada necesaria para acceder al servidor y ejecutar acciones en cada contenedor, también define las rutas de los archivos que se deben enviar o eliminar y las rutas donde se almacenan los backups de forma local.

Súper Cóndor ya no utiliza un único comando para ejecutar múltiples acciones; fue desarrollado para realizar una tarea específica según el comando que se ingrese. El archivo principal del proyecto redirige a las diferentes funciones según el comando recibido. Entre las funciones disponibles se incluyen las siguientes:

- Ejecuta una conexión de prueba para verificar que el programa funcione y el estado de los archivos de configuración.
- Despliegue del proyecto en el contenedor de pruebas.
- Despliegue del proyecto en los contenedores de los portales en producción.
- Enviar archivos de mantenimiento a los contenedores, incluyendo los crontab.
- Ejecuta script SQL a las bases de datos diamante, *division* y *poseidon*.

- Realizar una copia de seguridad de las bases de datos diamante, *division* y *poseidon*.
- Realizar una copia de seguridad de los archivos de cada contenedor.
- Copiar o eliminar archivos en cada contenedor.
- Reinicia los contenedores seleccionados.
- Ejecutar comandos de Linux en todos los contenedores.

Aprovechando la centralización de todos los portales web, se identificó una oportunidad para simplificar el despliegue de actualizaciones y mejoras. En lugar de listar uno a uno los archivos modificados o agregados, se propuso subir todo el proyecto compilado y, a partir de este, realizar el despliegue. Sin embargo, no fue posible utilizar la funcionalidad de Tomcat para desplegar automáticamente nuevas versiones al sustituir el archivo del proyecto compilado, ya que este proceso elimina las carpetas de imágenes y archivos específicas de cada escuela, almacenadas en Tomcat. Este comportamiento se debe a que, como se mencionó anteriormente, la carpeta *webapps* solo debe contener el código base de la aplicación web; en este sentido, Tomcat hace un despliegue total de la aplicación web, eliminando aquellos elementos que no pertenecen al proyecto. Para solucionar este problema, se decidió alojar el proyecto compilado fuera de Tomcat y utilizar *Súper Cóndor* para descomprimir el proyecto y agregar o reemplazar los archivos pertenecientes al código base de la aplicación web dentro de la carpeta webapps.

La plataforma GitHub, a través de la herramienta GitHub Actions, facilita la automatización de tareas de integración y entrega continua a partir de flujos de trabajo personalizados, también llamados *workflow,* definidos en archivos de tipo YAML. Estos flujos responden a eventos descritos el workflow como, por ejemplo, un push a una rama específica. Son ejecutados en una máquina llamada *runner*, encargada exclusivamente de ejecutar un flujo de trabajo de GitHub Actions. Hay dos tipos de runners: los proporcionados por GitHub y los *self-*

hosted runners. Estos últimos son máquinas propias que se vinculan a un repositorio mediante un token, permitiendo ejecutar los workflow en un servidor propio y superar las limitaciones de los recursos ofrecidos por GitHub. Para integrar esta funcionalidad con la nueva arquitectura de COMA, se creó un contenedor basado en Ubuntu 24.04 que aloja la aplicación *GitHub Actions Runner*, el cual fue vinculado con el repositorio que contiene el código fuente de COMA.

Con base en lo anterior, se crearon tres workflow. El primero se activa ante cambios en la rama *master*, indicando que hay modificaciones para ser desplegadas en producción. Este workflow compila el proyecto y almacena el archivo resultante (el proyecto compilado de tipo WAR) en un directorio específico del nuevo servidor, compartido con los contenedores de Tomcat para el despliegue la aplicación web (ver Apéndice A). El segundo también compila el proyecto, sin embargo, se ejecuta cuando es llamado manualmente por algún desarrollador sobre una rama específica (ver Apéndice B). Esto permite probar funcionalidades en un entorno de desarrollo aislado dentro la nueva arquitectura. Para ello, se crea un contenedor de prueba, similar a los de producción, el cual contiene sus propias bases de datos con registros modificados. Por último, debido a que los *self-hosted runners* son necesitan tener al menos una actividad cada catorce días para no ser desvinculados del repositorio, se creó un tercer workflow que se ejecuta cada seis días, encargado de mantener el runner activo (ver Apéndice C).

Una vez compilado el proyecto, el despliegue de los nuevos cambios se realiza mediante un comando ejecutado desde *Súper Cóndor*. Este comando descomprime el proyecto en cada contenedor y los reinicia para que Tomcat despliegue la aplicación web, preservando los archivos específicos de cada portal. Sin embargo, debido a que el proyecto compilado se descomprime sobre la versión existente, surge una limitación: los archivos eliminados del código fuente no son eliminados de los contenedores. Para solucionar este inconveniente, se añadió a *Súper Cóndor* una

funcionalidad que permite eliminar los archivos o directorios especificados en un archivo de configuración. Como son pocas las ocasiones en que se eliminan archivos, la tarea del despliegue de nuevas versiones de COMA se simplificó significativamente.

5.3.5 Pruebas y Validación

Para validar las mejoras de la nueva arquitectura frente a la anterior, se decidió realizar pruebas en uno de los servidores de Azure en producción, específicamente en el de la Escuela de Ingeniería de Sistemas e Informática, así como en un contenedor de prueba del nuevo servidor. No se llevarán a cabo pruebas de estrés en el servidor de Azure, ya que este presenta una limitación en el número de peticiones permitidas por minuto. La Tabla 2 muestra el plan de pruebas planteado.

Tabla 2

Plan de pruebas JMeter

Servidor	Tipo de prueba	Peticiones/Tiempo
Servidor Azure	Carga	100 peticiones/seg.
	Resistencia	1 petición/5 seg., por 24 horas
Nuevo Servidor	Carga	100 peticiones/seg.
	Estrés	500 peticiones/seg.
	Estrés	1000 peticiones/seg,
	Resistencia	1 petición/5 seg., por 24 horas

Nota. En esta tabla se observa el plan de pruebas a realizar para los dos servidores, así como las peticiones a realizar o tiempo de duración de la prueba.

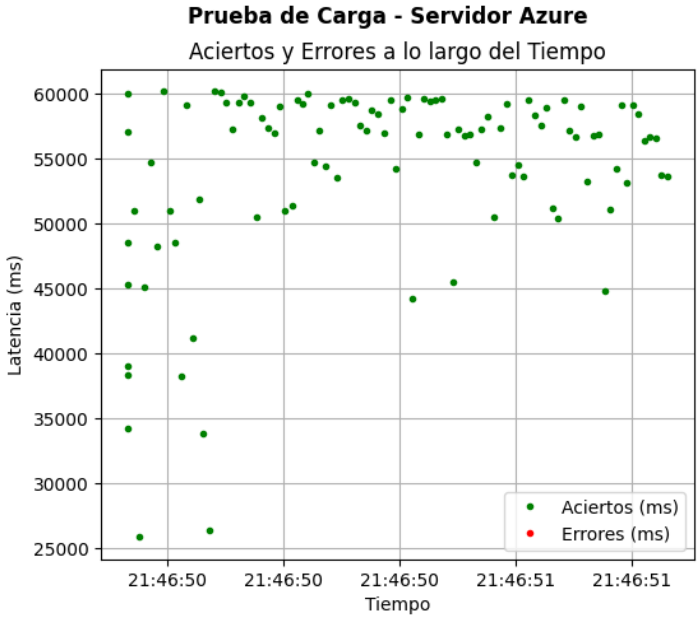
5.3.5.1 Pruebas Servidor Azure

5.3.5.1.1 Prueba de Carga 100 Peticiones. Los resultados obtenidos en esta prueba de carga muestran un porcentaje de error del 0%, lo que indica que todas las solicitudes se completaron exitosamente. No obstante, el tiempo de latencia promedio registrado fue de 54,220.26 ms, un valor bastante elevado. Este tiempo se debe a las limitaciones técnicas que tienen los servidores de Azure en cuanto a procesamiento y RAM.

En la Figura 12, se observan las peticiones acertadas y erradas con sus tiempos de latencia durante el tiempo de ejecución de la prueba. Se pueden apreciar una latencia muy alta, exceptuando algunos casos al inicio de la prueba.

Figura 12

Prueba de carga en el servidor de Azure



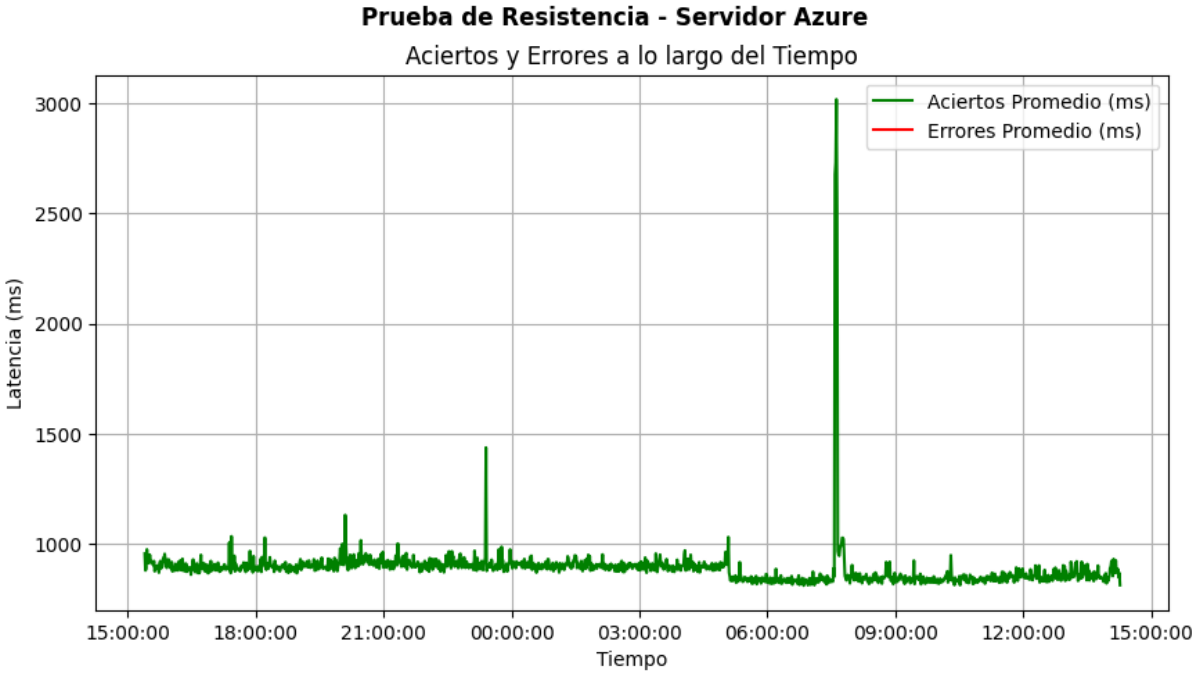
5.3.5.1.2 Prueba de Resistencia 24 Horas. Los resultados de esta prueba de resistencia muestran un porcentaje de errores del 0%. La latencia promedio fue de 884.23 ms, un rendimiento aceptable que indica una buena capacidad de manejo de resistencias bajo las condiciones

evaluadas. Este tiempo sugiere que el sistema puede gestionar de manera aceptable la carga actual sin problemas de latencia significativos, a excepción de ciertos valores atípicos.

En la Figura 13, se presenta una línea de tiempo de color verde que ilustra la latencia promedio de las solicitudes exitosas. Esta representación visual permite observar cómo varía la latencia a lo largo del periodo evaluado. En ella se pueden observar algunos picos, lo que indica que la arquitectura previa no era estable. Sin embargo, tiene un tiempo promedio de latencia aceptable.

Figura 13

Prueba de resistencia en el servidor de Azure



5.3.5.2 Pruebas Servidor Nuevo

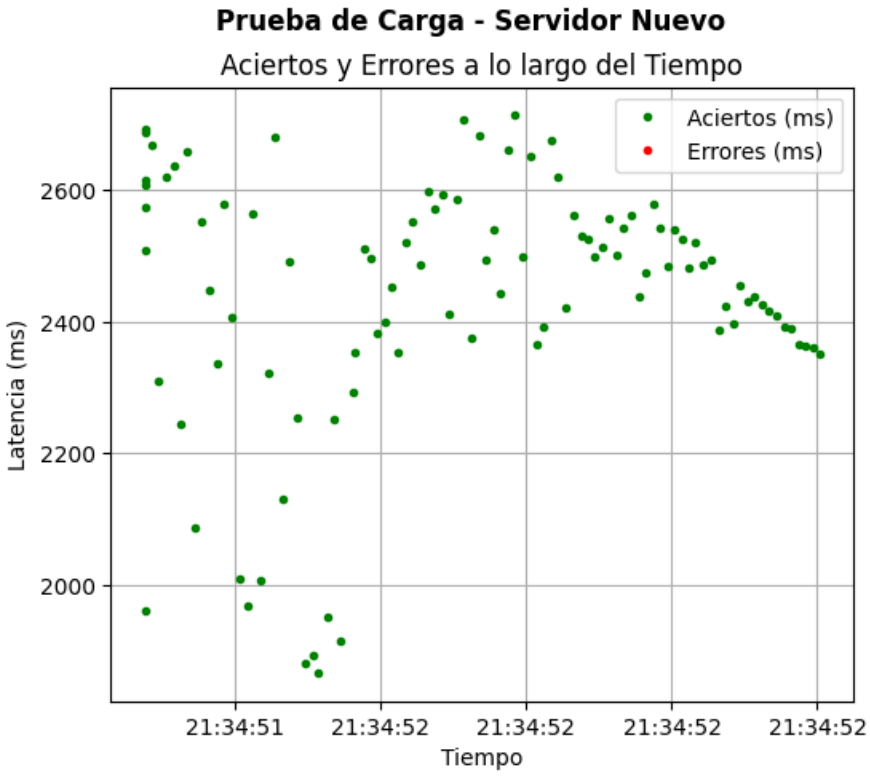
5.3.5.2.1 Prueba de Carga 100 Peticiones. Los resultados obtenidos en esta prueba de carga reflejan un porcentaje de errores del 0%. La latencia promedio fue de 2,435.63 ms, lo cual

es un valor excelente en comparación con los de Azure, y representa un buen rendimiento sin cuellos de botella o limitaciones.

En la Figura 14 se presentan las peticiones exitosas y fallidas, junto con la latencia durante la ejecución de la prueba. Se puede observar que no hubo ningún fallo en las solicitudes y que los valores de latencia no están tan dispersos en comparación con la prueba realizada en el servidor de Azure. A pesar de esto, se mantiene la tendencia de obtener mejores tiempos al inicio de la prueba.

Figura 14

Pruebas de carga en el nuevo servidor



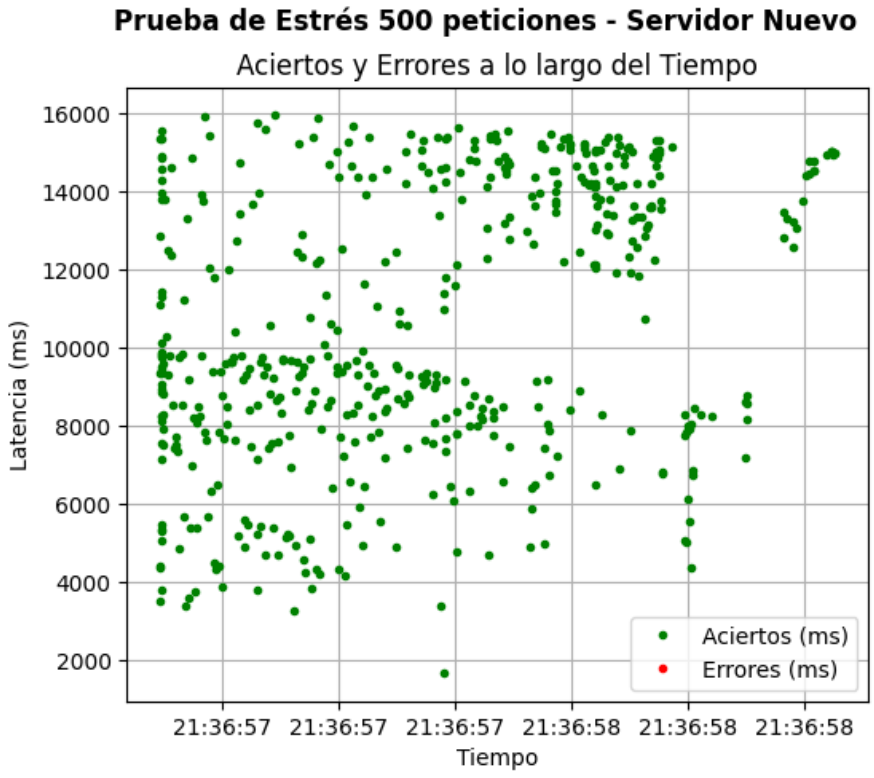
5.3.5.2.2 Prueba de Estrés 500 Peticiones. Los resultados obtenidos en esta prueba de estrés reflejan un porcentaje de errores del 0%. La latencia promedio fue de 10,602.40 ms, lo cual

es un valor bastante bueno teniendo en cuenta la cantidad de peticiones simultáneas en el periodo de tiempo de la prueba, que fue de 2 segundos.

En la Figura 15, se observan las peticiones acertadas y erradas con su latencia durante el tiempo de ejecución de la prueba. En ella se puede observar el mayor volumen de solicitudes, y aunque la latencia osciló en un rango más amplio en comparación con la anterior, se mantuvo un tiempo promedio adecuado, sin ningún fallo en las solicitudes, lo que indica una latencia excelente bajo situaciones de estrés.

Figura 15

Pruebas de estrés de 500 peticiones en el nuevo servidor

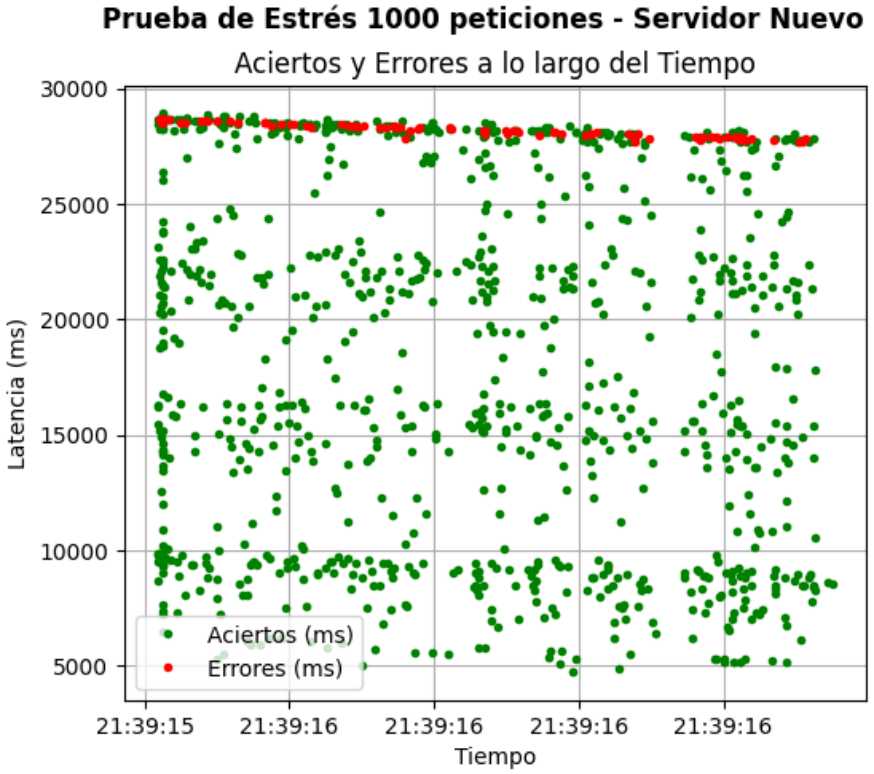


5.3.5.2.3 Prueba de Estrés 1000 Peticiones. La latencia promedio de esta prueba de estrés fue de 17,869.92 ms, lo cual es un valor aceptable teniendo en cuenta la cantidad de peticiones simultáneas realizadas. Sin embargo, el porcentaje de error fue del 9.2%.

En la Figura 16 se observan las peticiones exitosas y fallidas, junto con la latencia durante la ejecución de la prueba. Se aprecia el aumento en el número de peticiones, así como las peticiones fallidas, representadas en color rojo. A pesar de los fallos obtenidos, el tiempo de latencia se mantuvo en un rango aceptable, considerando que se duplico la cantidad de solicitudes realizadas en un periodo de 2 segundos con respecto a la prueba anterior, esto indica que el servidor responde bien en condiciones de estrés alto, sin embargo, puede fallar.

Figura 16

Pruebas de estrés de 1000 peticiones en el nuevo servidor



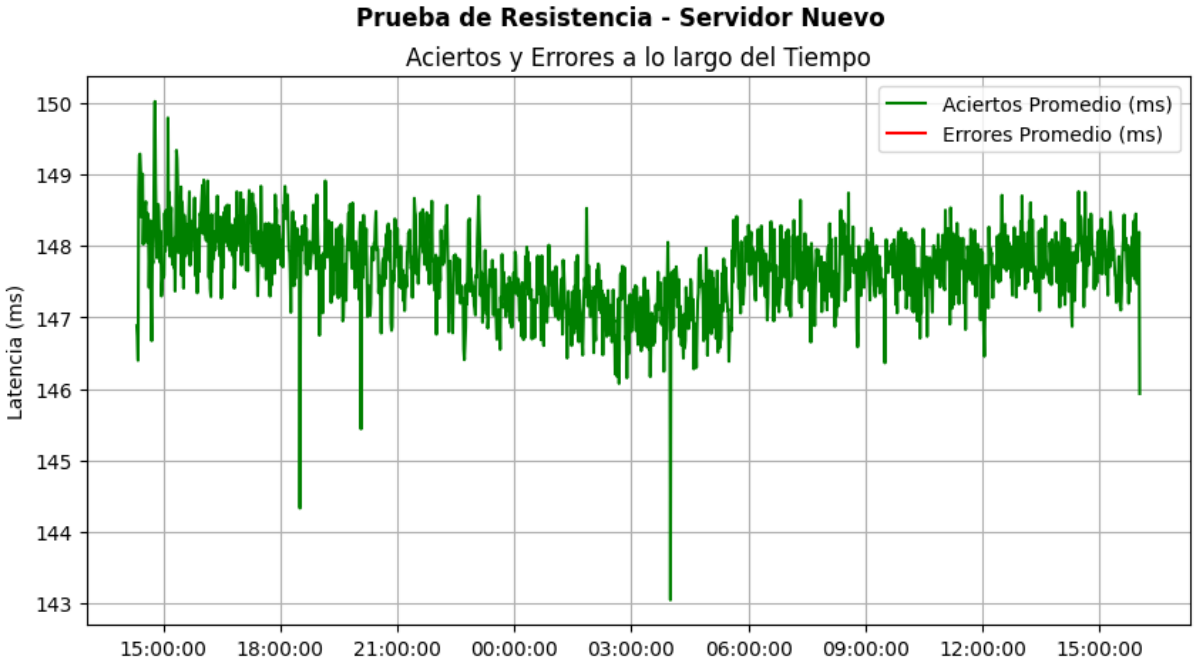
5.3.5.2.4 Prueba de Resistencia 24 Horas. Los resultados de esta prueba de resistencia indican un porcentaje de error del 0%, lo cual confirma que todas las solicitudes fueron procesadas correctamente. La latencia promedio alcanzada fue de 147.66 ms, un valor que sugiere que el sistema puede gestionar eficientemente sin problemas significativos de latencia, lo cual es una

señal positiva en términos de escalabilidad y eficiencia al responder a múltiples solicitudes simultáneas.

En la Figura 17 se presenta una línea de tiempo de color verde que ilustra la latencia promedio de las solicitudes exitosas. Se puede observar que, durante el periodo de prueba, el servidor experimentó solo picos mínimos de latencia y se mantuvo cerca del tiempo promedio, lo que representa un rendimiento notable y una mejora significativa en comparación con el servidor de Azure.

Figura 17

Prueba de resistencia en el nuevo servidor



5.3.5.3 Comparación y Análisis de Resultados

5.3.5.3.1 Comparación de Latencia. En la Tabla 3 se presentan los resultados de las pruebas realizadas a los dos servidores. Los datos muestran que el servidor nuevo se destaca notablemente en latencia, comparado con el servidor de Azure. En las pruebas de carga con 100

solicitudes simultáneas, el servidor nuevo logró resultados superiores a los obtenidos por el servidor de Azure en la misma prueba. Además, incluso en las pruebas de estrés, el servidor nuevo demostró un rendimiento mejor que el servidor de Azure en las pruebas de carga.

En cuanto a las pruebas de resistencia, se observó que ambos servidores mantuvieron un comportamiento estable, sin presentar fallos durante la ejecución. Sin embargo, las diferencias en latencia fueron significativas, con el servidor nuevo mostrando una superioridad considerable frente al servidor de Azure con 750 ms, aproximadamente, de diferencia.

A pesar de los excelentes resultados obtenidos en las pruebas de estrés para la prueba de 1000 peticiones, el nuevo servidor presentó errores en las peticiones. Esto se debe a que, de acuerdo a como está programado la conexión de las bases de datos en el backend, hay dos posibilidades: cada consulta crea una nueva conexión a MySQL, o una misma conexión se utiliza para un fragmento de código; el uso de cada una es arbitrario. Cada conexión utiliza un puerto efímero del contenedor. El propósito de dichos puertos es permitir que varias aplicaciones o clientes establezcan conexiones simultáneas sin interferencia entre ellas mismas (Coursera, 2023). En este sentido, al existir tantas conexiones simultáneas, los puertos efímeros son todos ocupados durante un instante de tiempo, por lo que algunas peticiones de esta prueba de estrés fallan. Sin embargo, en las peticiones acertadas de la prueba, el promedio de latencia fue bastante bueno con 17,869.92 ms, superando también los valores de las pruebas de carga en el servidor de Azure.

Tabla 3

Resultados plan de pruebas

Servidor	**Tipo de prueba**	**Peticiones/Tiempo**	**Tiempo promedio de latencia (ms)**	**Porcentaje de error**

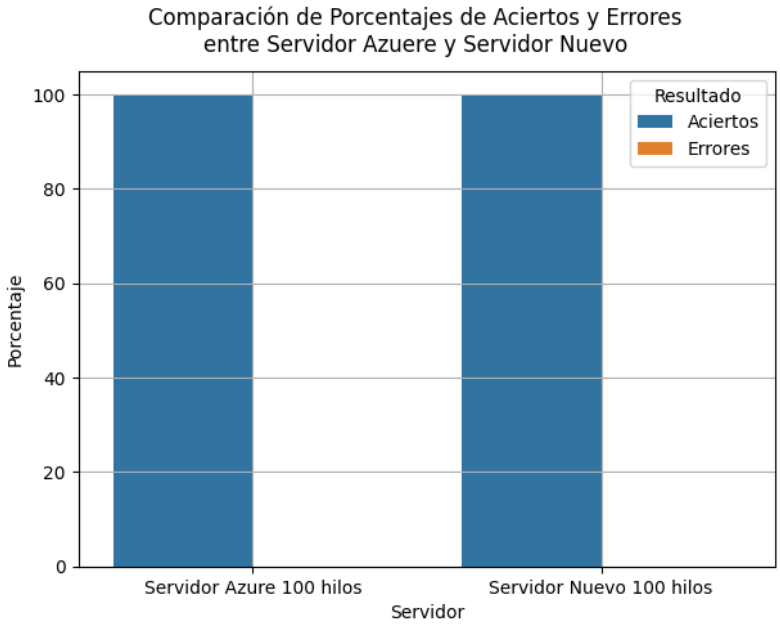
Servidor Azure	Carga	100 peticiones/seg	54,220.26 ms	0%
	Resistencia	1 petición/5seg, por 24 horas	884.23 ms	0%
Nuevo Servidor	Carga	100 peticiones/seg	2,435.63 ms	0%
	Estrés	500 peticiones/seg	10,602.40 ms	0%
	Estrés	1000 peticiones/seg	17,869.92 ms	9.2%
	Resistencia	1 petición/5seg, por 24 horas	147.66 ms	0%

Nota. En esta tabla se observan los resultados del plan de pruebas realizado para los dos servidores con su tiempo promedio de latencia y el porcentaje de error de las peticiones.

5.3.5.3.2 Comparación de Porcentajes de Error. En la Figura 18 se muestra el porcentaje de aciertos y errores en las pruebas de carga realizadas a ambos servidores. Se puede observar que ninguno de los servidores presentó fallos en las peticiones. No obstante, se destaca una diferencia notable en los tiempos de latencia: el servidor nuevo fue aproximadamente 52,000 ms más rápido, lo que representa una mejora significativa en comparación con el servidor de Azure. Esta comparación solo se pudo hacer con las pruebas comunes realizadas en ambos servidores.

Figura 18

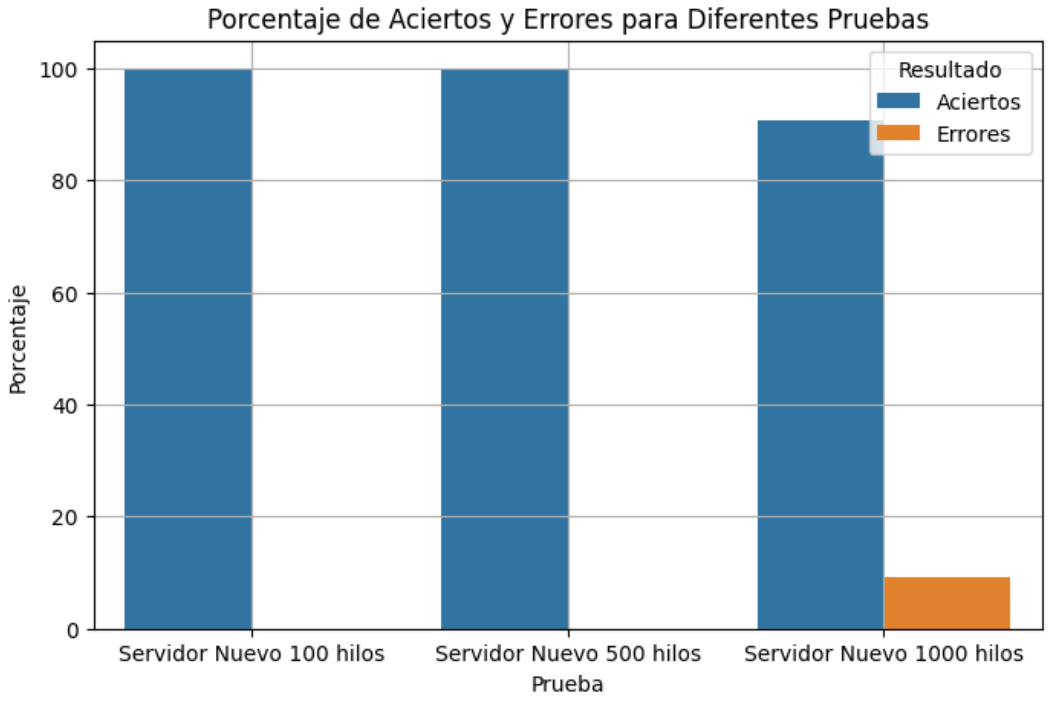
Comparación de porcentajes de error entre servidores



5.3.5.3.3 Comparación de Pruebas en Servidor Nuevo. En la Figura 19 se ilustra el porcentaje de aciertos y errores obtenidos en las distintas pruebas realizadas al nuevo servidor. Se puede observar que tanto las pruebas con 100 como con 500 solicitudes no presentaron fallos y lograron una latencia óptima, destacando la estabilidad y eficiencia de la arquitectura hasta ese punto de carga. Sin embargo, en la prueba de estrés con 1,000 solicitudes, se registró un porcentaje de error del 9.2%. Este resultado se atribuye a las limitaciones de arquitectura interna del proyecto y al problema previamente mencionado de los puertos efímeros, esto representan un desafío en términos de escalabilidad y rendimiento, especialmente bajo cargas extremas. Aunque la implementación en Docker y el nuevo servidor proporciona un mejor entorno, este tipo de limitaciones expone la necesidad de optimizar la gestión de conexiones y de considerar la implementación de un sistema más robusto en el código de COMA.

Figura 19

Comparación de porcentajes de error entre pruebas del nuevo servidor

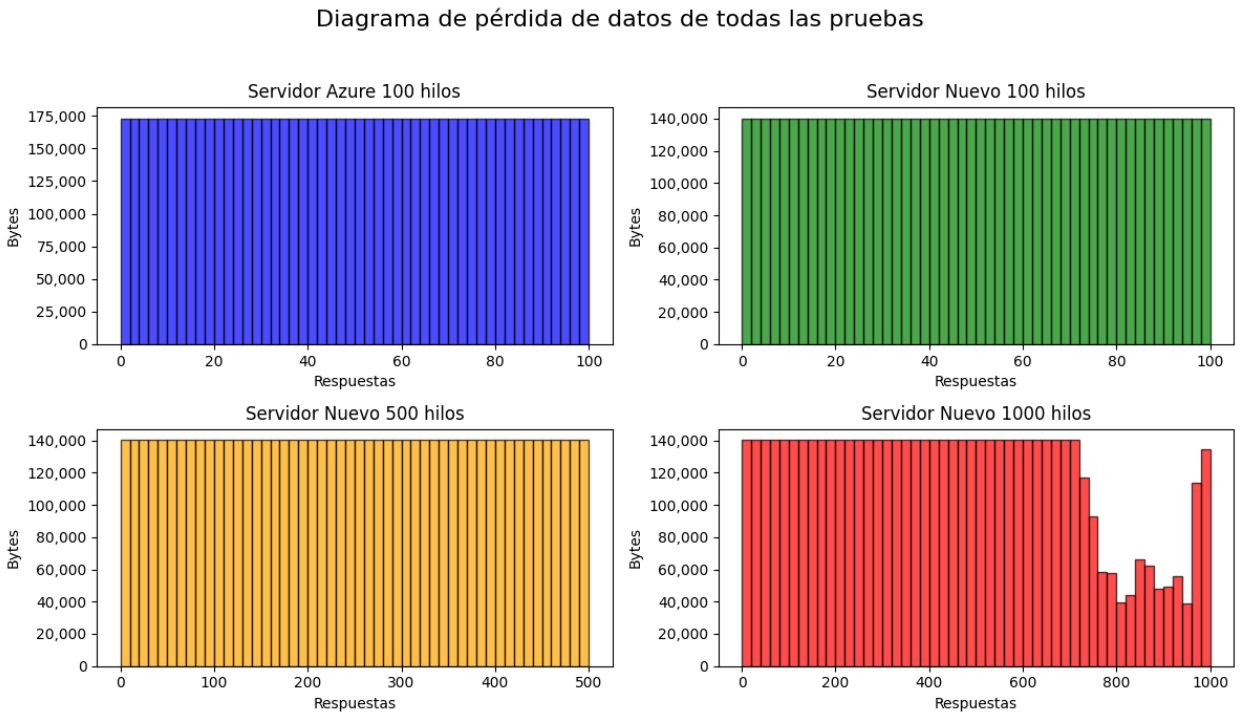


5.3.5.3.4 Diagrama de Pérdida de Datos. En la Figura 20 se muestra la cantidad de bytes transferidos durante las peticiones realizadas en las distintas pruebas. Se observa que, tanto en las pruebas de carga para ambos servidores como en la prueba de estrés con 500 solicitudes, no se presentó pérdida de datos, lo cual evidencia una buena capacidad de manejo de la carga dentro de estos escenarios. Sin embargo, en la prueba de estrés con 1,000 solicitudes, se detectó una pérdida de datos en algunas de las peticiones.

Esto se debe a las limitaciones en la arquitectura de la aplicación web, que, bajo una carga tan elevada, enfrenta dificultades para gestionar de manera eficiente el tráfico y el volumen de solicitudes simultáneas, específicamente con las conexiones a las bases de datos, como se menciona previamente.

Figura 20

Diagrama de pérdida de datos de todas las pruebas

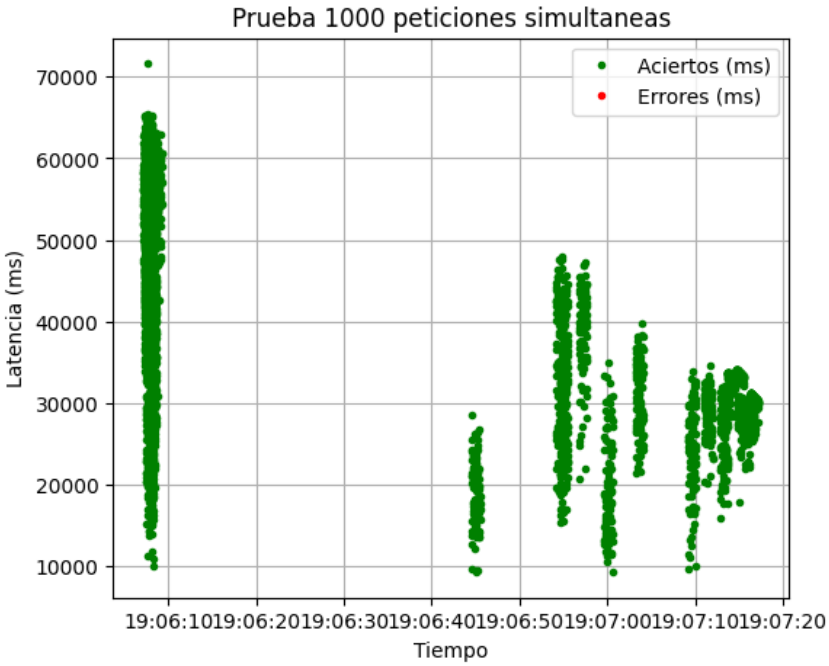


Nota. Los valores fueron promediados por intervalos para mejorar la visibilidad de la figura.

5.3.5.3.5 Prueba a Todos los Contenedores. Como adicional a las pruebas de estrés, se realizó un test enviando 100 peticiones simultáneamente a los 32 contenedores de prueba, cada uno representando a un portal web. Los resultados obtenidos se muestran en la Figura 21. La prueba arrojó un tiempo promedio de latencia de 37,000 ms y un porcentaje de error del 0%, lo que demuestra que el servidor mantiene una latencia aceptable, incluso bajo condiciones de alta carga en todos sus contenedores, sin presentar fallas en las peticiones.

Figura 21

Prueba simultánea en todos los contenedores



Después de realizadas las diversas pruebas en el servidor nuevo, se agregan las mismas limitaciones de peticiones por minuto presentes en los servidores de Azure para evitar ataques a la plataforma.

5.4 Fase de Migración

5.4.1 Despliegue en Producción

Para iniciar el despliegue, se restringió temporalmente el inicio de sesión en la plataforma para evitar modificaciones en las bases de datos, y no permitir que los usuarios suban nuevos elementos. Posteriormente, se realizaron copias de archivos y bases de datos en la arquitectura previa, similar a lo realizado en la prueba piloto, para ser transferidos a una ubicación específica del nuevo servidor. Una vez transferidos, se inicializaron todos los contenedores, cada uno con su propio archivo de configuración de tipo YAML, incluyendo los de las bases de datos y el servicio de Realtime. Además, se ejecutaron todos los procesos automáticos de COMA en los crontab.

La División de Tecnologías de la Información y la Comunicación (DTIC) colaboró en la reconfiguración de los alias de cada portal para que redirigieran al nuevo servidor, incluido los grupos y demás alias pertenecientes a COMA. Como preparación a esto, se finalizó la configuración de Nginx con la lista de todos los alias, para que, al momento de ser redirigidos al nuevo servidor, ya tengan un contenedor destino al que llegar. A partir de esto, se generaron los certificados SSL/TLS de todos los alias, agrupados en 6 certificados multidominio brindados por Let's Encrypt. Esta agrupación se hizo según la facultad a la que pertenece cada escuela, con la excepción de Fisicomecánicas que requirió de dos certificados diferentes debido a la cantidad de alias.

Tras generar los certificados, se realizó una verificación completa del funcionamiento de los portales. Finalmente, después de un día, se reactivó el inicio de sesión a los usuarios, notificándoles que la plataforma ahora está operando en el nuevo servidor.

La arquitectura en producción incluyó un total de treinta y dos portales distribuidos en 32 contendores de Tomcat, 6 contenedores de MySQL, un contenedor de Nginx y un contenedor de Node.js para Realtime.

5.4.2 Documentación

Se desarrolló un documento que describe en detalle los pasos necesarios para montar el nuevo esquema, agregar nuevos contenedores y realizar los despliegues en producción utilizando GitHub Actions y la herramienta Super Cóndor. Este documento incluye los comandos básicos de Docker para realizar el monitoreo de las escuelas, ingresar a los contenedores, reiniciarlos, descargar archivos y revisión de logs, así como el proceso para agregar nuevos dominios en NGINX, generar certificados y llevar a cabo otras tareas administrativas.

Además, en los repositorios correspondientes a la arquitectura y a Super Cóndor, se dejó una documentación detallada. En el caso de la arquitectura, se incluyó una descripción clara de su diseño y comandos básicos, mientras que para Super Cóndor se añadió una guía de instalación, configuración y uso de la herramienta, junto con una lista de comandos y una descripción detallada de la funcionalidad de cada uno.

Este documento fue entregado al director del grupo CALUMET y, tanto el código fuente, como la documentación de la arquitectura Docker y Super Cóndor, se encuentran alojados en repositorios privados del grupo CALUMET en GitHub.

6. Conclusiones

A lo largo de este proyecto se llevaron a cabo diversas acciones que permitieron abordar y resolver los desafíos identificados en la plataforma COMA, además de implementar soluciones que fortalecieron su infraestructura y funcionalidad. A continuación, se presentan las conclusiones de los resultados más destacados, evidenciando el impacto positivo de las mejoras realizadas.

Con las labores de mantenimiento de la plataforma COMA, se lograron resolver los errores reportados en los diferentes módulos y atender las mejoras solicitadas por las diversas escuelas. Esto facilita el desarrollo de las actividades misionales de las diferentes facultades y escuelas de la universidad, realizadas por los usuarios en la plataforma.

El algoritmo de actualización automática diseñado fue implementado en cada una de las escuelas cumpliendo satisfactoriamente con sus funciones. Este algoritmo permite mantener la paridad entre las bases de datos de las escuelas y la universidad, reduciendo significativamente el trabajo manual y la necesidad de monitoreos constantes por parte de los administradores de la plataforma. Además, previene errores humanos que podrían desencadenar problemas graves en las bases de datos de la plataforma.

La migración de los portales de las escuelas solucionó algunos problemas existentes en la plataforma durante su desarrollo e implementó una arquitectura más sólida y eficiente en comparación con la anterior. Esta nueva arquitectura es más escalable y permite realizar futuras mejoras con mayor facilidad, algo que no sería posible con la arquitectura previa. Como resultado, se logró una mejora significativa en los tiempos de respuesta y en la estabilidad de la plataforma permitiendo a los usuarios una mejor experiencia en el uso de la plataforma.

Por otro lado, se facilitaron las labores de mantenimiento mediante el desarrollo de *Super Cóndor*, una herramienta que realiza las mismas funciones que su predecesora, pero de manera más eficiente y con una ampliación de sus capacidades. Esta herramienta representa un avance en la gestión, seguridad y operación de la plataforma COMA, consolidándola como un recurso esencial para el cumplimiento de las metas del grupo CALUMET.

7. Trabajo a Futuro

Como parte del trabajo a futuro, se recomienda trasladar los archivos relacionados con “imágenes”, "profesores", "portales", "grupos" y "colores" a un directorio externo al de despliegue de Tomcat. Esta reestructuración permitiría realizar un despliegue de la aplicación completamente automatizado, simplificando el proceso y mejorando su implementación y mantenimiento.

Además, se sugiere implementar un pool de conexiones para abordar el problema de los puertos efímeros, lo cual contribuiría a una gestión más eficiente de las conexiones y evitaría la saturación de dichos puertos en cada contenedor, mejorando así la estabilidad de la aplicación.

La transición a la arquitectura Docker simplificaría la migración de la plataforma hacia un entorno de microservicios, ofreciendo mayor modularidad y flexibilidad. Esta estrategia facilitaría, a su vez, la adopción a una arquitectura en Kubernetes, en la que sería fundamental implementar réplicas para asegurar alta disponibilidad y balanceo de carga, lo que garantizaría un desempeño óptimo y resistencia ante fallos.

Finalmente, se propone el desarrollo de una interfaz gráfica para el proyecto "super cóndor", con el fin de mejorar la experiencia de usuario y simplificar la interacción. Esta interfaz visual permitiría una gestión más intuitiva y accesible, y podría expandirse con funcionalidades adicionales conforme se identifiquen nuevas necesidades derivadas de las mejoras y cambios implementados en el proyecto.

Referencias Bibliográficas

About Certbot. (s.f.). Certbot. https://certbot.eff.org/pages/about

Apache JMeter. (s.f.). JMeter en Español. https://jmeterenespanol.org

Apache Tomcat®. (s.f.). Apache Tomcat® - Welcome! https://tomcat.apache.org/

Buchanan, I. (s.f.). Containers vs Virtual Machines. Atlassian. https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms

Campbell, J. (s.f.). Comparación entre Kubernetes y Docker. Atlassian. https://www.atlassian.com/es/microservices/microservices-architecture/kubernetes-vs-docker

Celi-Párraga, R. J., Boné-Andrade, M. F., & Mora-Olivero, A. P. (2023). Ingeniería del Software I: Requerimientos y Modelado del Software. Editorial Grupo AEA.

Cómo Funciona. (18 de octubre de 2019). Let's Encrypt. https://letsencrypt.org/es/how-it-works/

Coursera. (29 de noviembre de 2023). What Are Ephemeral Ports? Coursera. https://www.coursera.org/articles/ephemeral-ports

Docker Compose. (s.f.). Docker Docs. https://docs.docker.com/compose/

Docker vs VM - Difference Between Application Deployment Technologies. (s.f.). AWS. https://aws.amazon.com/compare/the-difference-between-docker-vm/

Entender las GitHub Actions - Documentación de GitHub. (s.f.). GitHub Docs. https://docs.github.com/es/actions/about-github-actions/understanding-github-actions

Google. (s.f.). Transición de aplicaciones menos seguras a OAuth. Ayuda de Administrador de Google Workspace. https://support.google.com/a/answer/14114704?hl=es

IBM. (s.f.). *¿Qué es la migración de aplicaciones?* IBM. https://www.ibm.com/mx-es/think/topics/application-migration

Jangla, K. (2018). Accelerating Development Velocity Using Docker: Docker Across Microservices. Apress.

Java Documentation - Get Started. (s.f.). Oracle Help Center. https://docs.oracle.com/en/java/

MySQL :: Developer Zone. (s.f.). Developer Zone. https://dev.mysql.com/

¿Qué es Kubernetes? (17 de julio de 2022). Kubernetes. https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/

¿Qué Es NGINX y Cómo Funciona? NGINX explicado para principiantes. (25 de marzo de 2018). Kinsta. https://kinsta.com/es/base-de-conocimiento/que-es-nginx/

Qué es NodeJS y para qué sirve. (4 de septiembre de 2019). OpenWebinars. https://openwebinars.net/blog/que-es-nodejs/

¿Qué es una base de datos relacional? (s.f.). Oracle. https://www.oracle.com/co/database/what-is-a-relational-database/

¿Qué es una máquina virtual (VM)? (3 de marzo de 2024). IBM. https://www.ibm.com/mx-es/topics/virtual-machines

¿Qué es un certificado SSL? - Explicación del certificado SSL/TLS. (s.f.). AWS. https://aws.amazon.com/es/what-is/ssl-certificate/

¿Qué son las pruebas de software? (3 de febrero de 2024). IBM. https://www.ibm.com/es-es/topics/software-testing

Schildt, H. (2021). Java: The Complete Reference, Twelfth Edition. McGraw-Hill Education.

Tecnología JSP (JavaServer Pages). (s.f.). IBM. https://www.ibm.com/docs/es/dmrt/9.5?topic=files-javaserver-pages-jsp-technology

Vanier, E., Shah, B., & Malepati, T. (2018). Advanced MySQL 8: Discover the Full Potential of MySQL and Ensure High Performance of Your Database. Packt Publishing.

Vargas, D. (22 de mayo de 2024). ¿Qué es un servidor web y cómo funciona? Hostinger. https://www.hostinger.co/tutoriales/que-es-un-servidor-web

What is a Reverse Proxy? | Reverse Proxy Server - Kemp. (s.f.). Kemp Technologies. https://kemptechnologies.com/reverse-proxy

Apéndices

Apéndice A. Código del workflow de GitHub Actions para producción

```yaml
name: Java CI for Master

on:
  push:
    branches: [ "master" ]

jobs:
  build:

    runs-on: [self-hosted, linux]

    steps:
    - uses: actions/checkout@v4
    - name: Build with Ant
      run: ant -noinput -buildfile ./eisi/build.xml clean dist -Dj2ee.server.home=$CATALINA_HOME
    - name: Copy .war file to host server for master
      run: cp /home/docker/actions-runner/_work/coma/coma/eisi/dist/eisi.war /datadrive/deployment/master/
```

Apéndice B. Código del workflow de GitHub Actions para desarrollo

```yaml
name: Java CI for Develop

on:
  workflow_dispatch:

jobs:
  build:

    runs-on: [self-hosted, linux]

    steps:
    - uses: actions/checkout@v4
    - name: Build with Ant
      run: ant -noinput -buildfile ./eisi/build.xml clean dist -Dj2ee.server.home=$CATALINA_HOME
    - name: Copy .war file to host server for develop
      run: cp /home/docker/actions-runner/_work/coma/coma/eisi/dist/eisi.war /datadrive/deployment/develop/
```

Apéndice C. Código del workflow de GitHub Actions para mantener el runner activo

```yaml
name: Keep Runner Alive

on:
  schedule:
    - cron: '0 0 */6 * *'

jobs:
  keep-alive:
    runs-on: [self-hosted, linux]
    steps:
      - name: Keep Alive
        run: echo "Workflow ejecutado para mantener el runner vivo."
```