UNIVERSIDAD TÉCNICA DEL NORTE

Facultad De Ingeniería En Ciencias Aplicadas.

Carrera de Software.

DESARROLLO DE UNA ARQUITECTURA DEVOPS CON JENKINS Y DOCKER PARA REFORZAR EL MANEJO DE INTEGRACIÓN CONTÍNUA BASADO EN LA FASE 3 Y 4 DE TOGAF USANDO UNA PRUEBA DE CONCEPTO DEL MÓDULO DE CONSULTA DE UN SISTEMA FINANCIERO

Trabajo de grado previo la obtención del título de Ingeniero en Software

Autor:

José Luis Parra Vite

Director:

José Antonio Quiña Mera, PhD.

Ibarra – Ecuador

2023

UNIVERSIDAD TÉCNICA DE NORTE

BIBLIOTECA UNIVERSITARIA

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

1. IDENTIFICACIÓN DE LA OBRA

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

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1004048144		
APELLIDOS Y NOMBRES:	PARRA VITE JOSÉ LUIS		
DIRECCIÓN:	Ibarra, Parroquia de Alpachaca		
EMAIL:	jlparrav@utn.edu.ec		
TELÉFONO FIJO:	N/A	**TELÉFONO MÓVIL:**	0991458702
DATOS DE LA OBRA			
TÍTULO:	DESARROLLO DE UNA ARQUITECTURA DEVOPS CON JENKINS Y DOCKER PARA REFORZAR EL MANEJO DE INTEGRACIÓN CONTÍNUA BASADO EN LA FASE 3 Y 4 DE TOGAF USANDO UNA PRUEBA DE CONCEPTO DEL MÓDULO DE CONSULTA DE UN SISTEMA FINANCIERO		
AUTOR (ES):	JOSÉ LUIS PARRA VITE		
FECHA:	19/07/2023		
PROGRAMA:	■ PREGRADO	☐ POSTGRADO	
TÍTULO POR EL QUE OPTA:	INGENIERO DE SOFTWARE		
ASESOR / DIRECTOR:	PHD. JOSÉ ANTONIO QUIÑA MERA		

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de esta y saldrá en defensa de la Universidad Técnica del Norte en caso de reclamación por partes de terceros.

Ibarra, 19 de julio de 2023

EL AUTOR:

José Luis Parra Vite

C.I: 1004048144

CERTIFICADO DEL DIRECTOR DE TRABAJO DE GRADO

UNIVERSIDAD TECNICA DEL NORTE

FACULTAD DE INGENIERIA EN CIENCIAS APLICADAS

Ibarra, 19 de julio de 2023

CERTIFICACIÓN DEL DIRECTOR

Certifico que la Tesis previa a la obtención del título de Ingeniero de Software con el tema: **"DESARROLLO DE UNA ARQUITECTURA DEVOPS CON JENKINS Y DOCKER PARA REFORZAR EL MANEJO DE INTEGRACIÓN CONTÍNUA BASADO EN LA FASE 3 Y 4 DE TOGAF USANDO UNA PRUEBA DE CONCEPTO DEL MÓDULO DE CONSULTA DE UN SISTEMA FINANCIERO"** ha sido desarrollado y terminado en su totalidad por el Sr. José Luis Parra Vite, con cédula de identidad Nro. 100404814-4 bajo mi supervisión para lo cual firmo en constancia.

Es todo en cuanto puedo certificar a la verdad

PhD. Antonio Quiña Mera, MSc.

DIRECTOR DE TRABAJO DE TITULACIÓN

DEDICATORIA

Este trabajo se lo dedico a mis familiares que me apoyaron incondicionalmente a lo largo de mi vida estudiantil. Especialmente a mi madre, por sus consejos, sacrificio, amor y nobleza, me motivan para siempre dar mi mejor versión, y a mi padre por su rectitud, firmeza y fortaleza me inspira para mejorar como persona y profesionalmente.

A mis hermanas, por su infinito apoyo y motivación, y demostrarme que con esfuerzo y dedicación se puede salir adelante sin importar los obstáculos.

También, quiero dedicar mi trabajo a mi sobrina Gianella, esperando ser un gran ejemplo para ti y demostrarte que cada esfuerzo tiene su recompensa.

AGRADECIMIENTO

Quiero empezar agradeciendo a Dios por permitirme cumplir esta meta muy importante en mi vida. Agradezco con mucho cariño a mi madre, por su constante presencia, apoyo incondicional y ser un gran ejemplo de lucha. A mi padre por ser ejemplo de rectitud e inculcarme disciplina y trabajo duro. A mis hermanas por todo el valioso tiempo que me brindaron.

A la Universidad Técnica del Norte y a sus docentes profesionales mi eterno agradecimiento, por compartirme sus conocimientos, experiencias, por cada consejo recibido, por su paciencia y guía a lo largo de este proceso.

Agradezco a mis compañeros, especialmente a quienes supieron entregarme su sincera y desinteresada amistad, gracias por las risas y buenos momentos de diversión que supieron hacer esta etapa de mi vida mucho más placentera.

Tabla de contenido

Índice de figuras

Índice de tablas

RESUMEN

Mediante el presente trabajo de investigación se tiene como finalidad proponer el diseño de una arquitectura de software que cumpla con las características principales del framework TOGAF, respetando los principios que ofrece la norma ISO/IEC/IEEE 42010, permitiendo tener un diseño arquitectónico que puede ser implementado en proyectos integradores de la carrera CSOFT.

Este proyecto busca analizar las tecnologías y herramientas que pueden complementarse para dar cumplimiento con el ciclo de vida de DevOps, tomando principalmente a Jenkins como servidor de Integración y Entrega Continua, su implementación garantiza la automatización de procesos que permiten reducir tiempos de operatividad y detección de errores antes de desplegar los activos en producción con el fin de generar soluciones más eficientes y colaborativas. Además, se promueve el uso de Contenedores Docker para mejorar la portabilidad de las aplicaciones, optimizando recursos y ofreciendo una arquitectura escalable y medible.

Por otro lado, se procura generar una cultura de agilismo, tomando como base la gestión de proyectos con Scrum y adoptar los principios de comunicación y colaboración con DevOps, con el fin de tener mayor confiabilidad y estabilidad.

Palabras clave: Arquitectura de Software, TOGAF, DevOps, Scrum, Jenkins, Docker.

ABSTRACT

The purpose of this research work is to propose the design of a software architecture that complies with the main characteristics of the TOGAF framework, respecting the principles offered by the ISO/IEC/IEEE 42010 standard, allowing to have an architectural design that can be implemented in integration projects of the CSOFT career.

This project seeks to analyze the technologies and tools that can be complemented to comply with the DevOps life cycle, mainly taking Jenkins as an Integration and Continuous Delivery server, its implementation guarantees the automation of processes that allow to reduce operation times and error detection before deploying the assets in production in order to generate more efficient and collaborative solutions. In addition, the use of Docker Containers is promoted to improve the portability of applications, optimizing resources, and offering a scalable and measurable architecture.

On the other hand, we seek to generate a culture of agility, based on project management with Scrum and adopt the principles of communication and collaboration with DevOps, in order to have greater reliability and stability.

Keywords: Software Architecture, TOGAF, DevOps, Scrum, Jenkins, Docker.

Introducción

Antecedentes

La gestión de sistemas complejos es uno de los retos que debe afrontar la Ingeniería del Software actual, donde se debe tomar en cuenta que el mundo real cambia constantemente y los sistemas deben adaptarse a él para seguir siendo útiles. Por esta razón, los ingenieros de software deben encontrar técnicas y herramientas que le permitan adaptar los sistemas de manera sencilla a los nuevos entornos (Navasa Martínez, 2008).

Se ha identificado que estudiantes de la Carrera de Ingeniería de Software (CSOFT-UTN), durante proyectos integradores, tienen dificultad de integrar las funcionalidades o requerimientos de sus sistemas cuando se realizan trabajos en equipo, ya que no emplean una metodología de versionamiento e integración de código y porque no se utilizan correctamente estrategias de trabajo en equipo como DevOps, que es una filosofía de automatización de trabajo entre desarrolladores y operadores de Tecnologías de Información (TI), que ayuda a incrementar la calidad de los sistemas; automatizando y monitoreando los procesos de integración de software, testing, despliegue y cambios en la infraestructura (Díaz Cortés, 2019).

Situación Actual

Actualmente las metodologías de ingeniería de software son necesarias para poder realizar un proyecto profesional, tanto para poder desarrollar efectiva y eficientemente el software, como para que sirvan de documentación y se puedan rendir cuentas de los resultados obtenidos (Maida & Pacienzia, 2015).

El proceso de desarrollo de software depende del interés del equipo del proyecto y demás recursos. Uno de los principales problemas del desarrollo de software se produce durante el cambio de tecnología; como utilización de diferentes lenguajes de programación en distintos módulos o desarrollo e implementación de nuevos requisitos del proyecto y del entorno profesional (Dhir et al., 2019).

Los estudiantes que cursan la materia de Fábrica de Software de la carrera CSOFT, tienen bajo nivel de conocimiento sobre técnicas y herramientas de Integración Continua (CI por sus siglas en inglés), manejo de contenedores y a su vez sobre automatizar procesos con DevOps. A pesar del uso de diferentes lenguajes de programación y de la necesidad de un

desarrollo de software flexible, se ha reconocido la necesidad de métodos de mejora continua para optimizar los recursos de forma más eficaz y mejorar la calidad del software.

Prospectiva

La presente investigación pretende establecer una arquitectura de software, implementando metodologías y prácticas de DevOps, adicionalmente se utilizará Jenkins y Docker como herramientas para automatizar procesos de integración, acortar tiempos de operatividad y detección de errores en el desarrollo, de esta forma se garantiza una mayor calidad de los activos desplegados en producción, donde se creará una transformación digital de una forma ágil ajustando los costes de los proyectos (Cuervo, 2019). Esta arquitectura puede formar parte en la vida académica de los estudiantes logrando implementar estas técnicas en sus proyectos académicos y a su vez podría ser la base para proyectos profesionales.

Planteamiento del Problema

Actualmente no se le da mucha importancia al tener un ambiente de control de calidad automatizado, gestionado a través de orquestadores como Jenkins como parte del proceso de integración y despliegue continuo. En la Figura 1 se puede observar el árbol de problemas del proyecto.

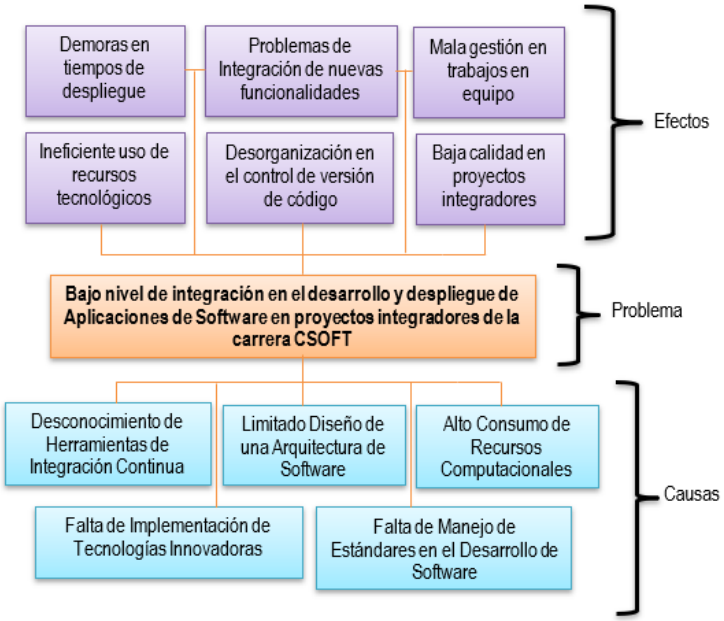

Figura 1. Diagrama de Causa y Efecto

Objetivos

Objetivo General

Implementar el módulo de consulta de un sistema financiero (cuentas y transferencias) usando una arquitectura DevOps con Jenkins y Docker para reforzar el manejo de Integración Continua de aplicaciones de Software basado en la fase 3 y 4 de TOGAF.

Objetivos Específicos

- Establecer el marco teórico de herramientas para el desarrollo e integración continua de aplicaciones de software.
- Definir la arquitectura utilizando la fase 3 y 4 del framework TOGAF.
- Validar la arquitectura mediante una prueba de concepto de un módulo de consulta de cuentas y transferencias.

Alcance

Se pretende diseñar una arquitectura de software implementando herramientas Open Source (Jenkins y Docker), especializadas en la integración continua de aplicaciones, donde forman una práctica constitutiva de DevOps, cuyo objetivo es reducir el tiempo que transcurre entre la aplicación de un cambio en un sistema y su puesta en producción, manteniendo al mismo tiempo una alta calidad del software (Marijan et al., 2018).

Para el diseño de la arquitectura se utilizará el framework de TOGAF (The Open Group Architecture Framework), que está apalancado a la norma ISO/IEC/IEEE 42010:2011, el cual establece requisitos para describir la arquitectura de sistemas y software a través de una convención, terminología común y mejores prácticas de Diseño y Descripción de Arquitectura (Life Art Tech, n.d.).

TOGAF proporciona un Método de Desarrollo de la Arquitectura (ADM, por sus siglas en inglés Architecture Development Method) que ofrece un manual referido a la planificación de la arquitectura y migración, en esta planificación se incluye una solución de implementación (Harani et al., 2018). Se utilizará la tercera y cuarta fase del ADM (arquitectura de sistemas de información y arquitectura tecnológica), dando como resultado el diseño de una arquitectura DevOps para el desarrollo e Integración Continua de aplicaciones de Software.

Para la validación de la arquitectura se utilizará una prueba de concepto, de este modo se simulará un sistema financiero básico, donde se desarrollará únicamente un módulo de consulta de cuentas y transferencias realizadas, dicha información estará almacenada

directamente a una base de datos PostgreSQL. Para el desarrollo de las API se utilizará la arquitectura REST utilizando NodeJS para el backend y el control de versionamiento de código será gestionado por GitHub.

Los beneficios potenciales al implementar herramientas tecnológicas en una organización son muchos, entre ellos se incluye; el carácter operativo, como el aumento de eficiencia automatizando procesos, la parte estratégica, como la mejora de los procesos empresariales (Canabal et al., 2017).

Adicionalmente se adoptará las técnicas y estrategias de DevOps para gestionar el ambiente de desarrollo y operacional del módulo definido, debido a que Jenkins y Docker son herramientas que trabajan a la par con esta filosofía, permitiendo tener un mejor rendimiento.

En la Figura 2 se detalla la arquitectura propuesta

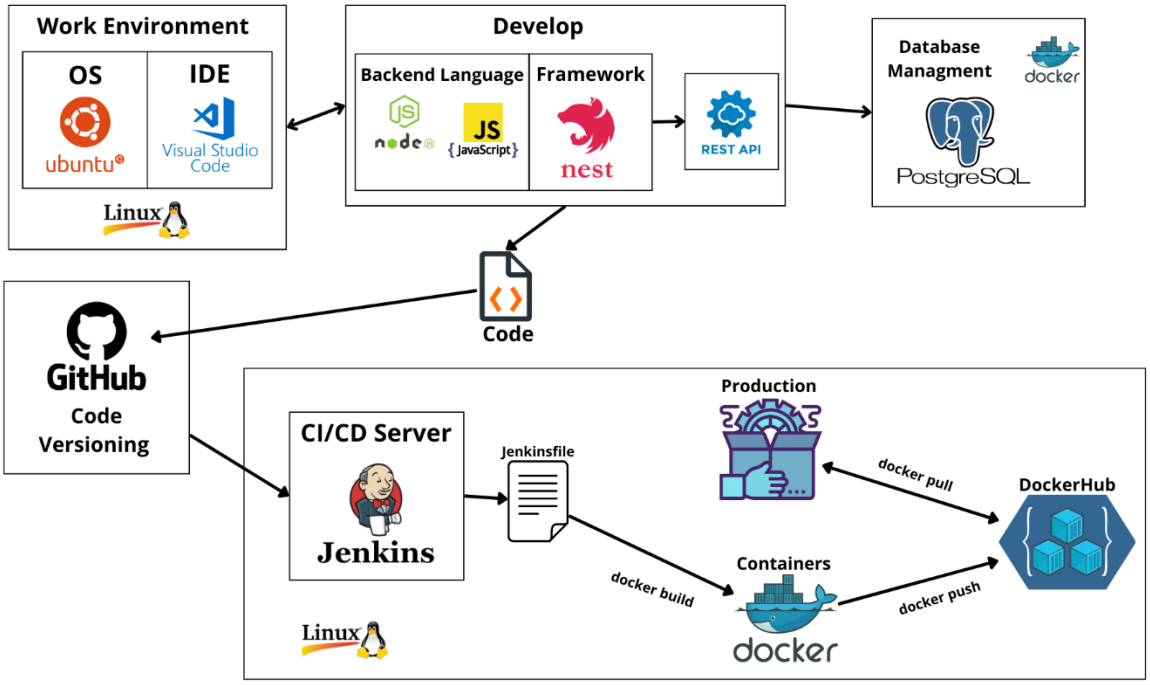

Figura 2. Arquitectura Objetivo

Justificación

Al ejecutar el presente trabajo de Titulación en el cual se diseñará una arquitectura con tecnologías innovadoras y altamente eficientes, se pretende solventar o dar solución a uno de los Objetivos de Desarrollo Sostenible, específicamente el objetivo N9; “Construir infraestructuras resilientes, promover la industrialización inclusiva y sostenible y fomentar la innovación”.

En base a una característica del Perfil Profesional de la carrera CSOFT, se reforzará el conocimiento de los estudiantes al aprender nuevos modelos, técnicas y tecnologías que van surgiendo y apreciando la necesidad del desarrollo profesional continuo (Universidad Técnica del Norte, 2022).

Justificación Tecnológica

El presente trabajo de titulación pretende crear un diseño de arquitectura de software, integrado con herramientas Open Source enfocado en integración continua de aplicaciones, para reforzar las habilidades tecnológicas de los estudiantes y a su vez puedan generar soluciones más eficientes y colaborativas.

En los últimos años, los entornos de nube han evolucionado con la llamada tecnología de "contenedores", que proporciona entornos de componentes ligeros que facilitan la migración de aplicaciones entre nubes y mejoran en gran medida la escalabilidad y el rendimiento, ofreciendo un enfoque alternativo a la virtualización de servidores (Iñiguez Sánchez, 2017).

Justificación Social

Se procura generar una cultura DevOps, donde los estudiantes tengan un mejor ambiente de trabajo, impulsando positivamente el proceso de gestión al cambio y puedan implementar buenas prácticas en el desarrollo y operatividad de aplicaciones de software.

Justificación Académica

El proyecto ayudará a los futuros estudiantes de la carrera CSOFT a entender sobre los temas de Integración y Despliegue Continuo y generar las bases de un modelo de fábrica de software empresarial que permita a la Universidad Técnica del Norte generar una fábrica de software y ser precursores de investigación innovadora.

CAPÍTULO I

1 Marco Teórico

1.1 Marco Conceptual de la Literatura

Para la creación del Marco Conceptual de la investigación, se toma como base la estructura de un Estudio de Mapeo Sistemático (SMS por sus siglas en inglés), permitiendo tener una amplia perspectiva de un área de investigación, haciendo uso de la clasificación de la información encontrada, es decir, se enfoca en realizar una búsqueda en la literatura y conocer los temas que se han cubierto y donde se han publicado (Bastidas Bastidas, 2020). Los pasos principales para realizar un estudio de Mapeo Sistemático son: planificación de la revisión, identificación del estudio y extracción y clasificación de datos (Galindo et al., 2018), que se puede observar en la Figura 3.

Figura 3. Proceso de SMS

Fuente: (Galindo et al., 2018)

1.1.1 Planificación de la Revisión

Esta fase es fundamental para dar inicio a la búsqueda de información, donde nos encontramos con 3 pasos clave (Galindo et al., 2018), que se muestran en la Figura 4:

Figura 4. Fase Planificación de la Revisión

Fuente: (Galindo et al., 2018)

Definición del Protocolo: en este paso se decide como será el proceso de la revisión de literatura y la manera más viable de reducir amenazas a la validez. La definición del protocolo para el SMS está adaptado por (Galindo et al., 2018).

Entre las amenazas que se pueden encontrar en la revisión de literatura es la información antigua que no ha tenido actualizaciones con nuevas investigaciones, por lo tanto, para reducir esta amenaza se buscará información del año 2017 hasta la actualidad. En caso de que un artículo proporcione información importante se revisará que en la actualidad se mantenga su validez.

Encuesta Bibliográfica: en este paso se obtiene la base de conocimiento del área de investigación a estudiar, tomando artículos como insumo y base de conocimiento.

La base de conocimiento de la presente investigación es el Área de la Computación, con todas sus sub-áreas (Ingeniería Informática, Ciencias de la Computación, Sistemas de Información, Tecnologías de la Información, Ingeniería de Software).

Definición de las Preguntas de Investigación: las preguntas de investigación se relacionan con el objetivo de los SMS, es decir, se requiere tener una visión clara de un área de investigación y reconocer en qué foros se ha publicado. En proceso se emplea efectivamente el modelo de 5W+1H el cuál consta de realizar 6 preguntas: Where, Who, What, Why, When y How.

- ¿Dónde se publican los trabajos más relevantes para mi investigación?
- ¿Quiénes son los autores de esos trabajos?
- ¿Qué área o sub-áreas de la computación están aplicando?
- ¿Por qué es importante este tema?
- ¿Cuándo se publicaron los trabajos?
- ¿Cómo se relacionan estos proyectos?

1.1.2 Identificación del Estudio

En esta fase se explica cómo se localizan los trabajos más relevantes para esta revisión, dividiéndolos en 4 sencillos pasos que se describen en la Figura 5:

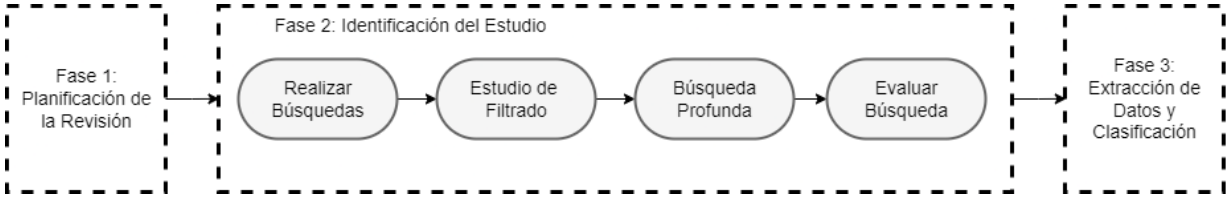

Figura 5. Fase Identificación del Estudio

Fuente: (Bastidas Bastidas, 2020)

Realizar Búsqueda: como base se tiene una pregunta de investigación especifica, que debe buscarse en investigaciones nuevas u originales obtenidas en congresos o publicaciones de revistas importantes.

Se identifica cuales son las bases de datos de búsquedas a utilizar, en este trabajo se utilizó: Scopus, Springer, IEEE Xplore y Google Schoolar. Posteriormente, en cada base de datos se utilizó la siguiente cadena de búsqueda:

"DevOps OR Continuous Integration AND Continuos Delivery OR Scrum OR Software AND Architecture"

Se seleccionó una cantidad de más de 200 trabajos obtenidos de las diferentes bases de datos, eliminando las investigaciones repetidas, teniendo así el primer set de estudios.

Estudio de Filtrado: se establece pautas de inclusión y exclusión para eliminar estudios que no son relevantes para las preguntas de investigación.

Partiendo como base el set de trabajos del paso anterior, se establece que la fecha de publicación debe estar comprendida entre el año 2017 y el año 2023, se descartaron las publicaciones antiguas, las que no tenían contenido relacionado al tema y las que no tenían suficientes referencias importantes.

Búsqueda Profunda: para evitar perder trabajos importantes se suele utilizar la técnica de muestreo bola de nieve, el cual consiste en tener un grupo inicial de artículos e ir hacia atrás revisando las referencias más importantes para encontrar nuevos trabajos.

Como punto importante para aplicar el muestreo bola de nieve, se tomó los artículos que han sido citados en diferentes trabajos sin importar el filtro de fechas, dando mayor importancia al contenido del trabajo. De igual manera se toma en cuenta sitios web oficiales.

Evaluar Búsqueda: se debe añadir más criterios de inclusión y exclusión para filtrar los nuevos resultados.

1.1.3 Extracción de Datos y Clasificación

En esta fase se concluye con 2 pasos finales que se muestran en la Figura 6.

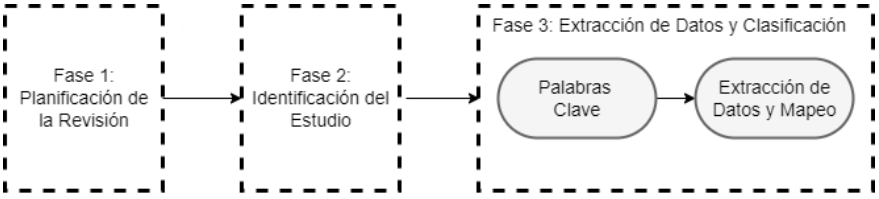

Figura 6. Fase Extracción de Datos y Clasificación

Fuente: (Bastidas Bastidas, 2020)

Palabras clave: se utilizan las palabras clave para reducir el tiempo de la construcción del esquema de clasificación, se agrupan las keywords para desarrollar un alto nivel de comprensión de la investigación.

El keywording o palabras clave comprende la necesidad de realizar una lectura rápida al resumen, introducción, conclusiones o recomendaciones, dependiendo de la complejidad de evidenciar los aspectos importantes de cada artículo, para definir el foco de investigación y poder agruparlos en diferentes categorías.

Extracción de Datos y Mapeo: Para extraer datos de los estudios identificados se crea una tabla con cada categoría de clasificación, el resultado se concentra en mostrar las frecuencias de las investigaciones de una manera estructurada, para así, responder las preguntas de investigación planteadas por el investigador.

1.2 DevOps

1.2.1 ¿Qué es DevOps?

Durante muchos años el mundo de la tecnología ha vivido en una separación total entre el mundo del desarrollo y el mundo de operaciones. Es decir, entre aquellos que diseñaban y desarrollaban las aplicaciones y aquellos que desplegaban las aplicaciones en una infraestructura y las operaban (Cuervo, 2019).

Definir DevOps puede llegar a ser más complejo que delimitarlo como una metodología, debido a su increíble flexibilidad e innovación al acoplar equipos de desarrollo con operadores de TI, encontrando así, una forma que permita tomar las mejores características de los dos mundos y aprovecharlas para mejorar el ciclo de vida de un activo de software. La idea principal de DevOps es encontrar la forma de realizar una entrega más continua de software, permitiendo agregar más funciones en producción y desplegándolas en menos tiempo.

Sin embargo, DevOps va más allá de la tecnología. Es el resultado de aplicar los principios más confiables de un dominio de manufactura y liderazgo a la cadena de valor de TI.

DevOps se basa en los conocimientos del paradigma ágil, teoría de restricciones, ingeniería de resiliencia, aprendizaje organizacional, cultura de seguridad, factor humano, entre otras (Retamal, 2019).

DevOps es un conjunto de prácticas y un movimiento cultural que tiene como objetivo romper las barreras entre los equipos de desarrollo y operación para mejorar la colaboración y la comunicación. Diferentes organizaciones han adoptado los principios de DevOps debido al enorme potencial, como un tiempo de producción mucho más corto, mayor confiabilidad y estabilidad. Sin embargo, a pesar de la adopción generalizada de DevOps y su infraestructura, existe una falta de comprensión y literatura sobre los conceptos, prácticas, herramientas y desafíos clave asociados con la implementación de estrategias DevOps (Khan et al., 2022).

Según (Rafi et al., 2020), DevOps es un nuevo paradigma que se enfoca en la naturaleza colaborativa y multidisciplinaria de la organización para controlar la entrega automatizada y las actualizaciones de software mientras se garantiza su efectividad. En la industria del software, DevOps es una tecnología de tendencia que se enfoca en la colaboración entre y dentro de los equipos involucrados en el desarrollo de software. Se refiere a mejorar el rendimiento de las empresas de software (implementación continua) mediante la coordinación de los equipos de desarrollo y operación en un solo proceso.

1.2.2 Ciclo de Vida de DevOps

Las Fases o etapas del ciclo de vida de DevOps las podemos observar en la Figura 7:

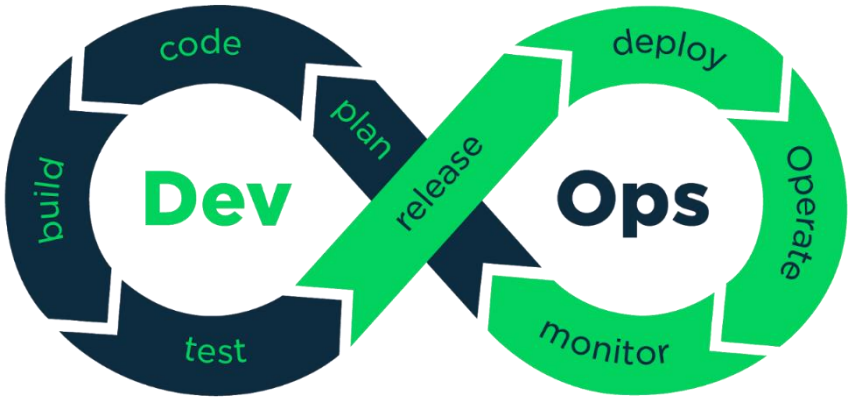

Figura 7. Ciclo de vida de DevOps

Fuente: https://www.rrhhdigital.com/secciones/tecnologia-e-innovacion/136859/Que-es-DevOps-y-que-debes-saber-para-convertirte-en-ello?target=_self

- **Plan:** En esta fase se definen los objetivos, requisitos y el alcance del proyecto, junto con las herramientas, planes de costos, recursos, tiempo y liberación a través de las iteraciones. Una herramienta muy común utilizada en esta fase es Jira de Atlassian.
- **Código:** Tomando como punto de referencia a la planificación, los desarrolladores se centran en escribir código de acuerdo con los requerimientos del cliente y se suele dividir el código en partes o unidades más pequeñas para tener una mejor visión del código. En esta fase se integra un gestor de versionamiento del código como lo es Git.
- **Construir:** En esta fase se realiza la construcción del código, validando y compilando todo el código fuente. Algunas herramientas utilizadas en esta fase son Maven y Gradle.
- **Prueba:** En esta fase se realizan pruebas a cada parte del código, con el fin de detectar errores o bugs que deben ser notificados inmediatamente para su posterior corrección. A diferencia de las pruebas tradicionales, se pueden automatizar las pruebas del código y ejecutarlas de forma continua para mejorar la calidad del software.
- **Integrar:** En esta etapa, se integran todas las unidades de los códigos. Eso significa que en este paso crearemos una conexión entre el equipo de desarrollo y el equipo de operaciones para implementar la integración y la implementación continuas. Un ejemplo de la herramienta utilizada es Jenkins (Edureka, 2022).
- **Desplegar:** Después de que el código pase por las pruebas y se integre, se realiza el despliegue continuo del software en un entorno de producción, donde se consigue mantener una versión estable del sistema.
- **Operar:** La operación en los ciclos DevOps se ocupa de la configuración y la gestión de la aplicación de software después del despliegue, por ejemplo, el aprovisionamiento de recursos y el autoescalado. Los orquestadores y otros métodos en tiempo de ejecución pueden utilizarse para instanciar y adaptar automáticamente en tiempo de ejecución la topología y los componentes de la aplicación (Alnafessah et al., 2021).
- **Monitorización:** Supervisar el rendimiento de las aplicaciones desplegadas mediante la recopilación y el análisis de los datos de uso, lo que puede ayudar a detectar e identificar las excepciones y proporcionar información para mejorar iterativamente el software. El rastreo y el diagnóstico continuo de los problemas de producción son importantes para guiar la evolución de la aplicación a lo largo de los ciclos de lanzamiento (Alnafessah et al., 2021).

1.2.3 Herramientas DevOps

Las herramientas son obligatorias para automatizar los procesos en DevOps. Las entregas de calidad con tiempos cortos necesitan un alto grado de automatización. Por lo tanto, elegir las herramientas adecuadas para su entorno o proyecto es importante cuando se implementa DevOps (Ebert et al., 2016).

Como DevOps pretende ser un modo de trabajo interfuncional para su funcionamiento, su ciclo de vida puede cumplirse a través de múltiples herramientas que interactúan entre sí y logran concretar una entrega continua (Maigualca Toapanta & Pillaga Zhagñay, 2021).

La implementación de integración de código, entrega continua y despliegue automatizado hacen uso de diferentes herramientas, teniendo muchas opciones en el mercado, lo cual dificulta decidir qué utilizar y cómo hacerlo. En este caso, el mercado de herramientas DevOps se formó para llenar el vacío creado por el desplazamiento del modelo de cascada (Kersten, 2018).

Para todas las fases del ciclo de vida de DevOps mencionadas en la sección 1.2.2 Ciclo de Vida de DevOps, existen varias herramientas que pueden dar cumplimiento con su respectiva funcionalidad, para alcanzar el nivel de automatización requerido. Existen herramientas gratuitas, de paga, open source, entre otras. En la Figura 8 podemos observar la tabla periódica de herramientas para la automatización de DevOps proporcionada por digital.ai.

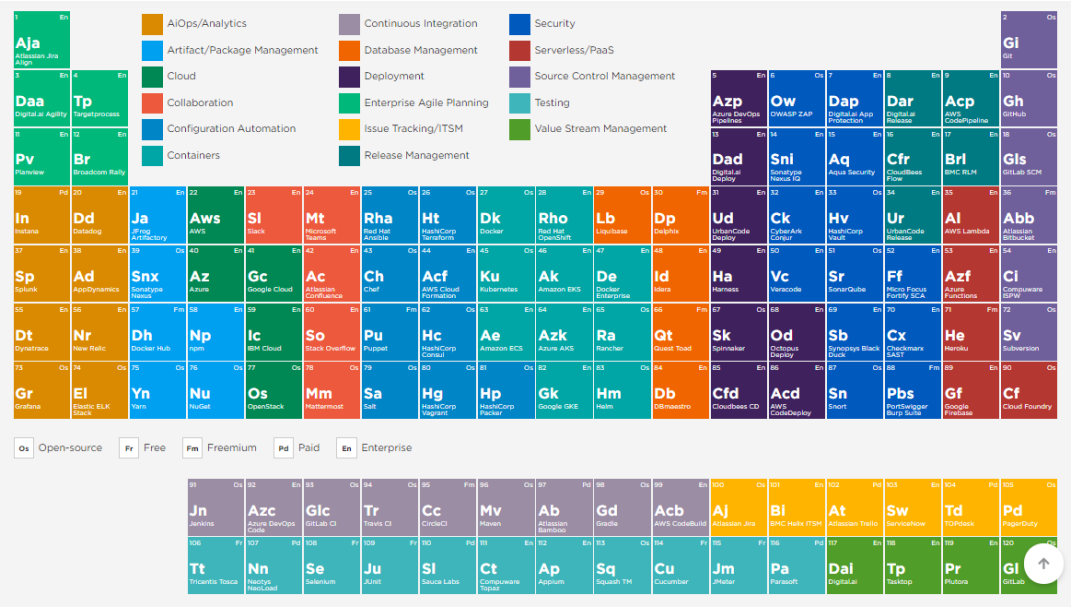

Figura 8. Tabla periódica de herramientas DevOps

Fuente: https://digital.ai/learn/devops-periodic-table/

1.3 Procesos de CI/CD

1.3.1 Integración Continua

Según (Eddy et al., 2017a), la Integración Continua es una práctica de desarrollo de software en la que los miembros de un equipo integran su trabajo con frecuencia, generalmente con cada persona integrando al menos una vez al día. La integración continua incluye la construcción automatizada y las pruebas unitarias y de integración de cada versión del sistema de software.

A medida que los procesos ágiles se han adaptado en la industria, también lo han hecho las prácticas de integración, entrega y despliegue continuo. Por esta razón, nace la importancia de la educación sobre estos temas, donde es importante que los estudiantes que están involucrados en proyectos de software empiecen a adquirir un nivel básico de conocimientos sobre estas áreas.

La Integración Continua (CI por sus siglas en inglés Continuous Integration) es un proceso de desarrollo en el que los desarrolladores introducen su código para obtener o completar cierta funcionalidad de un sistema. Cada integración es verificada con una construcción automática del software que anteriormente fue acompañada de pruebas automatizadas para detectar errores tan rápido como sea posible y que permiten que la aplicación sea probada cada vez que se realiza un cambio sobre el código de la misma (Vásconez Chávez, 2016). En la Figura 9 se puede observar el flujo de trabajo de la Integración Continua.

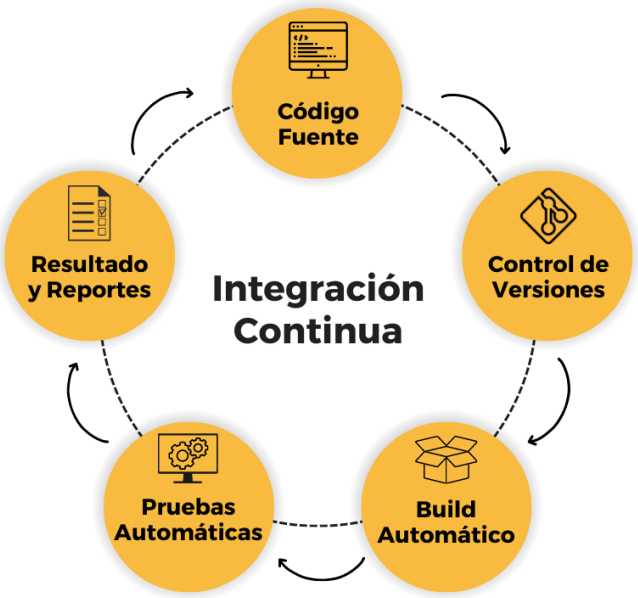

Figura 9. Ciclo de Vida de la Integración Continua

El riesgo es algo intrínseco a los proyectos dado su carácter temporal y único, específicamente en el desarrollo de software todavía no se tiene una mentalidad tan global, provocando que no se tenga en cuenta los riesgos que hay que asumir cuando se integran diferentes componentes, pero gracias a la Integración Continua se reduce el riesgo dado que los problemas durante el desarrollo se detectan mucho más antes permitiendo que los desarrolladores puedan corregirlos antes de llegar a un despliegue (Vásconez Chávez, 2016).

1.3.2 Entrega Continua

El objetivo principal de Entrega Continua o Continuous Delivery (CD) es pasar de un cambio de código a un cambio validado y listo para producción en la menor cantidad de tiempo posible y una intervención manual mínima en el proceso. En la Figura 10 se puede observar el alcance de la Entrega Continua en relación con la Integración Continua. El proceso de Entrega Continua incluye las etapas de prueba y análisis de la Integración Continua, pero también mejora los pipeline con varias herramientas de generación de informes y plantillas de implementación para un entorno de prueba, para garantizar la calidad de los testers (QA) (Eddy et al., 2017b).

La ejecución realizada manualmente de los procesos de integración, pruebas, construcción y liberación de software es mucho más propensa a encontrar errores o fallos en el producto final, debido a la intervención humana en cada proceso. El equipo de desarrollo por diferentes motivos puede pasar por alto el análisis y la evaluación de ciertas partes de código, dejando un resultado con menor calidad e incrementando el tiempo de entrega por motivo de corregir los errores que se encontraron en la etapa final.

La automatización de lo que se conoce como Pipeline de Entrega Continua, permite encontrar con mayor facilidad y de manera temprana fallos en la codificación, introducción de errores de regresión y errores en la integración de los diferentes componentes de software generando la retroalimentación que permitirá obtener un alto nivel de confianza y control sobre el proceso (Iñiguez Sánchez, 2017).

Un Pipeline de Entrega Continua es un proceso automatizado que se encarga de llevar el software desde el repositorio de código fuente hasta el usuario final. El proceso involucra la construcción de binarios seguido de varias etapas de prueba hasta el despliegue, ejecutado y monitoreado mediante una herramienta informática de integración y liberación continua de software como lo es Jenkins (Iñiguez Sánchez, 2017).

Los equipos de software pueden implementar los procesos de Integración y Entrega Continua y adaptarlos a la necesidad de su empresa o grupo de trabajo, empleando los grandes

beneficios de mantener entregas periódicas y detección de errores tempranos, para obtener una liberación de software mucho más fiable y garantizar la satisfacción del cliente, donde también se impulsará al equipo de trabajo a mantener una alta productividad para entregar software de calidad.

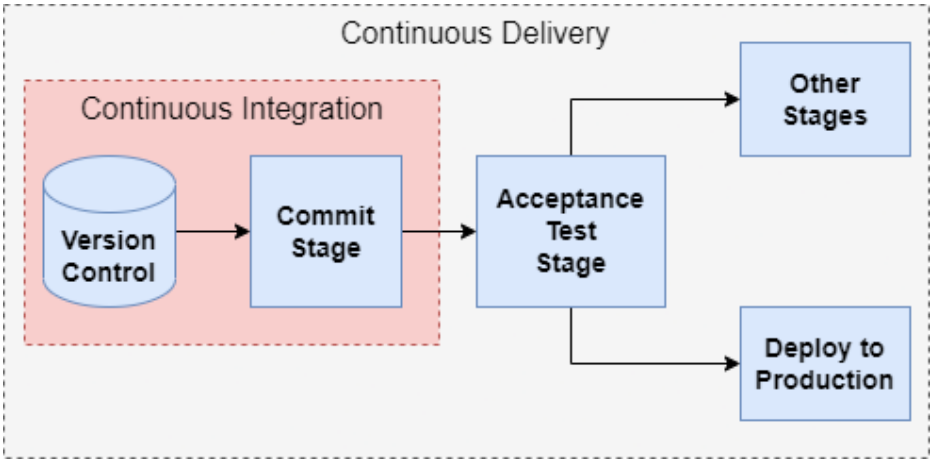

Figura 10. Relación entre CI y CD y sus alcances

Fuente: (Abbass et al., 2019)

1.3.3 Despliegue Continuo

El objetivo principal de todo proyecto de software es tener su producto listo en un ambiente de producción en el menor tiempo y con la mejor calidad posible, donde el cliente o usuario final pueda hacer uso del sistema desarrollado y presente una alta satisfacción y confianza al utilizarlo, es así como el producto final empieza a generar valor a la organización y/o empresa demostrando compromiso y calidad.

El Despliegue Continuo es el proceso que sigue una fábrica para adaptarse a sus cambios mediante la creación de un proceso de construcción personalizado para sus diversos productos. Este proceso de construcción consta de los siguientes pasos: obtener las dependencias de los repositorios, compilación del código, ejecutar pruebas automatizadas, ejecución de análisis de código, notificar a los desarrolladores sobre los posibles resultados y, finalmente, empaquetado de la compilación para el siguiente paso (El Khalyly et al., 2020).

En el Despliegue Continuo o Continuous Deployment (CD) se pretende proporcionar una manera ágil, fiable y totalmente automatizada de desplegar activos de software en diferentes entornos, como pueden ser preproducción o producción. Pero comúnmente en la industria existe

un debate sobre la distinción entre Continuous Delivery y Continuous Deployment ya que ambas utilizan la abreviatura “CD”.

Según (Shahin et al., 2017) menciona que la diferencia entre Despliegue Continuo de la Entrega Continua es un entorno de producción. Es importante señalar que la práctica del Despliegue Continuo implica la práctica de la Entrega Continua, pero lo contrario no es cierto. Mientras que el despliegue final en Entrega Continua es un paso manual, no debería haber pasos manuales en Despliegue Continuo, en el que tan pronto como los desarrolladores confirman un cambio, éste se despliega a producción a través de un Pipeline de despliegue, por consiguiente, la práctica de la Entrega Continua puede aplicarse a todo tipo de sistemas y organizaciones, la práctica del Despliegue Continuo sólo puede ser adecuada para determinados tipos de organizaciones o sistemas. En la Figura 11 se observa todo el proceso de CI/CD.

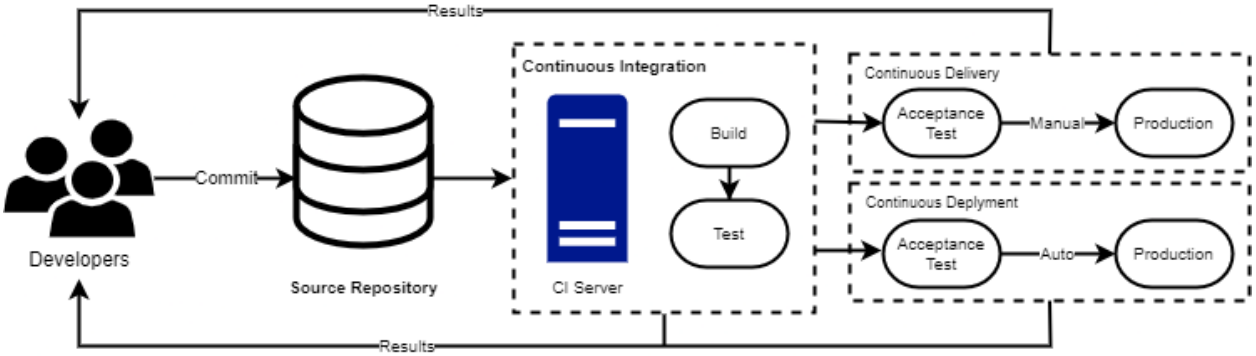

Figura 11. La relación entre la Integración, la Entrega y el Despliegue Continuo

Fuente**:** (Shahin et al., 2017)

1.4 Jenkins como herramienta de CI/CD

1.4.1 ¿Qué es Jenkins?

En el transcurso de este trabajo se mencionó diversos temas, como DevOps, para el trabajo entre desarrolladores y operadores de TI, que se involucran en los procesos de desarrollo hasta la puesta en producción de las aplicaciones. También se trató sobre Integración Continua, que es el proceso encargado de automatizar las fases de desarrollo, hasta prepararlo para producción a través de una Entrega o Despliegue Continuo.

De tal manera se necesita una herramienta o herramientas encargadas de integrar dichos procesos, por lo cual, la herramienta seleccionada en este trabajo es Jenkins, cuya función es automatizar y optimizar los procesos de CI/CD.

Jenkins es una herramienta de automatización de código abierto y autónoma que se utiliza para automatizar una amplia variedad de trabajos relacionadas con el desarrollo, integración de código, pruebas y entrega o despliegue de software. Su flexibilidad permite instalarlo en diferentes entornos, ya sea a través de paquetes nativos del sistema como Docker, o como una ejecución independiente en cualquier máquina que cuente con un entorno de ejecución de Java (JRE) instalado (Jenkins, 2022).

Figura 12. Logo de Jenkins

Fuente: https://github.com/jenkinsci/jenkins

Jenkins es un servidor para la Integración Continua, es decir, permite desplegar proyectos de software con mayor frecuencia, facilitando la integración de mejoras al proyecto, al analizar periódicamente el repositorio de código fuente o corrigiendo errores al notificar a los desarrolladores previo al despliegue de la aplicación. Jenkins está escrito en Java y es multiplataforma, permitiendo acceder mediante una interfaz web.

1.4.2 Modelo de CI/CD con Jenkins

Como se mencionó anteriormente, Jenkins es utilizado como una herramienta de automatización de Integración y Despliegue Continuo, siendo altamente personalizable y permitiendo definir flujos de trabajo de acuerdo con los desarrolladores y equipo DevOps. Adicionalmente, Jenkins cuenta con una amplia gama de plugins y herramientas de terceros que se pueden integrar para ampliar sus capacidades.

Las principales fases en las que interviene Jenkins para llevar una aplicación desde su desarrollo hasta producción pueden ser; control de código fuente, construcción y compilación de código fuente, ejecución de pruebas para el control de calidad y despliegue a producción, como se puede observar en la Figura 13.

Para cada fase existe una herramienta que se puede implementar a Jenkins, entre las principales tenemos aquellas que cumplen la función de conectar a Jenkins con diferentes repositorios de código, como GitHub, teniendo acceso a monitorear el código fuente, y en caso

de detectar un commit o cambio ejecuta un trabajo para compilar el código y enviarlo a un ambiente de pruebas, el cual puede conectarse a otra herramienta externa y dependiendo si el código pasa las pruebas se envía el build o construcción del código a un ambiente preproducción o directamente a producción en caso de tener total confianza en el ambiente de pruebas.

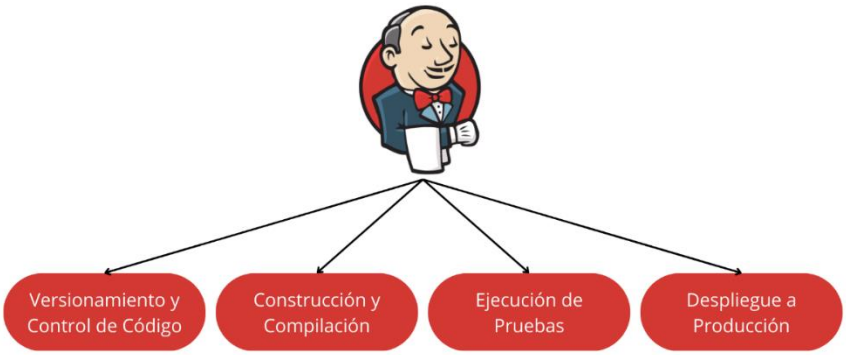

Figura 13. Modelo básico de CI/CD con Jenkins

1.4.3 Principales Componentes de Jenkins

Plugins para integrar otras herramientas

Jenkins cuenta con una amplia variedad de plugins que permiten extender sus funcionalidades y aportar nuevas características, como para integrar recursos de compilación, monitoreo, herramientas de análisis, herramientas de pruebas, conexión con servidores externos, entre otras.

Jenkins cuenta con una gran comunidad activa, la cual ha proporcionado en conjunto más de 1800 plugins, aparte de proveer soporte a otros programadores o corregir bugs encontrados en los plugins, con el fin de mejorar constantemente y entregar una herramienta confiable de automatización. La documentación proporcionada por Jenkins, los foros activos en diferentes plataformas, el contacto a través de la página web o por el equipo de Jenkins, son las opciones de soporte que se mantienen activas para ayudar a la comunidad.

Conexiones externas

Jenkins permite conectarse con servidores remotos por ssh haciendo la configuración del túnel ssh a través de la consola, para luego en el Job o pipeline ejecutar las acciones por medio de comandos de Shell que se ejecutarán directamente en el servidor remoto, Jenkins también permite realizar este proceso a través de plugins que gestionan las llaves ssh y permiten la conexión (Vinueza Celi, 2021).

Jenkins puede conectarse con varias herramientas de control de versiones de código, soportando las mayoría y más utilizadas basadas en Git, como; Github, Bitbucket o GitLab, también soporta Subversion, CVS, entre otros. Jenkins al conectarse con un repositorio puede monitorearlo y mantenerse alerta a cualquier cambio para ejecutar una acción preconfigurada.

Jobs y Pipelines

Un Job es una descripción del trabajo configurada por el usuario que debe realizar Jenkins, como la creación de una pieza de software, etc. (Jenkins, 2022).

Los Jobs se definen mediante archivos de configuración, que detalla los pasos para ejecutar una tarea, estos Jobs pueden ejecutarse manualmente o pueden ser programados para ejecutarse en un determinado tiempo o en respuesta a una acción externa.

Un Pipeline de Jenkins es un proceso continuo que describe el flujo de trabajo para construir, testear, desplegar un sistema, entre otras. Está compuesto por varias etapas secuenciales que especifican una acción determinada, si una etapa falla se detiene la ejecución del Pipeline para garantizar un alto nivel de calidad en los procesos finales.

La sintaxis de Pipeline es un Lenguaje Específico de Dominio (DSL) creado sobre el lenguaje de programación Groovy que permite al usuario especificar todas las propiedades y acciones de un trabajo de Jenkins en un lenguaje expresivo (Jenkins, 2022).

Los Pipelines y Jobs forman parte del ecosistema de Jenkins, los Jobs pueden configurarse desde la interfaz gráfica de la herramienta, de la misma forma que los Pipelines, sin embargo, estos últimos se pueden hacer también a través de un archivo de configuración con el formato de Jenkins el cual debe ser agregado a la raíz del proyecto (Vinueza Celi, 2021).

1.4.4 Control de Despliegues

Despliegue Inmediato

El despliegue inmediato es uno de los modelos de despliegue más rústicos y con mayor riesgo para los ambientes de producción, debido a que el cambio es de alto impacto porque la forma del cambio es apagar la versión existente y colocar la nueva versión y activarla. Este modo de despliegue es muy utilizado para los ambientes de desarrollo o escenarios de producción de bajo impacto. Las empresas que utilizan este método comúnmente ejecutan los despliegues en horarios no laborales o de menor impacto al usuario final (Casasola Girón, 2019). En la Figura 14 se puede ver una representación gráfica.

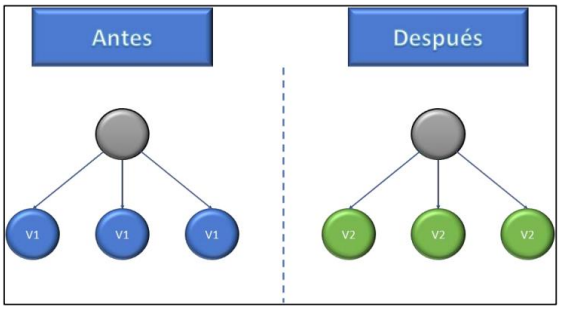

Figura 14. Despliegue Inmediato

Fuente: (Casasola Girón, 2019)

Despliegue Blue-Green

El despliegue Blue-Green se trata de replicar un ambiente igual al de producción con la nueva versión. Este es un método muy rápido y seguro, pero requiere que la arquitectura del sistema esté diseñada con un alto nivel de escalabilidad, introduciendo a este método conceptos como el uso de contenedores, virtualización, entre otros (Casasola Girón, 2019).

Este tipo de despliegue tiene las 2 versiones corriendo en paralelo, una en ambiente de producción (llamada Blue) y otra en ambiente de pruebas (llamada Green). Cuando se realiza una actualización al sistema, se lo hace en el ambiente Green, y después de comprobar que todo está funcionando correctamente, se hace el cambio de tráfico o redireccionamiento usando DNS o Load Balancer desde el ambiente Blue al ambiente Green, entonces el ambiente Blue ahora está esperando una nueva actualización y el ambiente Green ahora es de producción. De esta forma, en caso de tener errores con la nueva versión, se puede revertir fácilmente sin afectar la disponibilidad del servicio. Las etapas del despliegue Blue-Green pueden verse en la Figura 15.

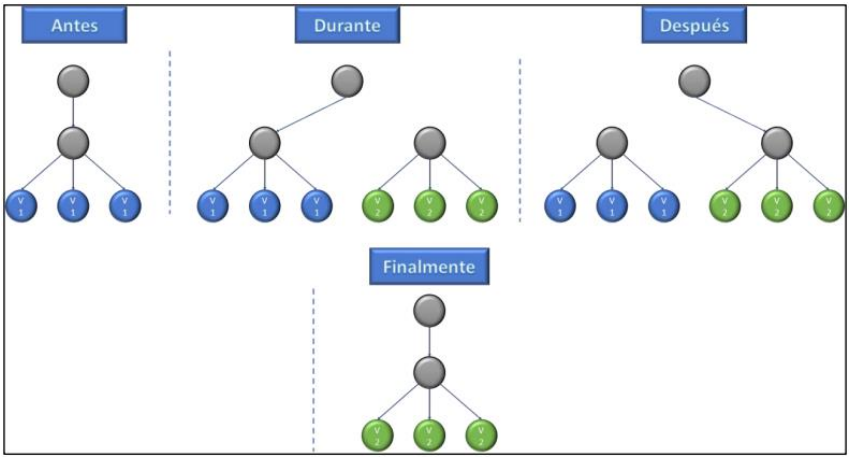

Figura 15. Despliegue Blue-Green

Fuente: (Casasola Girón, 2019)

Despliegue Canary

Este tipo de despliegue es similar al Blue-Green, con la diferencia de que se habilita controladamente, es decir, trasladando un porcentaje pequeño de usuarios a la nueva versión y aumentando gradualmente hasta llegar al 100% de los usuarios, y se monitorea su comportamiento antes de hacer el cambio completo. De esta forma, se pueden detectar posibles problemas de forma temprana y solucionarlos antes de que afecten a todos los usuarios.

Con este método también es necesario contar con un servidor de balanceo de tráfico, el cual se encarga de ir habilitando a cierta cantidad de usuarios la nueva versión de la aplicación. Como se ve, en este método y en el anterior, la arquitectura del sistema debe estar preparada para realizar este tipo de despliegues y nuevamente, se hace mención del uso de contenedores y virtualización (Casasola Girón, 2019). En la Figura 16 se puede observar el despliegue Canary.

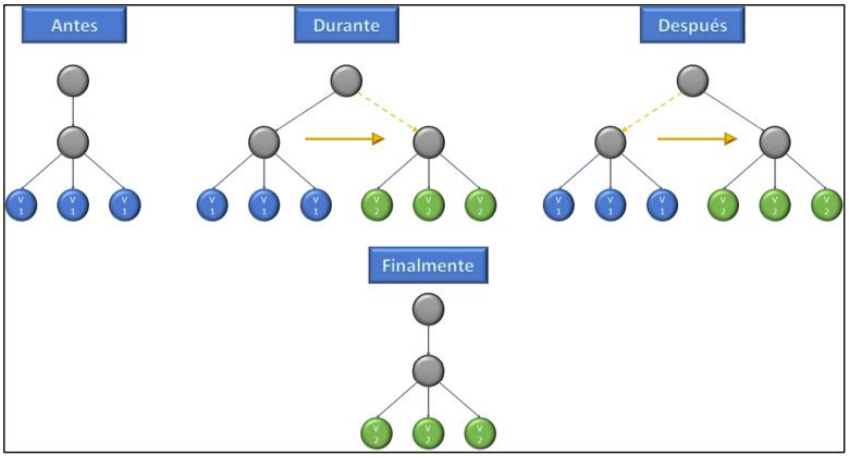

Figura 16. Despliegue Canary

Fuente: (Casasola Girón, 2019)

Despliegue Escalonado

En este modelo, el despliegue se realiza de manera escalonada, es decir, de servidor en servidor, o de servicio en servicio, por lo que el control de fallas en producción es menor. Es muy importante saber que este modelo puede ser utilizado solamente cuando los servidores o servicios funcionan independientemente, es decir, que la carga de trabajo sobre la aplicación está distribuida y que, si la base de datos también requiere cambios, no esté centralizada o que el cambio sobre la base de datos no afecte la versión anterior (Casasola Girón, 2019). En la Figura 17 se puede observar el proceso del despliegue escalonado de una manera gráfica.

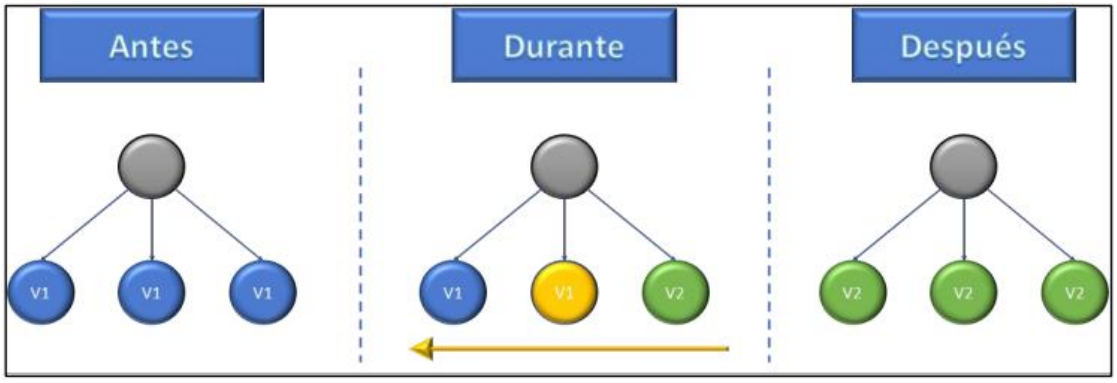

Figura 17. Despliegue Escalonado

Fuente: (Casasola Girón, 2019)

Mediante el uso de la herramienta Google Trends se pudo realizar una comparativa entre Jenkins, GitHub Actions y Azure DevOps, siendo herramientas muy utilizadas para procesos de CI/CD a nivel mundial. Tomando como periodo de tiempo los últimos 5 años, se realizó una comparativa de estos 3 software, proyectando como mayor búsqueda para CI/CD a Jenkins, obteniendo una media de 68 búsquedas semanales, mientras que GitHub Actions alcanza una media de 11 y Azure DevOps una media de 34. En la Figura 18 se puede observar la comparativa realizada de las herramientas.

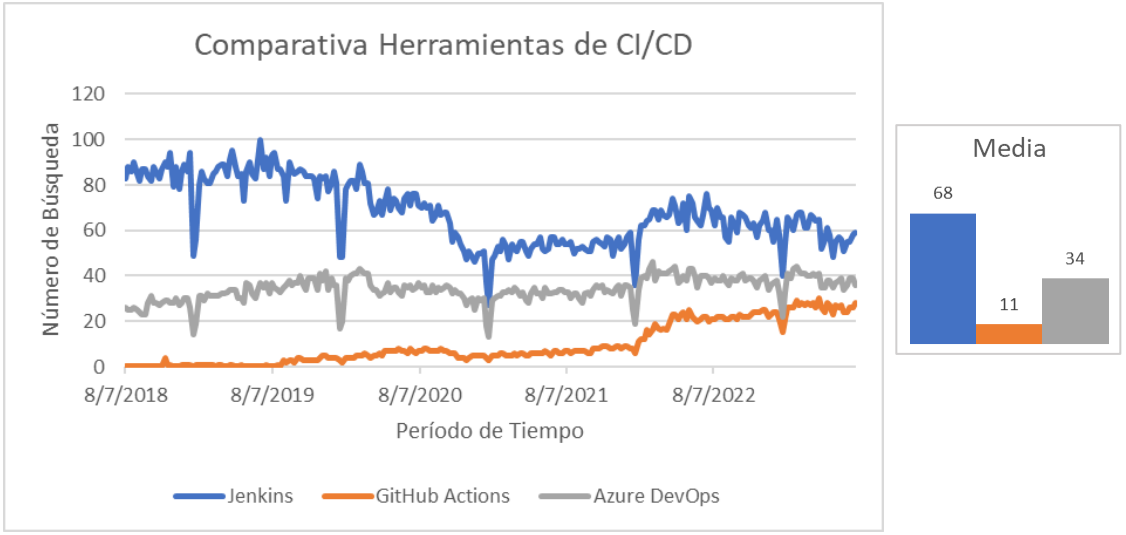

Figura 18. Comparativa de 3 herramientas de CI/CD a nivel mundial

Con la comparativa realizada se sustenta el uso de Jenkins como herramienta de CI/CD debido a su gran trayectoria y su amplia comunidad de usuarios a nivel mundial, brindando una amplia flexibilidad y estabilidad en diferentes entornos.

1.5 Tecnología de Virtualización con Contenedores

1.5.1 Definición de Contenedores

Hoy en día, las aplicaciones e infraestructuras distribuidas están pasando de un entorno centrado en las Máquinas Virtuales (MV) a un entorno centrado en los contenedores. La tecnología de contenedores es apoyada activamente por proveedores de PaaS, IaaS y proveedores de servicio de internet. Además, la tecnología de contenedores se está utilizando para desplegar aplicaciones a gran escala en áreas complejas como el análisis de Big Data, la Computación Científica, la Computación de Borde y el Internet de las cosas (IoT) (Casalicchio & Iannucci, 2020).

La tecnología de contenedores se usa ampliamente en todo tipo de centros de datos debido a su escalabilidad potente y flexible. Es una tecnología diseñada para soportar la arquitectura de microservicios. Como estándar de facto de la tecnología de contenedores, Docker juega un papel importante en el hospedaje de instancias de microservicios (Lu et al., 2019).

1.5.2 Máquinas Virtuales vs Contenedores

1.5.2.1 Arquitectura de Máquinas Virtuales

Según la documentación oficial de la plataforma de VMware (VMware, 2019), menciona que una máquina virtual es un equipo con software que, al igual que una computadora, ejecuta un sistema operativo y aplicaciones. Una máquina virtual está compuesta de un conjunto de especificaciones y archivos de configuración, y cuenta con el respaldo de los recursos físicos de un host. Todas las máquinas virtuales tienen dispositivos virtuales que ofrecen la misma funcionalidad que el hardware físico, pero son más portátiles, seguras y fáciles de administrar.

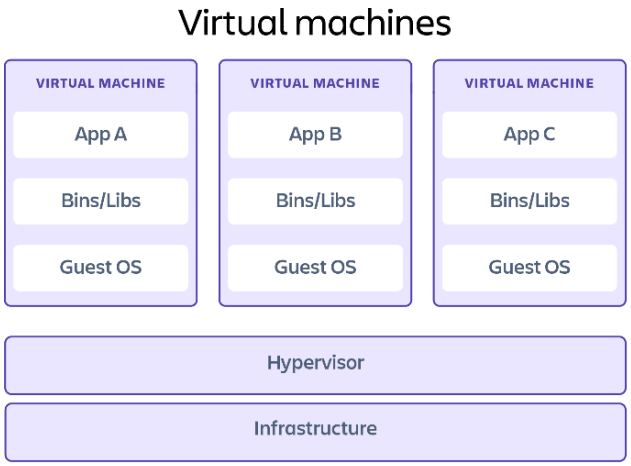

Figura 19. Capas de una Máquina Virtual

Fuente**:** (Atlassian, 2022)

La arquitectura de una Máquina Virtual está compuesta por varias capas, y para su correcto funcionamiento se necesita de componentes físicos o Hardware (procesador, RAM y almacenamiento). En la Figura 19, podemos ver la división de las capas, empezando por la capa física de la Infraestructura en la cual puede o no tener la capa del Sistema Operativo (SO) anfitrión y, encima se ubica la capa del Hypervisor que es una tecnología que permite alojar varias computadoras virtuales en una computadora física, entre los principales Hypervisors del mercado están VirtualBox de Oracle y VMware. Encima de estas capas se puede empezar a levantar una o varias máquinas virtuales, donde cada máquina virtual contará con su SO huésped, librerías, paquetes y demás recursos ejecutables que necesite para posteriormente desplegar o ejecutar una aplicación.

De este modo la arquitectura de las máquinas virtuales permite a cada máquina crear un procesador, memoria RAM y almacenamiento totalmente virtual, por lo tanto, los recursos de Hardware del computador anfitrión o la Infraestructura utilizada está siendo compartida con las máquinas alojadas, donde la limitante de rendimiento se enfoca en los recursos físicos que posee el host.

Por consiguiente, cada Máquina Virtual está totalmente aislada de las demás y no entiende que se encuentra alojada encima de otro Sistema Operativo, permitiendo que cuando exista algún daño o algo no funcione correctamente dentro de la máquina virtual, esto no afecte a las demás máquinas o al computador principal.

1.5.2.2 Arquitectura de Contenedores

Los contenedores son paquetes de software ligeros que contienen todas las dependencias necesarias para ejecutar la aplicación de software contenida. Estas dependencias incluyen elementos como bibliotecas del sistema, paquetes de código externos de terceros y otras aplicaciones a nivel del sistema operativo. Las dependencias incluidas en un contenedor existen en niveles de pila que se encuentran por encima del sistema operativo (Atlassian, 2022).

En la Figura 20, podemos observar el conjunto de capas que utiliza la tecnología basada en contenedores, donde se destaca la capa del Motor de Contenedor ubicada encima de la capa de infraestructura y del SO host. La capa del contenedor es utilizada como base para crear distintos contenedores que forman una sofisticada gestión y herramientas de virtualización, permitiendo empaquetar aplicaciones con sus respectivas dependencias y archivos de configuración.

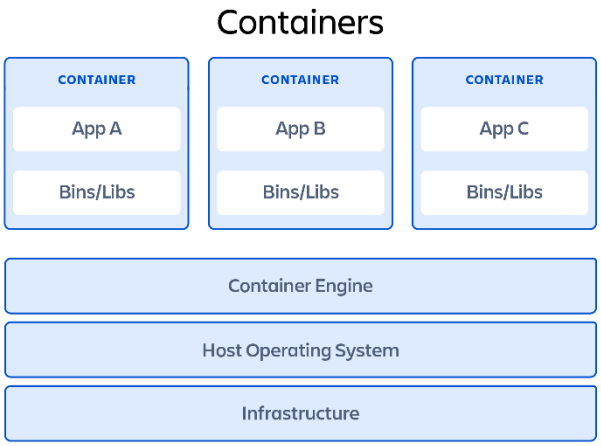

Figura 20. Capas de Contenedores

Fuente**:** (Atlassian, 2022)

Existen contenedores de sistemas y de aplicaciones, donde los contenedores de sistemas pueden ejecutar un Sistema Operativo por completo, mientras que un contenedor de aplicaciones se crea a partir de una serie de capas de imágenes que aprovechan el servicio Union File System (UFS) (Casalicchio & Iannucci, 2020). Con los UFS o Sistema de Archivos de Unión, los contenedores pueden utilizar una única jerarquía de sistemas de archivos sin tener que hacer varias copias, lo que ahorra espacio en el disco.

Las claves del éxito de los Contenedores son varias: posee una gestión más sencilla del ciclo de vida de las aplicaciones distribuidas, una sobrecarga insignificante para cuando se ejecutan en servidores físicos o virtuales y su tiempo de arranque, reinicio y parada, que se reduce hasta un orden de magnitud con respecto a las MV. Por último, pero no menos importante, los contenedores permiten la portabilidad de las aplicaciones, que es donde falló la virtualización basada en hipervisor (Casalicchio & Iannucci, 2020).

1.5.2.3 Diferencias y Similitudes entre MV y Contenedores

En la Tabla 1 se presenta varias características a tomar en cuenta donde se compara a las Máquinas Virtuales y a los Contenedores.

Tabla 1. Similitudes y Diferencias entre Máquinas Virtuales y Contenedores

Característica	Máquina Virtual	Contenedor
Aislamiento	Provee un aislamiento completo del sistema operativo host y otras máquinas virtuales.	Provee un aislamiento ligero del host y otros contenedores, pero no proporciona un límite de seguridad tan sólido como una máquina virtual.
Sistema Operativo	Ejecuta un sistema operativo completo, incluido el kernel, por lo	Ejecuta el modo de usuario de un sistema operativo y se puede adaptar para

	que requiere más recursos del sistema (CPU, memoria y almacenamiento).	contener solo los servicios necesarios para su aplicación, utilizando menos recursos del sistema.
Compatibilidad de invitados	Ejecuta casi cualquier sistema operativo dentro de la máquina virtual.	Se ejecuta en la misma versión del sistema operativo que el host (el aislamiento de Hyper-V le permite ejecutar versiones anteriores del mismo sistema operativo en un entorno de máquina virtual liviano)
Despliegue	Despliegue de máquinas virtuales individuales mediante el Centro de administración de Windows o Hyper-V Manager; despliegue de varias máquinas virtuales mediante PowerShell o System Center Virtual Machine Manager.	Despliegue de contenedores individuales mediante Docker a través de la línea de comandos; despliegue de varios contenedores mediante un orquestador como Azure Kubernetes Service.
Actualizaciones y mejoras del sistema operativo	Descargue e instale actualizaciones del sistema operativo en cada máquina virtual. La instalación de una nueva versión del sistema operativo requiere una actualización o, a menudo, simplemente la creación de una máquina virtual completamente nueva.	Para actualizar o mejorar archivos del SO se necesita: 1. Editar el archivo de compilación de la imagen de su contenedor (Dockerfile) para apuntar a la última versión de la imagen base del SO. 2. Reconstruir la imagen de contenedor con esta nueva imagen base. 3. Empujar la imagen del contenedor a su registro de contenedores. 4. Volver a implementar mediante un orquestador.
Almacenamiento persistente	Utiliza un disco duro virtual (VHD) para el almacenamiento local de una sola máquina virtual o un recurso compartido de archivos SMB para el almacenamiento compartido por varios servidores.	Use Azure Disks para almacenamiento local para un solo nodo o Azure Files (recursos compartidos SMB) para almacenamiento compartido por varios nodos o servidores.
Balanceo de carga	El equilibrio de carga de la máquina virtual mueve las máquinas virtuales en ejecución a otros servidores en un clúster de conmutación por error.	Los contenedores en sí mismos no se mueven; en su lugar, un orquestador puede iniciar o detener automáticamente los contenedores en los nodos del clúster para administrar los cambios en la carga y la disponibilidad.
Tolerancia a fallos	Las máquinas virtuales pueden conmutar por error a otro servidor en un clúster y el sistema operativo de la máquina virtual se reinicia en el nuevo servidor.	Si falla un nodo de clúster, el orquestador vuelve a crear rápidamente cualquier contenedor que se ejecute en él en otro nodo de clúster.
Redes	Utiliza adaptadores de red virtual.	Docker emplea una vista aislada de un adaptador de red virtual, permitiendo tener una virtualización más liviana al compartir el firewall del host con los contenedores, donde el uso de los recursos se utiliza con mayor eficiencia.

Fuente: (Microsoft Learn, 2021)

La principal diferencia entre las máquinas virtuales y los contenedores es cómo utilizan los recursos virtuales del hardware físico en el que se ejecutan. Con una máquina virtual, los

recursos de hardware físico se comparten y la asignación de recursos es responsabilidad de la capa de virtualización (el hipervisor en este caso). Los contenedores, por otro lado, se ejecutan como un proceso más y el kernel del Sistema Operativo administra la asignación de recursos físicos (Roldán Martínez et al., 2018).

En la Figura 21 se realiza una comparativa de búsqueda entre estas tecnologías de virtualización, mediante la herramienta Google Trends se estableció como referencia a Docker, VirtualBox y VMware como términos de búsqueda a nivel mundial en un periodo de los últimos 5 años, obteniendo la cantidad de búsquedas semanales de cada herramienta durante ese período.

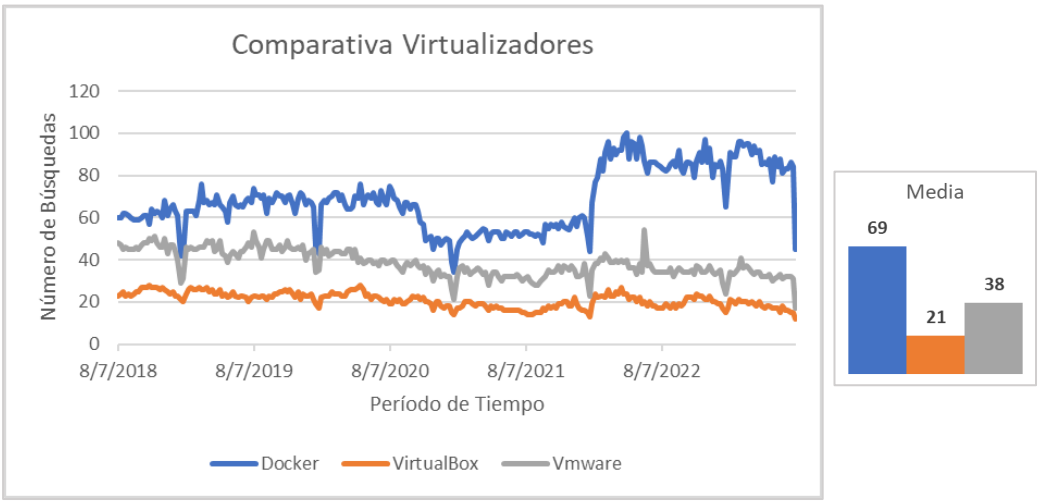

Figura 21. Comparativa de herramientas de Virtualización

De las 3 herramientas de virtualización se establece a Docker como la herramienta con mayor búsqueda a nivel mundial, teniendo una media de 69 búsquedas por semana y un gran crecimiento de interacciones a partir del año 2022 como lo indica en la Figura 21. Teniendo una gran ventaja sobre VMware con una media de 21 y VirtualBox con una media de 38. Docker se ha destacado como la herramienta más utilizada a nivel mundial en los últimos años para virtualización, gracias a su capacidad para agilizar el desarrollo de software, mejorar la portabilidad de aplicaciones y simplificar la implementación en diferentes entornos.

1.6 Administración de Contenedores con Docker

1.6.1 ¿Qué es Docker?

Docker es una plataforma de código abierto basada en contenedores, desarrollada para construir, enviar y ejecutar aplicaciones directamente, por ello, se utiliza ampliamente para desarrollar, testear e implementar aplicaciones. De la misma manera las aplicaciones Docker se

empaquetan juntas para garantizar la prueba de aplicaciones en otras plataformas al empaquetar las dependencias, bibliotecas, archivos binarios u otras piezas de software requeridas en un contenedor. Si bien no reemplaza completamente las Máquinas Virtuales, proporciona aislamiento para otras aplicaciones en contenedores (Lingayat et al., 2018).

Docker es una tecnología que facilita el proceso de desarrollo, ejecución hasta la distribución de software, gracias a que empaqueta todas y cada una de las dependencias de una aplicación en un contenedor para moverlas a otro ambiente diferente y ejecutarlas sin ningún problema, de modo que gracias a ello se logró dar solución al problema muy común entre desarrolladores en donde al ejecutar la aplicación en un ambiente diferente, la aplicación presentaba errores que no tenía en el entorno de desarrollo.

Según la investigación realizada por (Lingayat et al., 2018) menciona que Docker está desarrollado en el lenguaje de programación Go y utiliza características del kernel de Linux, al mismo tiempo, utiliza diferentes grupos de aislamiento de Linux, como espacio de nombres, cgroups y UFS o UnionFS que es un sistema de archivos utilizado para hacer que Docker sea más ligero y rápido mediante la creación de capas, a su vez, Docker usa variantes de UnionFS como AUFS, btrfs, vfs y Device Mapper, cabe mencionar que el sistema Docker consta del tiempo de ejecución de la herramienta de empaquetado, Docker Engine y Docker Hub.

Se utiliza Docker en el actual proyecto debido a su gran facilidad de implementación de aplicaciones y porque comúnmente es referenciada a las arquitecturas de microservicios y dentro de la cultura DevOps. Docker permite entregar código con mayor rapidez, estandarizar operaciones de aplicaciones y ahorrar recursos económicos utilizando lo necesario para su funcionalidad. Cabe mencionar que Docker trabaja y se integra muy bien con el resto de las herramientas que se utilizan en este proyecto, teniendo como resultado una arquitectura modular y escalable, que puede ser implementada en cualquier tipo de proyecto.

1.6.2 Arquitectura de Docker

Como se muestra en la Figura 22, el Servidor Docker es el Kernel Daemon que puede ser desplegado en servidores locales al igual que en servidores remotos. El puente de comunicación que existe entre el Servidor y el Cliente es la API Rest. Como el Cliente, tenemos el Docker CLI que además de proporcionar las interfaces correspondientes para los usuarios, puede gestionar contenedores, imágenes, redes y volúmenes de datos para la persistencia de la información. Después de empaquetar las imágenes, los usuarios pueden crear contenedores en

virtud de las imágenes y luego operarlos, durante lo cual el Servidor puede ser invocado para controlar los recursos del disco (You & Sun, 2022).

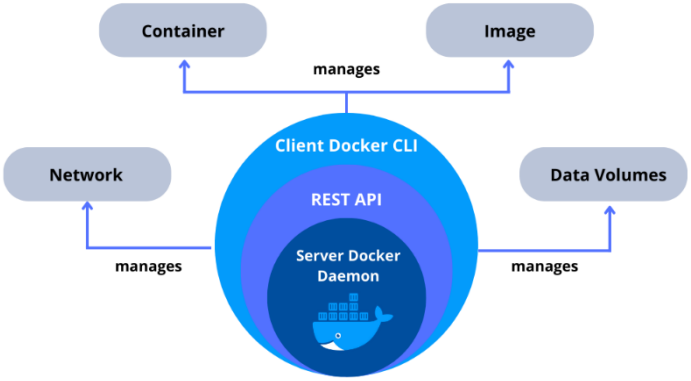

Figura 22. Arquitectura Docker

Fuente: Adaptada de (You & Sun, 2022)

1.6.3 Principales Componentes de Docker

Imágenes

Una imagen de Docker es un archivo binario que contiene todo lo necesario para ejecutar un contenedor de Docker. Esto se puede considerar como un conjunto superpuesto de capas en el sistema de archivos, o como una instantánea inmutable mucho más ligera y sin estado del sistema de archivos en un momento determinado. La Figura 23 muestra un contenedor Docker que proporciona un entorno de desarrollo web básico basado en una imagen de sistema operativo Ubuntu apilada, una imagen de Emacs y una imagen de servidor web Apache. Se identifica por su nombre y una etiqueta que identifica una versión específica de la imagen (Iñiguez Sánchez, 2017).

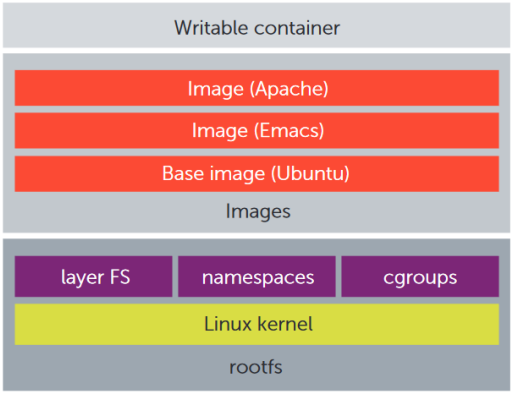

Figura 23. Arquitectura de Imagen Docker de un kernel de Linux

Fuente: (Iñiguez Sánchez, 2017)

Una imagen Docker es como una instantánea de una máquina virtual pero mucho más ligera. Sobre esta se debe añadir; código fuente, entorno de ejecución, librerías, variables de ambiente y ficheros de configuración para posteriormente ejecutar una aplicación en cualquier otro entorno. Para crear una imagen se puede hacer desde cero o a partir de una imagen existente que se obtenga desde un repositorio público o privado (Roldán Martínez et al., 2018). En la Figura 24 se observa los procesos que una imagen Docker puede realizar.

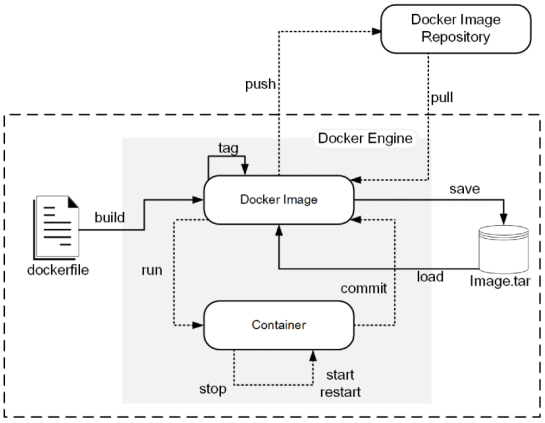

Figura 24. Proceso de Despliegue de una Imagen Docker

Fuente**:** (Kwon & Lee, 2020)

Contenedores

Un contenedor es una instanciación de una imagen que cumple con las características suficientes para ejecutar una aplicación, la cual se ejecutará en Docker. A partir de una misma imagen podemos instanciar uno o más contenedores como se muestra en la Figura 25, los cuales estarán totalmente aislados los unos de los otros, es decir, cada contendor se ejecutará en un entorno independiente con sus configuraciones respectivas, asegurando la independencia entre aplicaciones instaladas sobre el sistema operativo anfitrión (Roldán Martínez et al., 2018). Los contenedores poseen un único id, el cual se utiliza para realizar las diferentes acciones como crear, iniciar, ejecutar, detener o eliminar un contenedor.

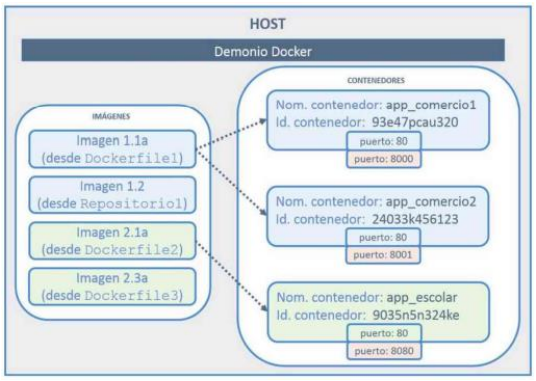

Figura 25. Contenedores expuestos al exterior con su id y puertos

Fuente**:** (Roldán Martínez et al., 2018)

Volúmenes

Los contenedores son elementos efímeros que una vez son borrados o detenidos totalmente pueden desaparecer sin dejar rastro de la información manipulada. Sin embargo, es posible que se necesite hacer persistencia de los datos que las aplicaciones generan, incluso una vez hayamos eliminado cualquier rastro de ellos. Para llevar a cabo esta tarea de persistencia surgen los volúmenes de datos, permitiendo separar el ciclo de vida de los contenedores de los datos. Esta separación es posible gracias a que Docker aloja los volúmenes fuera del contenedor, por lo general lo ubica en la ruta */var/lib/Docker/volumes/* (Roldán Martínez et al., 2018).

De acuerdo con lo mencionado por (Pahl, 2015), un volumen de datos es un directorio dedicado dentro de uno o más contenedores que pasa por alto el sistema de archivos de unión proporcionando roles para datos compartidos o persistentes. Los volúmenes se pueden compartir y reutilizar entre contenedores, como se muestra en la Figura 26.

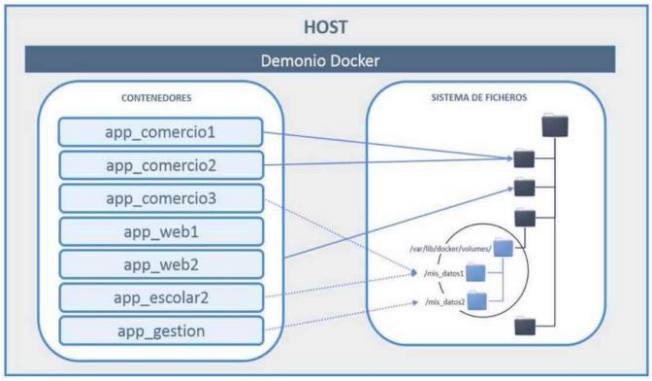

Figura 26. Contenedores y Persistencia de Datos

Fuente: (Roldán Martínez et al., 2018)

Redes

Las Redes de Docker sirven para crear conexiones entre los contenedores Docker existentes y con el exterior, para que puedan crear una comunicación entre sí y a su vez enlazarse con el host. Comúnmente Docker utiliza el Modelo de Red de Contenedores o Container Network Model (CNM) para la creación de redes, dicho modelo puede estandarizar los pasos para crear redes en contenedores con varios controladores de red como se observa en la Figura 27.

Por lo mencionado en la plataforma de edureka.co (edureka, 2022) se concluye que el CNM estandariza los pasos necesarios para proveer redes para contenedores que utilizan varios controladores de red. CNM requiere un almacén de clave-valor distribuido como una consola para almacenar la configuración de la red. El CNM cuenta con interfaces para complementos IPAM y complementos de red. Las API del complemento de IPAM se usan para crear o eliminar grupos de direcciones y asignar o desasignar direcciones IP de contenedores, mientras que las API del complemento de red se usan para crear o eliminar redes y agregar o eliminar contenedores de las redes.

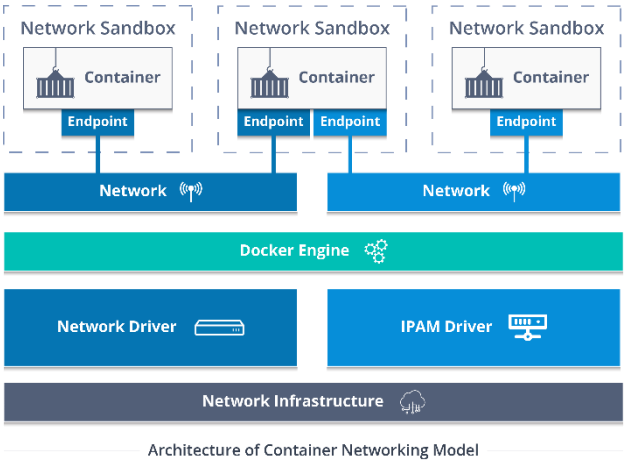

Figura 27. Arquitectura de CNM

Fuente**:** (edureka, 2022)

1.6.4 Terminología de Docker

De acuerdo con la publicación de Microsoft en su sitio web (Microsoft Learn, 2022), menciona los principales términos y definiciones utilizadas para tratar los temas de la contenedorización con Docker, con los que se debe familiarizarse para poder entender y profundizar en temas de Docker.

- **Imagen de Contenedor:** es un paquete con todas las dependencias, configuración e información necesaria para crear un contenedor. Una imagen puede derivarse de varias imágenes base que son capas apiladas para formar el sistema de archivos del contenedor.
- **Dockerfile:** es un archivo de texto que contiene una serie de instrucciones para crear una imagen Docker.
- **Build:** es el proceso de construir una imagen de contenedor en función de la información y contexto proporcionados por su Dockerfile y archivos adicionales en el directorio donde se construye la imagen.
- **Contenedor:** es una instancia de una imagen de Docker. Un contenedor puede representar la ejecución de una sola aplicación, proceso o servicio. Consta del contenido de una imagen de Docker, un entorno de ejecución y un conjunto estándar de instrucciones.
- **Volúmenes:** proporciona un sistema de archivos grabable que el contenedor puede usar. Los volúmenes agregan una capa de escritura encima de la imagen del contenedor, para que los programas puedan acceder al sistema de archivos de escritura.
- **Etiqueta o Tag:** es una marca o etiqueta que se puede aplicar a las imágenes para que se puedan identificar diferentes imágenes o versiones de esta.
- **Repositorio o repo:** es una colección de imágenes de Docker relacionadas e identificadas con una etiqueta que muestra la versión de la imagen y autoría. Existen repositorios públicos con acceso a cualquier persona y repositorios privados que las empresas suelen tener para almacenar y administrar sus propias imágenes.
- **DockerHub:** es un registro público o privado para alojar imágenes Docker y trabajar con ellas, además, proporciona activadores de compilación y enlaces web junto con integración con GitHub y Bitbucket.
- **Docker Compose:** es una herramienta de línea de comandos y formato de archivo YAML con metadatos para definir y ejecutar aplicaciones de varios contenedores. Permite definir una sola aplicación basada en varias imágenes con uno o más archivos (**.yml**) que pueden anular los valores según el entorno.
- **Clúster:** es una colección de hosts Docker expuestos como si fuera un solo host Docker virtual, de modo que la aplicación pueda escalar a múltiples instancias de los servicios distribuidos en varios hosts dentro del clúster. Los clústeres de Docker se pueden crear con Kubernetes, Azure Service Fabric, Docker Swarm y Mesosphere DC/OS.

- **Orquestador:** es una herramienta que simplifica la gestión de clústeres y hosts Docker. Los orquestadores permiten administrar sus imágenes, contenedores, redes configuraciones, equilibrio de carga, disponibilidad, configuración del host y demás, a través de una interfaz de línea de comandos (CLI) o una Interfaz de Usuario (UI) gráfica.

1.7 Arquitecturas de Software con TOGAF

Se define como una arquitectura empresarial que proporciona una base general para el desarrollo de software empresarial. Ayuda a organizar el proceso de desarrollo con un enfoque sistemático para reducir errores, cumplir con los plazos, mantenerse dentro del presupuesto y alinear la TI con las unidades comerciales para brindar resultados de calidad (Palacios Acosta, 2022).

La arquitectura empresarial (EA) trata de aprovechar los beneficios del uso de arquitecturas para atacar el problema de la alineación entre el negocio y la TI, mientras que la aplicación de Arquitectura Empresarial implica crear modelos que describen una empresa, incluyendo elementos comerciales y de TI, para que pueda cumplir con los requisitos de gestión y mantenerse durante el período de su vida útil (Proenca & Borbinha, 2017).

1.7.1 TOGAF

TOGAF (The Open Group Architecture Framework), es un marco de trabajo que proporciona un método detallado para crear, administrar e implementar arquitecturas empresariales y sistemas de información llamado Método de Desarrollo de Arquitectura (ADM). TOGAF ADM es un método general que incluye un conjunto de actividades que representan el progreso de cada fase de ADM y el modelo arquitectónico utilizado y creado durante la fase de Diseño de la Arquitectura Empresarial (Osadhani et al., 2019).

1.7.2 Fases de TOGAF

Como podemos ver en la Figura 28, se muestran las 8 principales fases que proporciona el Método de Desarrollo de Arquitectura (ADM) de TOGAF:

1) Preliminar
2) Visión de la Arquitectura
3) Arquitectura del Negocio
4) Arquitectura de Sistemas de Información
5) Arquitectura Tecnológica
6) Oportunidades y Soluciones

7) Planificación de la migración
8) Gobierno de la implementación
9) Gestión del Cambio de la Arquitectura

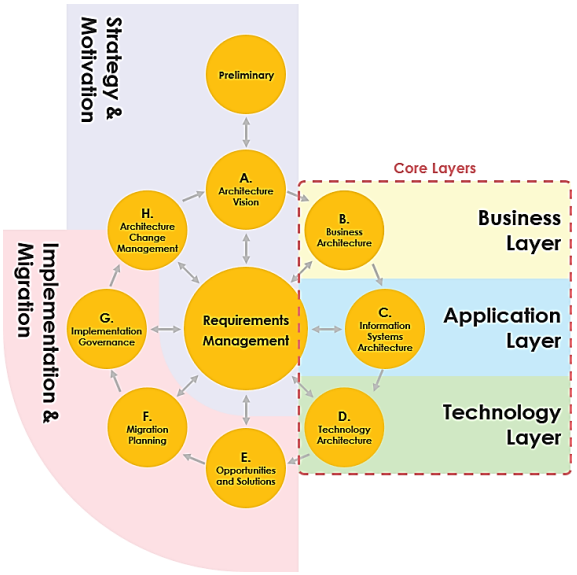

Figura 28. TOGAF ADM

Fuente: (Visual Paradigm, 2022a)

1.7.3 ArchiMate

ArchiMate es el lenguaje de modelado abierto e independiente proporcionado por The Open Group para la arquitectura empresarial, respaldado por diferentes proveedores de herramientas y firmas de consultoría. ArchiMate proporciona instrumentos para ayudar a los arquitectos empresariales a describir, analizar y visualizar las relaciones entre los diferentes dominios de la arquitectura de una manera inequívoca como se observa en la Figura 29, similar a disciplinas bien establecidas como la ingeniería civil o la edificación y la construcción que utilizan estándares internacionalmente aceptados para describir sus diseños (Visual Paradigm, 2022a).

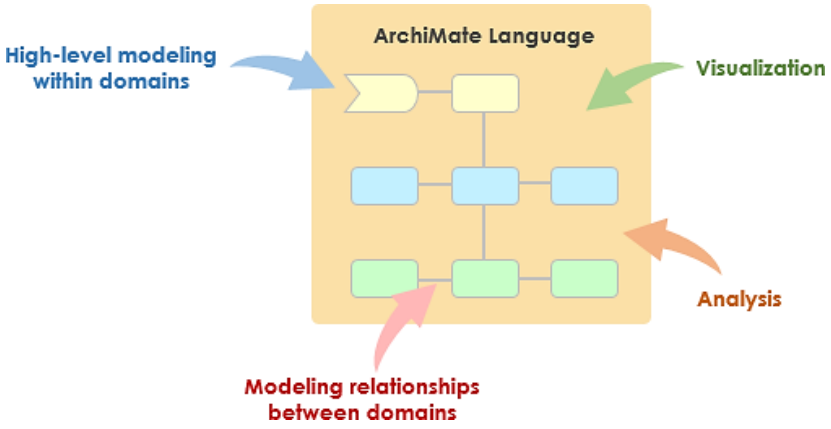

Figura 29. Lenguaje ArchiMate

Fuente: (Visual Paradigm, 2022a)

Entre sus grandes características ArchiMate ofrece un estándar para la representación de la arquitectura empresarial, siendo aceptado y utilizado por una amplia comunidad alrededor del mundo. Al utilizar ArchiMate como lenguaje de modelado se puede notar la utilización de lenguaje común para describir y visualizar arquitecturas empresariales, facilitando la comunicación y comprensión para tomar decisiones. Además, ArchiMate tiene una gran compatibilidad con TOGAF, lo que permite una integración fácil y eficiente. Sin embargo, este es un lenguaje avanzado, teniendo un nivel de complejidad para su uso y comprensión.

1.7.4 El marco principal de ArchiMate

ArchiMate tiene dos marcos para su uso, el marco principal que se muestra en la Figura 30, el cual consta de aspectos (estructura pasiva, estructura activa y de comportamiento) y capas (negocio, aplicación y tecnología). Pero también existe el marco completo de ArchiMate, en el cual se agregan nuevas capas y nuevos aspectos, siendo la estrategia, equipos físicos y elementos de implementación y migración las capas agregadas y en cuanto a los aspectos únicamente se agrega la motivación. En el presente trabajo se hace uso del marco principal de ArchiMate.

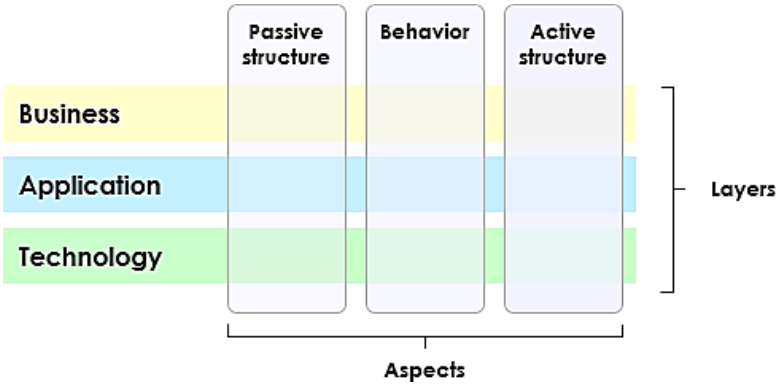

Figura 30. Marco Principal de ArchiMate

Fuente: (Visual Paradigm, 2022b)

Según el sitio web visual-paradigm.com (Visual Paradigm, 2022b), tiene la siguiente definición para los aspectos y capas del marco de ArchiMate:

Aspectos

- El aspecto de estructura activa representa los conceptos estructurales como; actores comerciales, componentes de aplicación y sujetos de la actividad que muestran el comportamiento real.
- El aspecto de comportamiento representa el comportamiento realizado por los actores como; procesos, funciones, eventos y servicios. Los conceptos de comportamiento se asignan a conceptos estructurales, para mostrar quién o qué muestra el comportamiento.
- El aspecto de estructura pasiva que puede interpretarse como información representa los objetos sore los que se realza el comportamiento. Suelen ser objetos de información en la capa empresarial y objetos de datos en la capa de aplicación, pero también se pueden usar para representar objetos físicos.

Capas

Teniendo en cuenta que las capas superiores utilizan los servicios proporcionados por las capas inferiores, como se puede observar en la Figura 30.

- La capa de negocio ofrece productos y servicios a clientes externos que se realizan mediante procesos comerciales realizados por actores comerciales.
- La capa de aplicación es compatible con la capa de negocio con servicios de aplicación que se realizan mediante aplicaciones de software.

- La capa de tecnología ofrece servicios de infraestructura como; procesamiento, almacenamiento o comunicación, necesarios para ejecutar aplicaciones, realizados por hardware de computadora y comunicación y software de sistema.

1.7.5 ISO/IEC/IEEE 42010 como estándar para el diseño de arquitecturas

La ISO (The International Organization for Standarization) y la IEC (The International Electrotechnical Commission) forman el sistema especializado de normalización mundial, encargados de elaborar normas internacionales para deferentes áreas. Por ende, la ISO/IEC/IEEE 42010 es una norma internacional elaborada por el Comité Técnico Conjunto ISO/IEC JTC 1, Tecnología de la Información, Subcomité SC7, Ingeniería de Software y Sistemas, en cooperación con el Comité de Normas de Ingeniería de Software y Sistemas de la Computer Society del IEEE, en virtud del acuerdo de Cooperación Partner Standards Development Organization entre ISO e IEEE. (International Organization Of Standardization, 2011)

Esta norma internacional establece una estructura fundamental para describir arquitecturas. Las disposiciones incluidas en esta norma internacional se aplican con el propósito de asegurar que las descripciones de arquitecturas cumplan con las características deseadas. Además, esta norma específica disposiciones que refuerzan los requerimientos deseados de los marcos de arquitectura y los lenguajes de descripción de arquitecturas (ADL por sus siglas en inglés) para compararlos e integrarlos.

En la Figura 31 se muestra un diagrama conceptual que detalla la estructura completa de la norma internacional ISO/IEC/IEEE 42010.

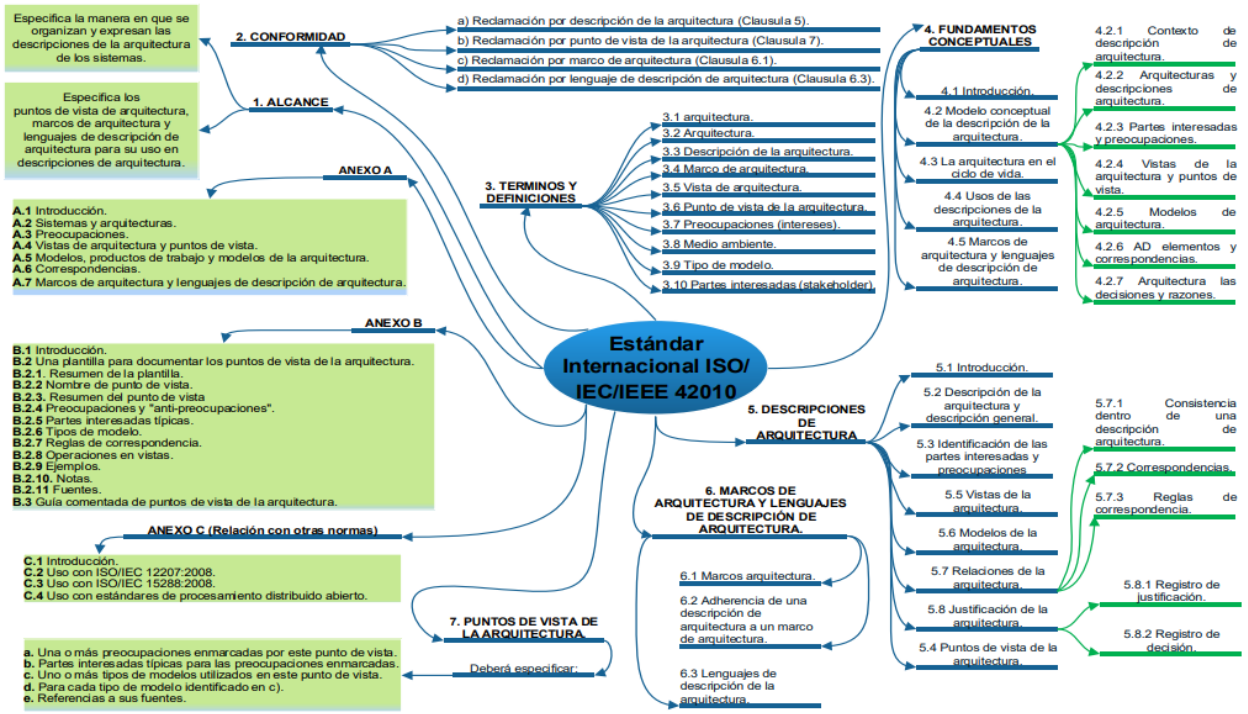

Figura 31. Mapa mental del Estándar Internacional ISO/IEC/IEEE 42010

Fuente: (Padilla Jaramillo, 2019)

1.8 Tecnologías de desarrollo

1.8.1 Microservicios

Actualmente los Microservicios son la última tendencia en el diseño de servicios de software, debido a que establecen un enfoque de la arquitectura de software y sistemas que se basa en el concepto establecido de modularización, teniendo en cuenta los límites técnicos. Cada microservicio puede operar independientemente de los demás servicios, ofreciendo acceso a su lógica y datos internos a través de una interfaz de red bien definida. Esto incrementa la flexibilidad del software convirtiendo a cada microservicio en una unidad independiente de desarrollo, despliegue, operaciones, versionado y escalado (Jamshidi et al., 2018).

De acuerdo con (Click-IT, 2022), podemos ver los principales beneficios a la hora de implementar microservicios dentro de una arquitectura de software como se puede observar en la Tabla 2.

Tabla 2. Beneficios de los microservicios

Beneficio	Descripción

Modularidad	Permite realizar despliegues y desarrollos de manera totalmente independiente, sin afectar a otros servicios, mejorando la escalabilidad de la aplicación por módulos de manera horizontal.
Seguridad	Al tener microservicios aislados de los demás, mejora la detección de problemas sin afectar al resto de servicios.
Versatilidad	Permite adaptar cada servicio al uso de diferentes tecnologías, lenguajes de programación y bases de datos.
Mejora Continua	Crear múltiples microservicios de manera simultánea, permite lanzar nuevas funcionalidades frecuentemente, para tener un servicio robusto, flexible y eficiente.
Test	Facilita los procesos de pruebas, donde se encuentra errores de programación para corregirlos con mayor rapidez.
Velocidad	Reduce el consumo de memoria y aumenta el rendimiento.

Fuente: (Click-IT, 2022)

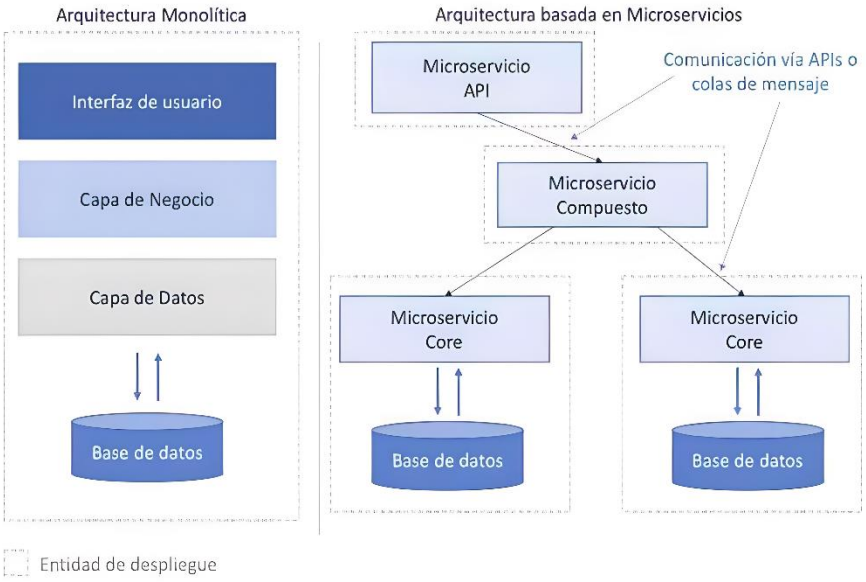

Figura 32. Arquitectura monolítica vs microservicios

Fuente**:** (Roldán Martínez et al., 2018)

Como se puede observar en la Figura 32, la comunicación entre los distintos componentes de la aplicación se realiza a través de API, eliminando la necesidad de que los desarrolladores deban conocer y comprender todo el código, por ende, se obtiene una ventaja en eficiencia, flexibilidad y capacidad de crecimiento. Ante el reto de aligerar y facilitar la portabilidad de los componentes en diferentes entornos, surge el uso de contenedores como Docker, empezando así, adoptar un enfoque DevOps (Click-IT, 2022).

Para el presente proyecto la comunicación que se tiene en el entorno distribuido o de microservicios es mediante el proceso de comunicación denominado HTTP REST, para realizar las peticiones o llamadas a las API de los microservicios. Este concepto de REST se lo presenta a continuación.

1.8.2 API REST

Según lo mencionado por (Roldán Martínez et al., 2018), REST o Representational State Transfer es un protocolo de comunicación basado en HTTP, definido para el desarrollo web sencillo y adaptado para el diseño de API. Se lo define como un protocolo sin estado, es decir, que no debe guardar información de estado en el servidor, sino que toda la información necesaria debe encontrarse en la consulta realizada por el cliente. Para el diseño de las API REST es imprescindible el uso correcto de las URI (Universal Resource Identifier), por tal motivo, se presentan las siguientes reglas que deben ser respetadas.

- Debe evitarse la inclusión de verbos en las URI, ya que no tienen que estar ligadas a una acción concreta. Las acciones permitidas se especifican mediante comandos o verbos de HTTP que podemos ver en la Tabla 3**.**
- Un mismo recurso debe identificarse con una sola acción.
- Las URI no dependen del formato del recurso si no de su identificador.
- Debe mantenerse una jerarquía lógica.
- Si es necesario filtrar la información, dichos filtros no deben incluirse en las URI, sino en los parámetros de la petición HTTP.

Tabla 3. Comandos HTTP

Comando HTTP	Utilidad
GET	Consulta de recursos
POST	Creación de recursos
PUT	Edición de recursos
DELETE	Borrado de recursos
PATCH	Edición de partes concretas de un recurso

Fuente**:** (Roldán Martínez et al., 2018)

Por otra parte, Red HAT (Red Hat, 2020), menciona que REST no es un protocolo ni un estándar, sino más bien es un conjunto de límites de arquitectura, debió a su implementación por diferentes formas. También es importante tener en cuenta los encabezados y parámetros en los

métodos HTTP, porque contienen información de identificación importante con respecto a; metadatos, autorización, URI, almacenamiento en caché, cookies y otros elementos como se observa en la Figura 33. En su misma plataforma de Red HAT describe ciertos criterios que se deben cumplir para que una API se considere de RESTful.

- Tener una arquitectura cliente-servidor con la gestión de solicitudes a través de HTTP.
- Comunicación sin estado entre el cliente y el servidor
- Datos que pueden almacenarse en caché, optimizando las interacciones entre el cliente y el servidor.
- Interfaz uniforme entre los elementos, para que a información se transfiera de forma estandarizada.
- Contar con un sistema de capas que organiza en jerarquías invisibles para el cliente cada uno de los servidores que participan en la recuperación de la información solicitada.
- Código disponible según se solicite, es decir, la capacidad para enviar códigos ejecutables del servidor al cliente cuando se requiera, lo cual amplía las funciones del cliente.

Figura 33. API REST

Fuente**:** https://www.seobility.net/en/wiki/REST_API

1.8.3 PostgreSQL

Como gestor de Base de Datos (BDD) se optó por utilizar PostgreSQL, debido a que es una BDD relacional muy utilizada actualmente en el mercado, multiplataforma, orientada a objetos, extensible, escalable y Open Source. Al ser una BDD tan reconocida y potente, tiene sus usos en múltiples campos, desde almacenamiento y procesamiento de datos, Sistemas de Información Geográfica (GIS), para servicios web, PaaS entre otras. Además, PostgreSQL es una BDD que encaja perfectamente en una arquitectura de microservicios y a su vez es igual de funcional en entornos de contenedores como Docker.

1.8.4 Github

Actualmente es indispensable utilizar algún repositorio de código si se trabaja en proyectos de software, para el presente proyecto se pretende utilizar Github, que es una plataforma propiedad de Microsoft, la cual ofrece a los desarrolladores poder alojar su código en la nube de manera segura y llevar un control de versiones de código con la herramienta GIT. Github ha crecido potencialmente en los últimos años, demostrando ser una herramienta que facilita la vida de los desarrolladores, entregando varios servicios como automatización de flujos de trabajo, despliegue de aplicaciones, entre otras.

1.8.5 Node.JS

Node.JS también llamado Node, es un entorno de ejecución diseñado para el lenguaje de programación JavaScript (JS) que se ejecutaba exclusivamente en el lado del cliente. Node utiliza un modelo de operaciones asíncrono y orientado a eventos, para así mantenerlo liviano y eficiente. Además, Node es multiplataforma y cuenta con su propio gestor de paquetes llamado NPM (Node Package Manager), que permite gestionar fácilmente las dependencias de un proyecto. Otra característica importante es que está diseñado a para crear aplicaciones network escalables y es especialista en aplicaciones RealTime.

De acuerdo con la plataforma de LucusHost (LucusHost, 2022), Node se creó como solución para poder ejecutar JavaScript en el lado del servidor. Está basado en el motor V8 de Google, que es uno de los intérpretes de lenguaje de programación que existen y, permite compilar el código JS en código de máquina, por tal motivo, Node es uno de los entornos de ejecución con más crecimiento y el preferido para el desarrollo de aplicaciones web o de escritorio.

1.9 Metodología de Desarrollo SCRUM

Como definición de Scrum la propia Guía de Scrum 2020 (Schwaber & Sutherland, 2020) menciona que, Scrum es un marco de trabajo liviano que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos. Donde se requiere que un Scrum Master fomente un ambiente donde un Product Owner elabora un Product Backlog bien detallado, permitiendo al Scrum team convertir una selección de trabajo en un Incremento de valor durante un Sprint, dicho proceso se repetirá las veces que sean necesarias.

De acuerdo con la definición de (Srivastava et al., 2017), Scrum es básicamente un marco ágil y ligero que proporciona pasos para gestionar y controlar el proceso de desarrollo de software y productos. Scrum es el resultado de la combinación del modelo iterativo y el modelo incremental porque las construcciones son sucesivas e incrementales en cuanto a las características para desarrollar software orientado a objetos.

Scrum al ser un marco de trabajo ágil, comparte junto con DevOps la flexibilidad de adaptación a cambios y mejorar el rendimiento del desarrollo a través de entregas frecuentes e iteraciones de trabajo, con el fin de fomentar una cultura de trabajo mejorada en base a los principios del agilismo. Por lo tanto, Scrum es una pieza fundamental para el desarrollo del presente trabajo.

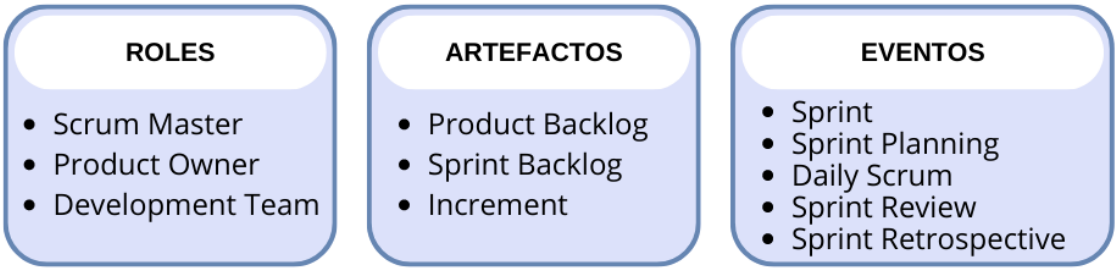

Figura 34. Elementos de Scrum

De acuerdo con las especificaciones detalladas de la Guía de Scrum, se describen los elementos que encontramos en la Figura 34.

1.9.1 Roles

Scrum Master

El Scrum Master es el experto y responsable de establecer Scrum como se define en la Guía de Scrum. Se encarga de proveer los recursos necesarios al equipo de desarrollo y solventar sus problemas de su entorno de trabajo, logrando alcanzar la mayor efectividad de rendimiento, a su vez, ayuda a comprender la teoría y práctica del marco de trabajo dentro del Scrum Team como de la organización. En general los Scrum Masters son verdaderos líderes que sirven al Scrum Team y a la organización (Schwaber & Sutherland, 2020).

Product Owner

El Product Owner es el responsable de maximizar el valor del producto resultante del trabajo del Scrum Team. Este proceso puede variar de acuerdo con la organización o el equipo Scrum. Igualmente, el Product Owner tiene la responsabilidad de la gestión del Product Backlog,

es decir, comunicar el objetivo del producto y desarrollar claramente los elementos del Product Backlog, que deben ser entendidos y acatados por el resto del equipo. El Product Owner puede delegar responsabilidades de su trabajo a otros, tomando en cuenta que él es el único responsable del Product Backlog (Schwaber & Sutherland, 2020).

Development Team

Son los encargados de desarrollar cualquier avance de un increment utilizable en cada sprint, demostrando sus habilidades que pueden variar según el ámbito de trabajo. Su principal responsabilidad es definir el plan para cada Sprint y presentar cada iteración con una revisión final (Schwaber & Sutherland, 2020).

1.9.2 Artefactos

Product Backlog

Es una lista emergente y ordenada de lo que se necesita para mejorar el producto. Es la única fuente del trabajo realizado por el Scrum Team, en otras palabras, el Product Backlog contiene pautas de desarrollo, donde encontramos los requerimientos detallados para dar cumplimiento con el objetivo del producto (Schwaber & Sutherland, 2020).

Sprint Backlog

El Sprint Backlog es un plan realizado para los desarrolladores, donde contiene detalladamente que actividades deben realizarse en el tiempo establecido de cada Sprint para cumplir con su objetivo. Si el trabajo resulta ser diferente de lo planificado en el Sprint Planning, los desarrolladores deben comunicarse con el Product Owner para replanificar el alcance del Sprint Backlog sin afectar el Objetivo del Sprint. Este Sprint Backlog está compuesto por el Objetivo del Sprint que responde al por qué, los elementos delimitados responden al qué y el plan de acción de entrega del increment responde al cómo (Schwaber & Sutherland, 2020).

Increment

Un increment es una pequeña funcionalidad que busca concretar en el objetivo del producto. Dicho objetivo puede establecerse como la suma de todos los increments validados y funcionales. Para tomar en cuenta un increment este debe encontrarse en el estado de terminado, habiendo cumplido con las medidas de calidad requeridas para el producto. Si un elemento del Product Backlog no cumple con la definición de terminado, no se puede publicar ni presentar en la Sprint Review, de tal manera que vuelve al Product Backlog para su consideración futura (Schwaber & Sutherland, 2020).

1.9.3 Eventos

El Sprint

Cada Sprint es un evento de duración fija de máximo un mes y mínimo una semana para crear consistencia. Todo el trabajo necesario para lograr el objetivo del producto, incluido la Sprint Planning, Daily Scrums, Sprint Review y Sprint Retrospective, ocurre dentro de los Sprints. Cada Sprint permite la previsibilidad al garantizar la inspección y adaptación del progreso hacia un objetivo del producto, ya que durante el Sprint no se debe realizar cambios que pongan en peligro el objetivo del Sprint, por lo cual la calidad no se ve afectada y, en caso de que las actividades planificadas no puedan cumplirse en el tiempo acordado, se puede refinar el Product Backlog y renegociar con el Product Owner (Schwaber & Sutherland, 2020).

Sprint Planning

De acuerdo a lo mencionado por (Schwaber & Sutherland, 2020), la Sprint Planning es el resultado del trabajo colaborativo del Scrum Team e inicia al establecer el trabajo que se realizará durante el tiempo establecido de cada iteración. El Sprint Planning aborda los siguientes temas:

- **¿Por qué es valioso este Sprint?:** el Product Owner establece como puede mejorar el valor del Sprint y el equipo colabora para definir el objetivo que comunica el por qué el Sprint es valioso para los interesados.
- **¿Qué se puede hacer en este Sprint?:** en este tema el equipo de desarrollo se encarga de seleccionar los elementos del Product Backlog para incluirlos en el Sprint Actual y establecer las ponderaciones de estimación.
- **¿Cómo se realizará el trabajo elegido?:** para cada elemento del Product Backlog seleccionado, los desarrolladores planifican el trabajo necesario para crear un increment que cumpla con la definición de terminado.

El Sprint Planning tiene un límite de tiempo de máximo ocho horas para un Sprint de un mes y, para Sprints más cortos, el evento suele ser de menor duración.

Daily Scrum

Es una revisión diaria al trabajo realizado cuyo propósito es inspeccionar el progreso hacia el objetivo del Sprint. Este evento se lo aplica al equipo de desarrollo y tiene una duración de no más de 15 minutos, y el Product Owner y Scrum Master pueden formar parte de esta actividad siempre y cuando estén trabajando activamente en elementos del Sprint Backlog. Los Daily Scrum mejoran la comunicación, identifican impedimentos, promueven la toma rápida de decisiones y, en consecuencia, eliminan la necesidad de otras reuniones (Schwaber & Sutherland, 2020). Generalmente en estas reuniones se deja en conocimiento a las preguntas: ¿qué hice ayer?, ¿qué voy a hacer hoy? y ¿qué impedimentos tengo?

Sprint Review

Es el penúltimo evento del Sprint donde el Scrum Team debe evitar limitarla a una presentación, teniendo un límite de tiempo de máximo 4 horas para un Sprint de un mes y, para Sprints más cortos, el evento es de menor duración. El propósito del Sprint Review es inspeccionar el resultado del Sprint, presentar a los interesados clave y determinar futuras adaptaciones, donde el Product Backlog puede sufrir nuevas modificaciones (Schwaber & Sutherland, 2020).

Sprint Retrospective

El propósito de la Sprint Retrospective es planificar formas de aumentar la calidad y la efectividad. Se identifican los supuestos que los llevaron por mal camino y se exploran sus orígenes. El Scrum Team analiza qué salió bien durante el Sprint, qué problemas encontró y cómo se resolvieron esos problemas. Este evento finaliza con el Sprint y suele tener un tiempo limitado de máximo 3 horas para un Sprint de un mes y, para Sprints más cortos, el evento es de menor duración (Schwaber & Sutherland, 2020).

CAPÍTULO II

2 Desarrollo

En el Capítulo 2, se presenta el proceso para definir el diseño una arquitectura DevOps utilizando las fases 3 y 4 del Método de Desarrollo de Arquitectura (ADM) del framework de TOGAF. Además, se detalla el desarrollo de una prueba de concepto de la arquitectura DevOps definida mediante la automatización de los módulos (consulta de cuentas y transferencias) utilizando el marco de trabajo Scrum, con el fin de mostrar el procedimiento del ciclo de vida de DevOps. A continuación, la Figura 35 muestra la estructura del capítulo de desarrollo.

Figura 35. Estructura Capítulo 2

2.1 Análisis

2.1.1 Equipo Scrum

Para el presente proyecto se definen los siguientes roles definidos en Scrum, los cuales se muestran en la Tabla 4.

Tabla 4. Equipo Scrum

Rol	Nombre	Responsabilidad
Product Owner	PhD. José Antonio Quiña Mera	Verificar los avances y funcionalidades del proyecto
Stakeholders	PhD. Irving Reascos Ing. Alexander Guevara	
Scrum Master	José Luis Parra Vite	Liderar el proyecto aplicando correctamente Scrum
Equipo de Desarrollo	José Luis Parra Vite	Desarrollar los requerimientos del proyecto

2.1.2 Definición de Requisitos

Para definir los requisitos del sistema se utiliza las Historias de Usuario (HU) correspondientes al marco de trabajo Scrum, donde se detalla de manera objetiva los requisitos, proponiendo formar una arquitectura que tenga las capacidades de soportar aplicaciones orientadas a microservicios tipo REST, utilizando contenedores Docker para facilitar la portabilidad de la aplicación y cumpliendo el ciclo de vida de DevOps. Es importante mencionar que cada HU tiene su propio identificador, responsable y detalles que permitan a los desarrolladores entender el objetivo y el alcance de la Historia de Usuario.

En el marco de este proyecto, se ha decidido desarrollar todos los requisitos en un backend utilizando la tecnología NestJs, procurando desarrollar una Interfaz de Programación de Aplicaciones robusta y funcional para gestionar la lógica de negocio con la base de datos.

Historia de Usuario 1

Historia de Usuario			
ID:	**HU-01**	**Usuario:** Desarrollador	
Nombre: Configuración de ambiente de desarrollo			
Prioridad: Alta		**Dependencia:** N/A	**Estimación:** 15 h
Responsable: José Luis Parra Vite			
Descripción: Como desarrollador quiero tener una configuración estándar y ordenada de mi ambiente de desarrollo para no tener dificultades o retrasos en el desarrollo			
Pruebas de aceptación: • Al levantar los contenedores Docker no se debe tener dificultades o errores de ejecución • Al levantar la Base de Datos PostgreSQL debe ejecutarse dentro de un contenedor • Al levantar el servicio de PgAdmin4 se debe visualizar el contenido de la Base de Datos PostgreSQL			

Historia de Usuario 2

Historia de Usuario		
ID:	**HU-02**	**Usuario:** Administrador

Nombre: Registrar nuevo Usuario		
Prioridad: Alta	**Dependencia:** 1	**Estimación:** 20 h
Responsable: José Luis Parra Vite		
Descripción: Como administrador quiero que el sistema cuente con un método que permita registrar un nuevo usuario asignado a uno o varios roles y que su identificador único sea la cédula, para administrar registros del sistema.		
Pruebas de aceptación: • Al guardar un nuevo usuario con campos vacíos o extras, se debe mostrar un mensaje de error. • Al guardar un nuevo usuario con un identificador existente (cédula), mostrar un mensaje de error. • Al guardar un nuevo usuario con los datos válidos, mostrar los datos del usuario registrado.		

Historia de Usuario 3

Historia de Usuario			
ID:	**HU-03**	**Usuario:** Administrador	
Nombre: Consultar todos los Usuarios y Usuario determinado			
Prioridad: Alta		**Dependencia:** 1,2	**Estimación:** 20 h
Responsable: José Luis Parra Vite			
Descripción: Como administrador quiero tener la facilidad de poder consultar todos los usuarios registrados, además de poder buscar más información de un usuario en específico, para tener una visualización de los registros de usuarios registrados en la base de datos.			
Pruebas de aceptación: • Al consultar el listado general de usuario se debe mostrar únicamente información básica de cada usuario como; identificador (cédula), nombres, apellidos, entre otros. • Al ingresar el ID de un usuario que no existe, mostrar un mensaje de error • Al ingresar correctamente el ID de un usuario, mostrar la información detallada y los roles asignados.			

Historia de Usuario 4

Historia de Usuario			
ID:	**HU-04**	**Usuario:** Cliente	
Nombre: Consultar Cuentas por Usuario			
Prioridad: Alta		**Riesgo:** Medio	**Estimación:** 30 h
Responsable: José Luis Parra Vite			
Descripción: Como cliente quiero que el sistema permita realizar la consulta de todas las cuentas que están vinculadas a mi usuario con el tipo de cuenta respectivo y su saldo actual de las cuentas activas y saldos en tiempo real, para visualizar el estado de cuenta.			
Pruebas de aceptación: • Al ingresar el ID de un usuario que no existe, mostrar un mensaje de error • Al ingresar correctamente el ID de un usuario, mostrar las cuentas registradas a dicho usuario con el tipo de cuenta y su saldo actual			

Historia de Usuario 5

Historia de Usuario			
ID:	**HU-05**	**Usuario:** Cliente	
Nombre: Consultar Movimientos por Cuenta de Usuario			
Prioridad: Alta		**Riesgo:** Medio	**Estimación:** 30 h
Responsable: José Luis Parra Vite			
Descripción: Como cliente quiero que el sistema muestre información de los movimientos realizados por una cuenta registrada a mi usuario y que contenga información detallada de la cuenta, el tipo de movimientos con su cantidad e información de mi usuario, para visualizar el estado de cuenta.			
Pruebas de aceptación: • Al ingresar el ID de un usuario no registrado, mostrar un mensaje de error • Al ingresar el ID de una cuenta que no existe, mostrar un mensaje de error • Al ingresar los datos correctamente, mostrar la información de la cuenta, el tipo de movimientos con su cantidad e información del usuario			

2.1.3 Product Backlog

En la Tabla 5 se muestra la información simplificada de las Historias de Usuario descritas anteriormente, agrupadas por el Product Owner quien las prioriza y define un orden para desarrollarlas de acuerdo con sus necesidades, a esto se le denomina Product Backlog.

Tabla 5. Product Backlog

Orden	ID	Descripción	Estimación (horas)
1	HU-01	Configuración de ambiente de desarrollo	20
2	HU-02	Registrar nuevo Usuario	20
3	HU-03	Consultar todos los Usuarios y Usuario determinado	20
4	HU-04	Consultar Cuentas por Usuario	30
5	HU-05	Consultar Movimientos por Cuenta de Usuario	30

2.2 Diseño – Sprint 0

Para la parte del Diseño del proyecto realizada en la iteración del Sprint 0, se hace énfasis en el desarrollo del diseño de la arquitectura empleando las fases 3 (Arquitectura de Sistemas de Aplicación) y 4 (Arquitectura Tecnológica) del ADM del framework TOGAF, en donde se utiliza el lenguaje de modelado Archimate mediante la herramienta ArchiMate Modelling Tool.

2.2.1 Planificación

En la planificación para el Sprint 0 realizada la fecha 6/12/2022, en presencia de Scrum Master, Development Team y el Product Owner, se aclaró las actividades a realizarse en las iteraciones, planteando el diseño arquitectónico respetando el marco principal de ArchiMate, que consta de tres aspectos y tres capas mostrados en la Figura 30 y del desarrollo del sistema con las tecnologías propuestas.

Cada sprint tiene una duración máxima de 2 semanas cada uno. En la Tabla 6 se muestra la fecha de inicio y fin de cada sprint junto con el tiempo de horas estimadas que tomará para culminar cada iteración.

Tabla 6. Resumen de los Sprints

Sprint	Fecha inicio	Fecha fin	Duración (Horas)
Sprint 0	06/12/2022	19/12/2022	30
Sprint 1	20/12/2022	26/12/2022	20
Sprint 2	27/12/2022	09/01/2022	42
Sprint 3	10/01/2022	23/01/2023	42
Sprint 4	24/01/2023	06/02/2023	42

Las fases de TOGAF a utilizar, se componen del diseño de la Arquitectura de Datos y Aplicaciones, correspondientes a la fase C (Arquitectura de Sistemas de Información) y del diseño de la Arquitectura Tecnológica correspondiente a la fase D del ADM. Teniendo como herramienta de diseño a Archimate que nos ofrece pautas de diseño para diagramar los diferentes componentes en base a los requisitos presentados.

Modelo de Datos

En la Figura 36 se presenta el modelo de datos, que cumple con la función de mapear de manera automática y bidireccional los objetos de una aplicación con las tablas y columnas de una base de datos, simplificando así el acceso y manipulación de los datos, agregando así una capa extra de seguridad para la integridad de los datos.

Figura 36. ORM para mapear la base de datos

Diagrama de Flujo

En la Figura 37 se describe el flujo de proceso que realiza el sistema.

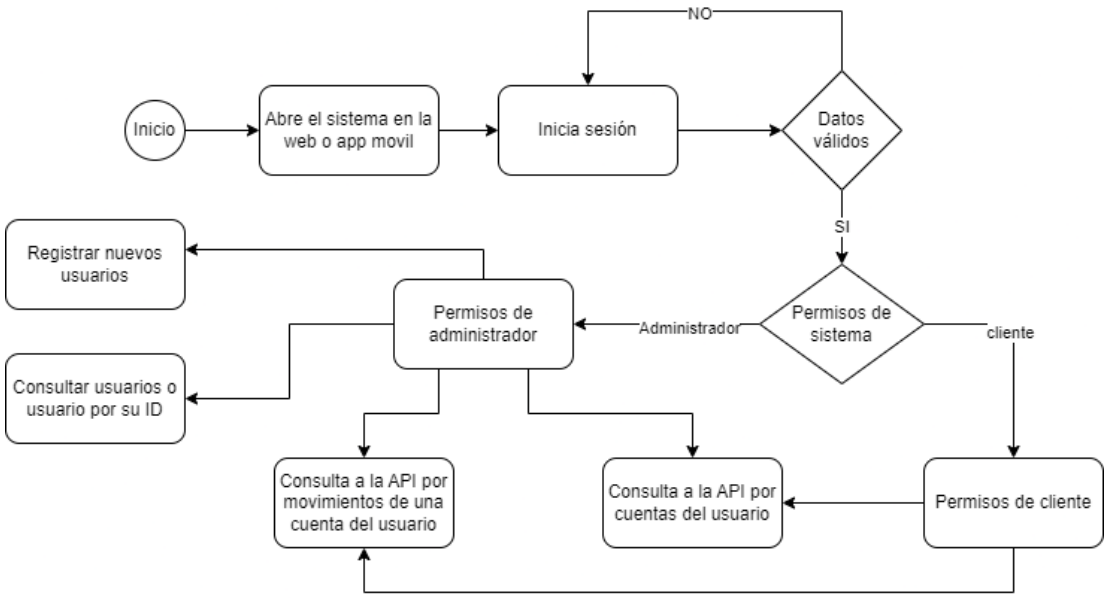

Figura 37. Diagrama de flujo

2.2.2 Diseño de las Capas del Core de TOGAF

Tomando en cuenta las fases del ADM de TOGAF, se optó por utilizar las 2 principales capas que componen el núcleo de este método, que, a su vez, se compacta con el Core de ArchiMate. Siendo las Capa de Aplicación y Tecnológica, las bases para diseñar la arquitectura objetivo del proyecto.

2.2.2.1 Arquitectura de Datos

A continuación, se muestra la propuesta de la arquitectura de datos como se observa en la Figura 38, basada en una vista lógica del proceso de realizar peticiones a la API REST.

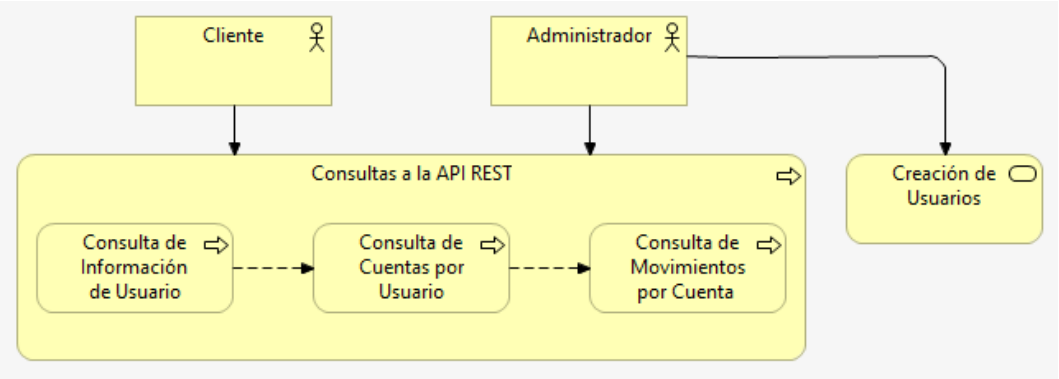

Figura 38. Capa de Negocio

2.2.2.2 Arquitectura de Aplicación

Para la arquitectura de aplicaciones, se mencionó que se utilizará un entorno localhost, y a su vez se usa el repositorio de código de la nube y se levanta un servidor de Jenkins que puede construirse mediante una imagen Docker o directamente en un servidor propio o en la nube. En la Figura 39 se presenta el diseño propuesto.

Figura 39. Capa de Aplicación

2.2.2.3 Arquitectura Tecnológica

Como se puede observar en la Figura 40, se detalla los módulos tecnológicos y su respectiva relación y funcionamiento.

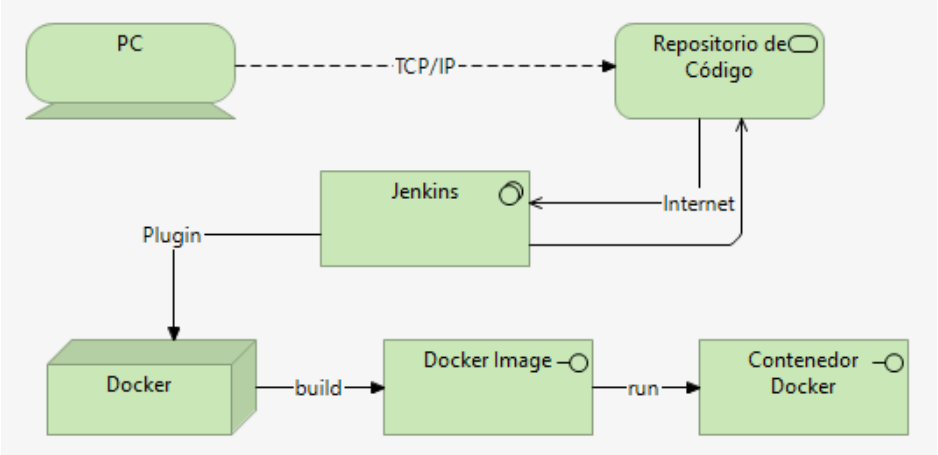

Figura 40. Capa Tecnológica

2.2.3 Esquema de Base de Datos

El motor de Base de Datos Relacional utilizado es PostgreSQL, adicionalmente para el ambiente de desarrollo se optó por emplear el visualizador PgAdmin4, que nos permite tener un mejor control del esquema de la Base de Datos mediante una interfaz gráfica muy intuitiva y facilita la creación del Diagrama de Entidad Relación de la Base de Datos, mostrándonos de una forma gráfica y clara como se relacionan las tablas entre sí. En la Figura 41, se muestra el diseño de la Base de Datos finalizado de acuerdo con los requerimientos.

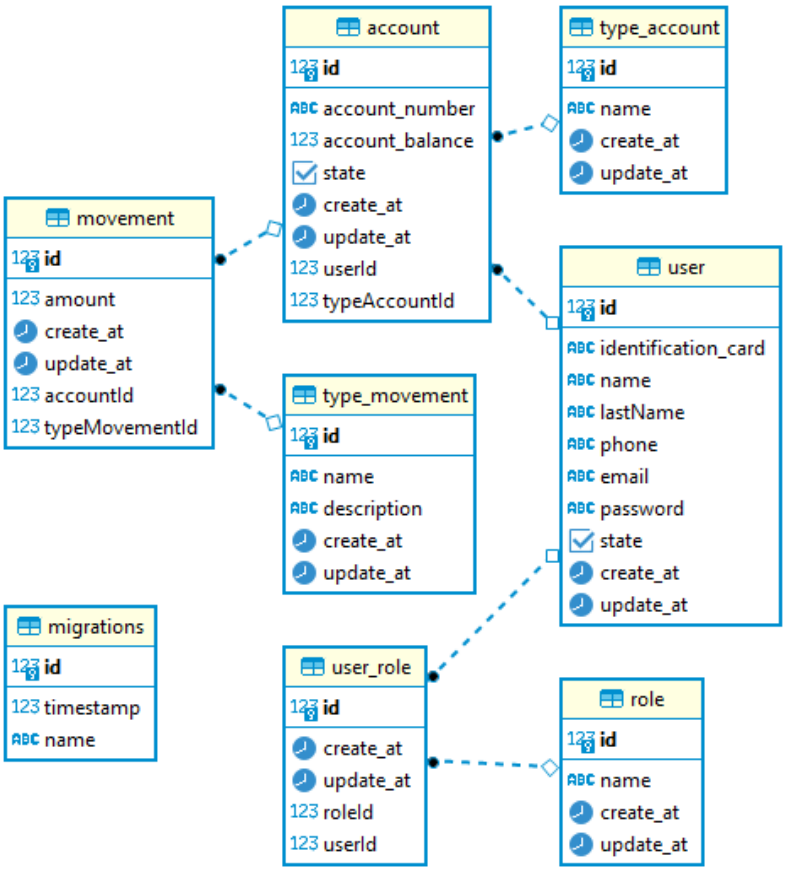

Figura 41. Diagrama de Entidad-Relación de la Base de Datos

En el esquema de Base de Datos se puede observar las tablas con sus respectivas relaciones para cumplir con los requisitos planteados en el análisis del proyecto, pudiendo realizar niveles de consulta hasta de cinto tablas.

Adicionalmente se muestra una tabla llamada “migrations”, la cual sirve para llevar un control de los cambios en el modelo inicial de Base de Datos, dicha tabla se crea a partir de las entidades creadas en el proyecto con TypeORM de NestJS para montar consultas y realizar operaciones con los datos.

2.3 Desarrollo de módulos (Cuentas y Transferencias)

Para el desarrollo del módulo del Sistema Financiero se utilizó NestJs, que es el framework de NodeJs para desarrollar proyectos con arquitectura escalable y modular. Se creará un microservicio backend de tipo RESTful (API REST), siguiendo patrones de diseño y buenas prácticas como: arquitectura modular, principio SOLID, integrabilidad de datos, Inyección de Dependencias y documentación automática de la API mediante la librería Swagger.

Además, se utiliza TypeORM, que es una librería de TypeScript que proporciona un enfoque para trabajar con bases de datos SQL y NoSQL, siendo altamente compatible con PostgreSQL, permitiendo definir modelos de datos y realizar operaciones sin la necesidad de escribir código SQL explícito. Adicionalmente permite realizar transacciones, migraciones de base de datos, caché, validaciones de modelos, entre otras cosas.

2.3.1 Sprint 1

En el Sprint 1 se realizó la creación del proyecto, configuración del ambiente de desarrollo con Docker, creando volúmenes de persistencia y contenedores con el archivo docker-compose.yml, el cual contiene los servicios de la Base de Datos y el visualizador PgAdmin4. En la Tabla 7 se presenta las actividades propuestas para la actual iteración que se detallan en el Sprint Backlog de la Tabla 8.

- **Reunión de planificación – Sprint 1**

Tabla 7. Reunión de la planificación – Sprint 1

Sprint 1	
Fecha de la reunión:	20/12/2022
Asistentes:	Scrum Master, Product Owner y Team
Resultado	Sprint Backlog – Sprint 1
Objetivos del sprint: • Configurar ambiente de desarrollo, usando el editor de texto VSCode y extensiones para facilitar el desarrollo. Habilitar puertos de trabajo • Creación del proyecto con NestJs • Configurar Docker para levantar un contenedor con una base de datos Postgres • Configurar la base de datos Postgres • Configurar y conectar el proyecto con la base de datos Postgres y validar variables de entorno	

- ✓ **Sprint Backlog – Sprint 1**

Tabla 8. Sprint Backlog – Sprint 1

Nro. HU	Nombre	Tarea	Horas
HU-01		Instalación de Editor de código	1

	Configuración de ambiente de desarrollo	Instalación y configuración de Docker para crear contenedores	4
		Levantamiento del servicio de base de datos Postgres con Docker	4
		Creación del Proyecto con NestJs	1
		Conexión del Proyecto con la base de datos Postgres	2
		Implementación y configuración de variables de ambiente para el proyecto	2
		Conexión del proyecto con un repositorio remoto en GitHub	1

- **Reunión de revisión – Sprint 1**

Finalizado el Sprint 1, se realiza la reunión de revisión con todos los responsables del proyecto, revisando el avance y verificando el cumplimiento de la configuración total del ambiente de desarrollo del proyecto, con todas las herramientas necesarias. En la Tabla 9 se muestran las tareas con su actual estado conforme a la planificación.

Tabla 9. Reunión de Revisión – Sprint 1

Responsable	Tarea	Tiempo estimado (horas)	Tiempo real (horas)	Estado
Desarrollador	Instalación de Editor de código	2	4	Realizado
	Instalación y configuración de Docker para crear contenedores	4	1	Realizado
	Levantamiento del servicio de base de datos Postgres con Docker	4	4	Realizado
	Creación del Proyecto con NestJs	2	2	Realizado
	Conexión del Proyecto con la base de datos Postgres	4	4	Realizado

	Implementación y configuración de variables de ambiente para el proyecto	3	4	Realizado
	Conexión del proyecto con un repositorio remoto en GitHub	1	1	Realizado
Reuniones Scrum	Planificación	1	1	Realizado
	Revisión	1	1	Realizado
	Retrospectiva	1	1	Realizado
	Total	24	23	

✓ **Incremento**

En esta iteración del proyecto, se levantó el servicio de una base de datos PostgreSQL en un contenedor de Docker, junto con el visualizador de base de datos Postgres PgAdmin4 y, la creación del proyecto y conexión exitosa entre la base de datos y el proyecto de NestJs.

- **Archivo Docker Compose con los servicios de Base de Datos y Jenkins**

Se definen todos los servicios a utilizar en un archivo docker-compose.yml, permitiendo administrar las fases del ciclo de vida de un servicio definido (iniciar, detener, generar servicios, revisar el estado y transmitir registros). Estos servicios usan un archivo llamado Dockerfile o una imagen existente cargada en un repositorio remoto o local. En la Figura 42 se puede observar los servicios utilizados.

```yaml
version: '3.3'

services:
  postgres:
    image: postgres:13
    container_name: postgres
    environment:
      - POSTGRES_DB=finance_db
      - POSTGRES_USER=root
      - POSTGRES_PASSWORD=123456
    ports:
      - "5432:5432"
    volumes:
      - postgres_db:/var/lib/postgresql/data
    networks:
      - FSystem-net
    restart: unless-stopped

  pgadmin:
    image: dpage/pgadmin4
    container_name: pgadmin
    environment:
      - PGADMIN_DEFAULT_EMAIL=ejemplo@gmail.com
      - PGADMIN_DEFAULT_PASSWORD=123456
    ports:
      - "5050:80"
    volumes:
      - pgadmin_db:/var/lib/pgadmin
    networks:
      - FSystem-net
    depends_on:
      - postgres
    restart: unless-stopped

  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins
    privileged: true
    user: root
    ports:
      - "8080:8080"
      - "50000:50000"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - jenkins_vol:/var/jenkins_home
    networks:
      - FSystem-net
    restart: unless-stopped

volumes:
  postgres_db:
  pgadmin_db:
  jenkins_vol:

networks:
  FSystem-net:
```

Figura 42. Archivo docker-compose.yml

- **Servicios de Postgres, PgAdmin y Jenkins levantados correctamente**

Se inician los servicios con docker compose, donde primero se descarga la imagen de contenedor en caso de no tenerla descargada y luego si no existe ningún error se levantan correctamente los servicios. En la Figura 43 se puede observar que los servicios se encuentran levantados correctamente.

```
~/Documentos/JL-WEB/Curso-NestJS  dev !11 ?1                                    joseluis@Jose-PCL
❯ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS          PORTS
                NAMES
6a1f9f853fab   jenkins/jenkins:lts  "/usr/bin/tini -- /u…"   11 hours ago   Up 40 minutes   50000/tcp, 0.0.0.0:8082->8080/tcp, :
::8082->8080/tcp   jenkins
dd2ac2d262c0   dpage/pgadmin4       "/entrypoint.sh"         13 days ago    Up 2 minutes    443/tcp, 0.0.0.0:5050->80/tcp, :::50
50->80/tcp         pgadmindb
8601a8af327f   postgres:13          "docker-entrypoint.s…"   13 days ago    Up 2 minutes    0.0.0.0:5432->5432/tcp, :::5432->543
2/tcp              postgresdb
```

Figura 43. Servicios levantados y en ejecución

- **Comprobación del servicio de PgAdmin conectado con PostgreSQL**

Para comprobar que el servicio se inició correctamente, ingresamos en un navegador para iniciar sesión en PgAdmin y conectarnos a la base de datos de PostgreSQL, como se muestra en la Figura 44.

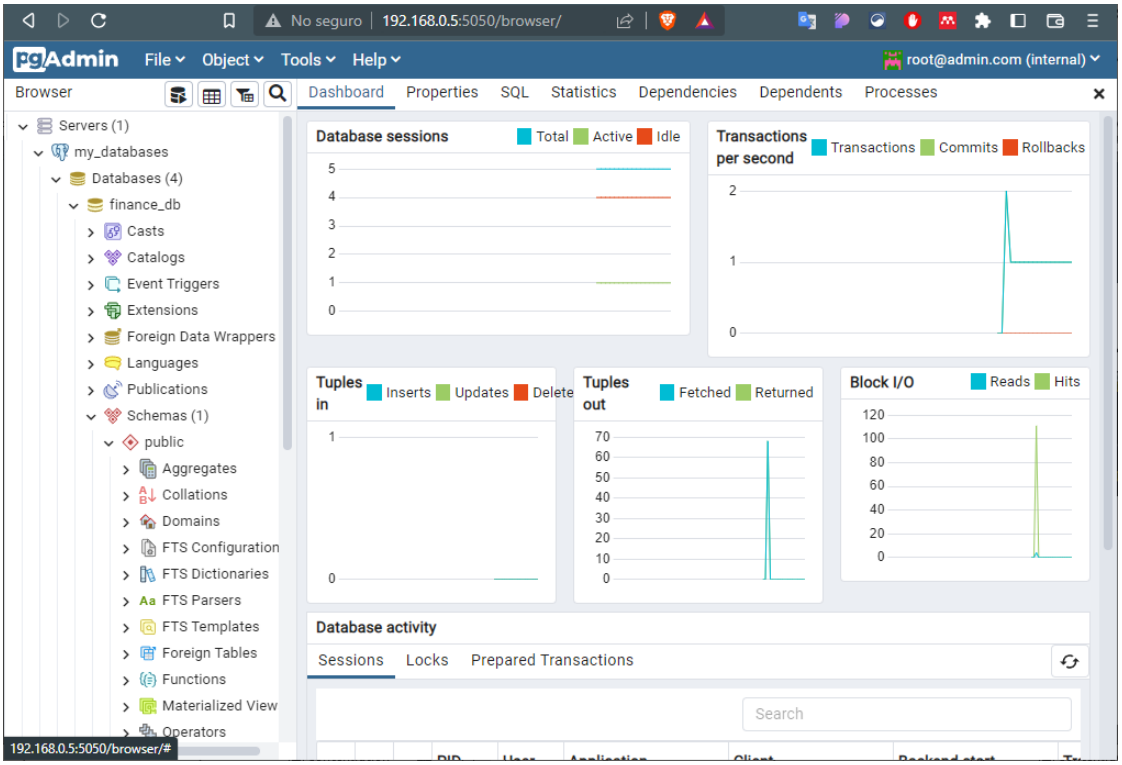

Figura 44. Servicio de Postgres y PgAdmin en ejecución

- **Creación del proyecto**

Después de levantar los servicios, podemos crear el proyecto en NestJs, siguiendo la estructura por defecto para el desarrollo de la aplicación como se observa en la Figura 45.

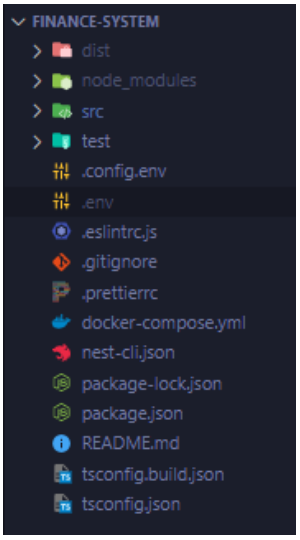

Figura 45. Estructura del proyecto

2.3.2 Sprint 2

En el Sprint 2 se desarrolla la funcionalidad de insertar y consultar registros de la tabla usuario mediante la arquitectura de microservicios REST, implementando una capa de validación a los atributos que recibe y a los parámetros de búsqueda. Se realizó la documentación automática de las rutas de la API REST generada por la librería **Swagger**. En la tabla 11 se presenta a detalle las actividades que deben cumplirse de acuerdo con los objetivos planteados en la reunión de planificación de la Tabla 10.

- **Reunión de planificación – Sprint 2**

Tabla 10. Reunión de la planificación – Sprint 2

Sprint 2	
Fecha de la reunión:	27/12/2022
Asistentes:	Scrum Master, Product Owner y Team
Resultado	Sprint Backlog – Sprint 2
Objetivos del sprint: • Implementar un servicio que permita registrar usuarios	

- Implementar la funcionalidad para consultar todos los usuarios registrados y realizar una consulta más detallada de un usuario en específico.
- Agregar una capa de validación y control de mensajes de error
- Ejecutar los servicios de inserción y búsqueda de usuarios en el entorno Localhost desarrollado en API REST

✓ **Sprint Backlog – Sprint 2**

Tabla 11. Sprint Backlog – Sprint 2

Nro. HU	Nombre	Tarea	Horas
HU-02	Insertar usuarios	Construir un servicio que permita insertar usuarios	4
		Crear la Entity de Usuario para mapear a la base de datos	3
		Crear el DTO para manipular la información que recibe la tabla usuario	3
		Crear el servicio para usuario con sus respectivas funciones	4
		Crear el componente controlador y añadir el método para consumir el servicio	3
		Ejecución del servicio insertar usuario	2
HU-03	Consulta usuarios	Construir un servicio que permita la consulta de usuarios	3
		Realizar Inyección de dependencias en el controlador utilizando el servicio	3
		Definir en el controlador las rutas para realizar las búsquedas	2
		Ejecución del servicio consultar usuario	2

➢ **Reunión de revisión – Sprint 2**

Una vez concluido el Sprint 2, se realiza la respectiva reunión de revisión con los involucrados. Como resultado se obtuvo la verificación de la funcionalidad de los requerimientos

establecidos para esta iteración. En la Tabla 12 muestra las tareas realizadas conforme a la planificación.

Tabla 12. Reunión de Revisión - Sprint 2

Responsable	Tarea	Tiempo estimado (horas)	Tiempo real (horas)	Estado
Desarrollador	Construir un servicio que permita insertar usuarios	3	4	Realizado
	Crear la Entity de Usuario para mapear a la base de datos	3	3	Realizado
	Crear el DTO para manipular la información que recibe la tabla usuario	3	3	Realizado
	Crear el servicio para usuario con sus respectivas funciones	4	4	Realizado
	Crear el componente controlador y añadir el método para consumir el servicio	3	3	Realizado
	Ejecución del servicio insertar usuario	2	2	Realizado
	Construir un servicio que permita la consulta de usuarios	3	3	Realizado
	Realizar Inyección de dependencias en el controlador utilizando el servicio	2	3	Realizado
	Definir en el controlador las rutas para realizar las búsquedas	2	2	Realizado
	Ejecución del servicio consultar usuarios	2	2	Realizado
Reuniones Scrum	Planificación	1	1	Realizado
	Revisión	1	1	Realizado
	Retrospectiva	1	1	Realizado
	Total	30	32	

✓ **Incremento**

En esta iteración se obtuvo como resultado la implementación de los servicios insertar y consultar usuarios, desplegados en el ambiente de desarrollo localhost.

- **Funcionalidad Servicio crear Usuario**

Se realiza la prueba para insertar un usuario como se observa en la Figura 46.

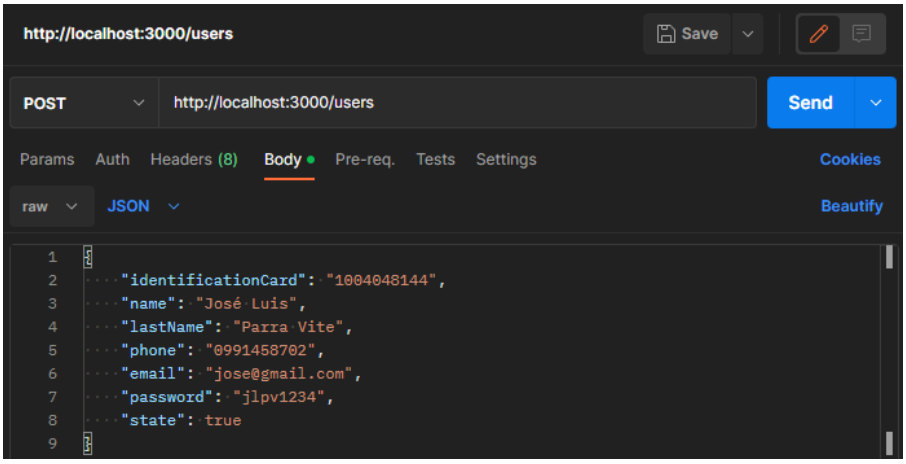

Figura 46. Registrar un nuevo usuario

- **Funcionalidad Servicio Consultar Usuarios**

Se prueba que el servicio permita consultar todos los usuarios que fueron registrados en el sistema, como se observa en la Figura 47.

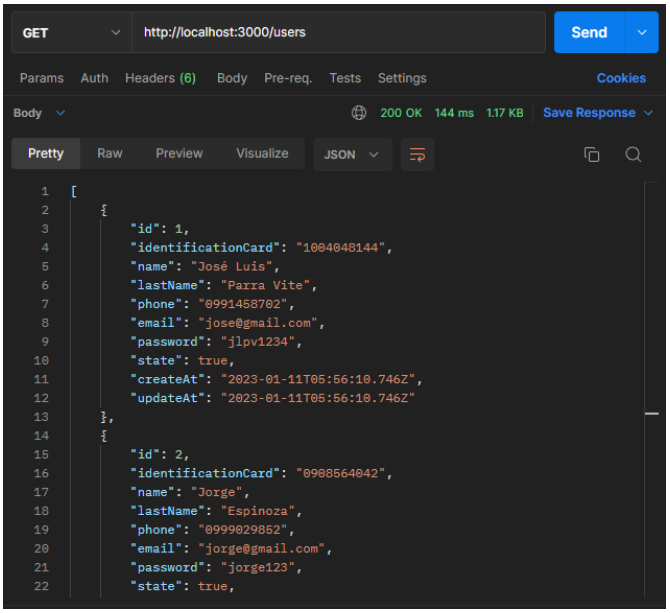

Figura 47. Consultar todos los usuarios registrados

- **Servicio para Consultar un Usuario específico**

De la misma forma, se realiza la prueba de funcionalidad de consulta un usuario mediante su ID. Ver Figura 48.

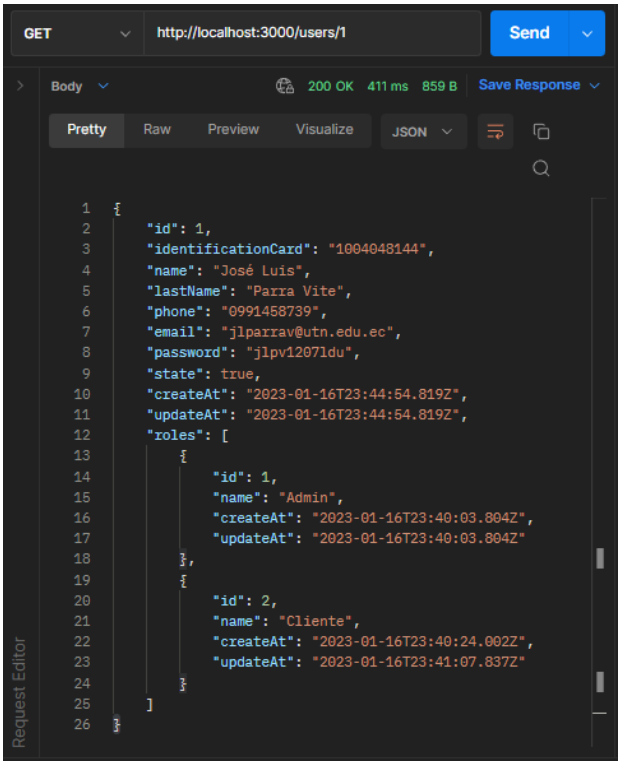

Figura 48. Consultar usuario por ID

2.3.3 Sprint 3

En el Sprint 3 se desarrolla la funcionalidad de consultar cuentas por usuario mediante la arquitectura REST. Se agrega la capa de validación de los parámetros de búsqueda y las advertencias en caso de cometer algún error en la petición al servicio. La Tabla 13 establece los objetivos del Sprint que posteriormente se desglosan en actividades para el Sprint Backlog que muestra la Tabla 14.

- **Reunión de planificación – Sprint 3**

Tabla 13. Reunión de planificación – Sprint 3

Sprint 3	
Fecha de la reunión:	10/01/2022
Asistentes:	Scrum Master, Product Owner y Team
Resultado	Sprint Backlog – Sprint 3

Objetivos del sprint:

- Implementar la funcionalidad a la aplicación que permita realizar la consulta de cuentas por usuario
- Agregar una capa de validación y control de mensajes de error
- Ejecutar los servicios de consulta de usuarios por cuenta en el entorno Localhost desarrollado en REST

✓ **Sprint Backlog – Sprint 3**

Tabla 14. Sprint backlog - Sprint 3

Nro. HU	Nombre	Tarea	Horas
HU-04	Consulta cuentas por usuario	Crear las Entidades necesarias para mapear a la base de datos	4
		Crear los DTO para manipular la información que reciben las tablas	4
		Construir un servicio que permita la consulta de cuentas por usuario	4
		Añadir una capa de validación a los parámetros de búsqueda	3
		Añadir el controlador para que se comunique con el servicio	4
		Ejecución del servicio consultar cuentas por usuario en REST	3

➢ **Reunión de revisión – Sprint 3**

Finalizando el Sprint 3 se reúnen los involucrados para hacer la respectiva revisión de las actividades que fueron definidas, obteniendo la comprobación del funcionamiento de cada actividad del backlog, teniendo la estimación en horas y su estado final como se puede observar en la Tabla 15.

Tabla 15. Reunión de revisión - Sprint 3

Responsable	Tarea	Tiempo estimado (horas)	Tiempo real (horas)	Estado

Desarrollador	Crear las Entidades necesarias para mapear a la base de datos	3	4	Realizado
	Crear los DTO para manipular la información que reciben las tablas	3	4	Realizado
	Construir un servicio que permita la consulta de cuentas por usuario	4	4	Realizado
	Añadir una capa de validación a los parámetros de búsqueda	2	3	Realizado
	Añadir el controlador para que se comunique con el servicio	2	4	Realizado
	Ejecución del servicio consultar cuentas por usuario en REST	4	3	Realizado
Reuniones Scrum	Planificación	1	1	Realizado
	Revisión	1	1	Realizado
	Retrospectiva	1	1	Realizado
	Total	21	25	

- ✓ **Incremento**

En esta iteración se obtuvo como resultado la funcionalidad de consulta de cuentas por usuario, desplegados en localhost.

- **Consultar cuentas por usuario**

Se realiza la prueba para consultar todas las cuentas que le pertenecen a un usuario. Ver Figura 49.

Figura 49. Consulta las cuentas de un usuario

2.3.4 Sprint 4

En el Sprint 4 se agrega la funcionalidad de consultar los movimientos que se han registrado a una cuenta específica que pertenece a un usuario, mostrando la información de los movimientos. Presenta un alto nivel de complejidad de consulta debido a que se emplea cinco tablas relacionadas. La Tabla 16 define los objetivos a cumplir en esta iteración y la Tabla 17 desglosa las las tareas del Sprint.

- **Reunión de planificación – Sprint 4**

Tabla 16. Reunión de planificación - Sprint 4

Sprint 4	
Fecha de la reunión:	24/01/2023
Asistentes:	Scrum Master, Product Owner y Team
Resultado	Sprint Backlog – Sprint 4
Objetivos del sprint: • Implementar la funcionalidad para consultar los movimientos realizados por una cuenta que pertenece a un usuario • Validar los parámetros de búsqueda del identificador de la cuenta y usuario • Agregar una capa de validación y control de mensajes de error • Ejecutar los servicios de consulta de usuarios por cuenta en el entorno Localhost desarrollado en REST	

- ✓ **Sprint Backlog – Sprint 4**

Tabla 17. Sprint backlog - Sprint 4

Nro. HU	Nombre	Tarea	Horas
HU-05	Consultar Movimientos por Cuenta de Usuario	Crear las Entidades necesarias para mapear a la base de datos	4
		Crear los DTO para manipular la información que reciben las tablas	4
		Construir un servicio que permita la consulta de movimientos de una cuenta que pertenecen a un usuario	4
		Añadir una capa de validación a los parámetros de búsqueda de cuenta y de usuario	3
		Agregar el componente controlador y añadir el método para consumir el servicio	4
		Ejecución del servicio consultar movimientos por cuenta de un usuario en REST	4

- **Reunión de revisión – Sprint 4**

Finalizado el Sprint 4 se realiza la reunión de revisión para presentar y evaluar el cumplimiento de las actividades establecidas anteriormente, con la finalidad de mostrar la funcionalidad, la duración y el estado de las tareas, como se observa en la Tabla 18.

Tabla 18. Reunión de revisión - Sprint 4

Responsable	Tarea	Tiempo estimado (horas)	Tiempo real (horas)	Estado
Desarrollador	Crear las Entidades necesarias para mapear a la base de datos	3	4	Realizado
	Crear los DTO para manipular la información que reciben las tablas	3	4	Realizado
	Construir un servicio que permita la consulta de movimientos de una cuenta que pertenecen a un usuario	4	4	Realizado
	Añadir una capa de validación a los parámetros de búsqueda de cuenta y de usuario	3	3	Realizado
	Agregar el componente controlador y añadir el método para consumir el servicio	2	4	Realizado
	Ejecución del servicio consultar movimientos por cuenta de un usuario en REST	2	4	Realizado
Reuniones Scrum	Planificación	1	1	Realizado
	Revisión	1	1	Realizado
	Retrospectiva	1	1	Realizado
	Total	18	26	

✓ **Incremento**

En esta iteración se obtuvo como resultado la implementación del servicio de consulta de movimientos registrados a una cuenta perteneciente a un usuario en específico.

- **Consultar movimientos por cuenta de un usuario**

En este Sprint se comprueba la funcionalidad de búsqueda de movimientos que ha realizado un usuario con una cuenta específica, como se puede observar en la Figura 50.

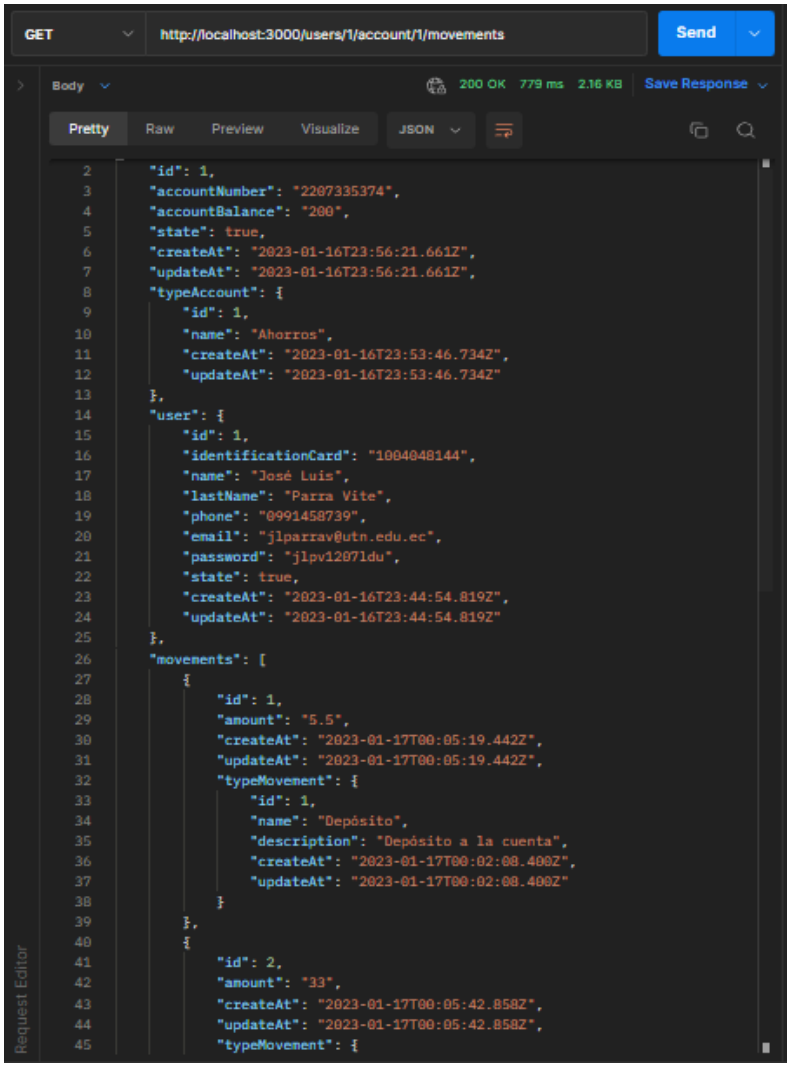

Figura 50. Consulta de movimientos por cuenta de usuario

- **Documentación de la API REST con Swagger**

Adicionalmente como requisito para cada iteración se debe presentar una documentación de la API REST utilizando la librería Swagger, donde a continuación se muestran las capturas de la documentación obtenida. Ver Figuras 51, 52, 53 y 54**.**

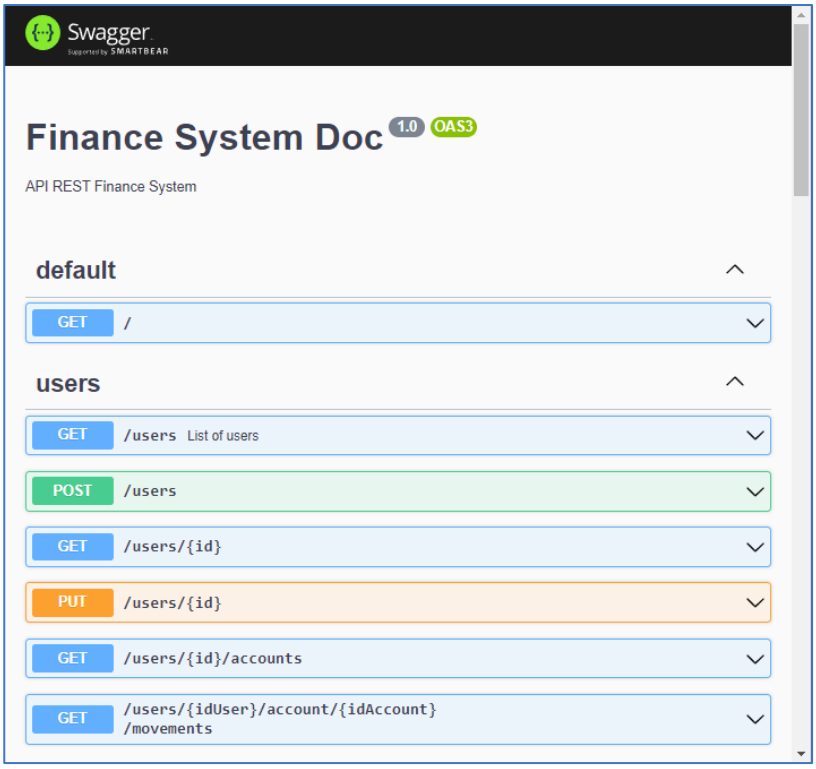

Figura 51. Documentación de la API con Swagger (parte 1)

Figura 52. Documentación de la API con Swagger (parte 2)

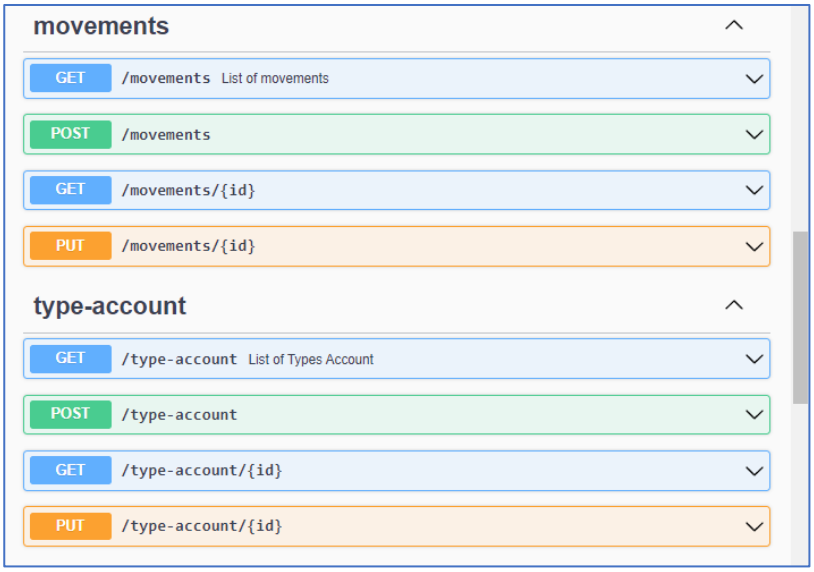

Figura 53. Documentación de la API con Swagger (parte 3)

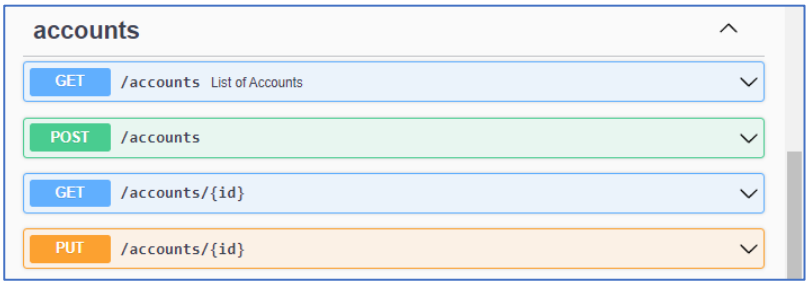

Figura 54. Documentación de la API con Swagger (parte 4)

2.3.5 Repositorio del Proyecto

Una vez concluido con el desarrollo y pruebas de los servicios realizados en los Sprint se obtiene una aplicación funcional y escalable, que puede implementarse con otras herramientas para mejorar aún más su portabilidad y automatización al generar builds entregables. El código de la aplicación se encuentra alojado en el repositorio de GitHub como se observa en la Figura 55.

https://github.com/jose0112luis/finance-system

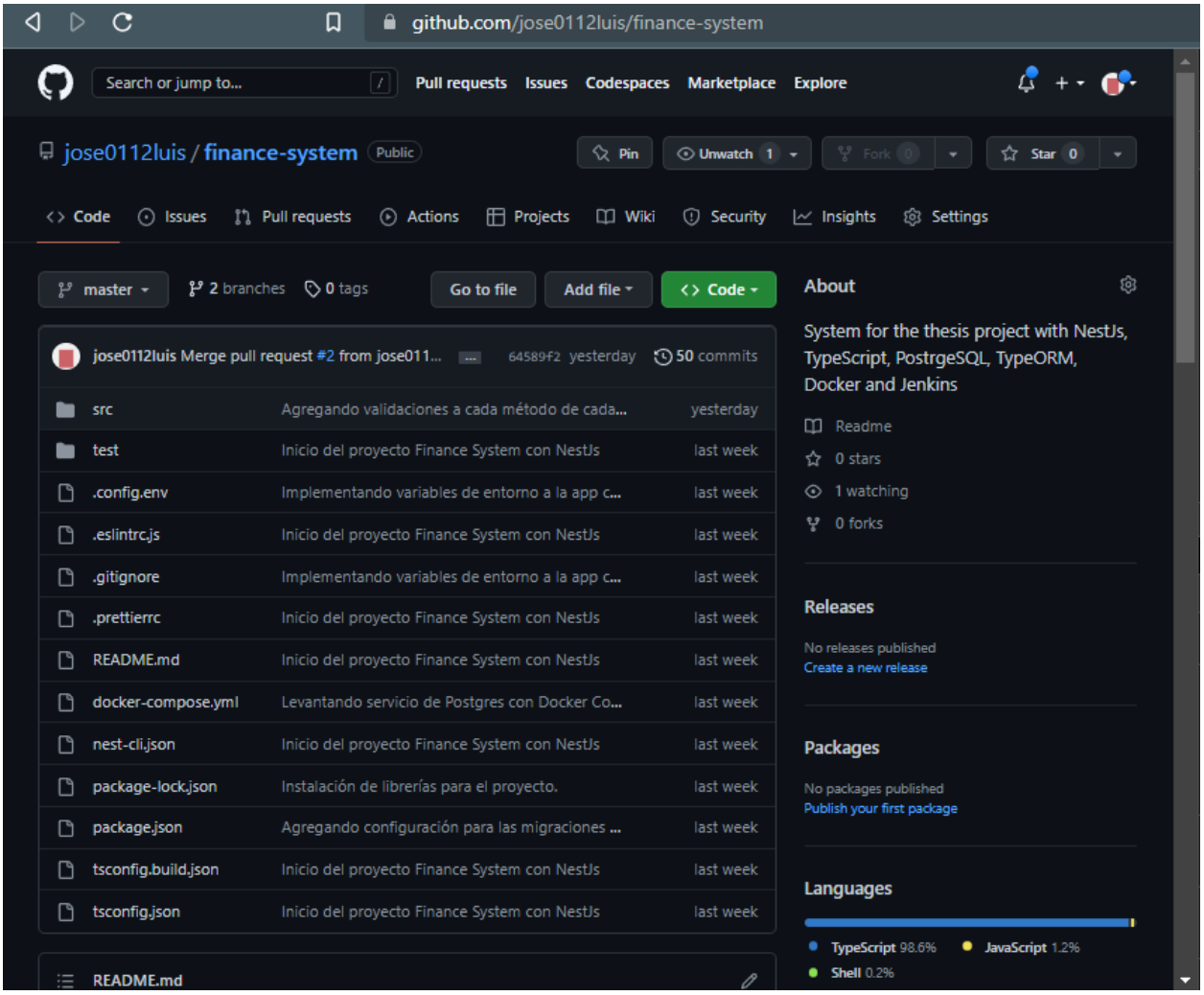

Figura 55. Repositorio del proyecto en GitHub

2.3.6 Acta de Entrega del Desarrollo

Luego de haber desarrollado todas las historias de usuario del Product Backlog en la sección 2.3 Desarrollo de módulos (Cuentas y Transferencias), se obtuvo como resultado un producto software terminado y revisado, el cual será entregado al Product Owner del proyecto mediante un acta entrega recepción del desarrollo.

CAPÍTULO III

3 Validación de Resultados

En este capítulo, se realiza la validación de la arquitectura de software que ha sido diseñada utilizando Archimate Tool como lenguaje de modelado o diseño de arquitecturas. Se utiliza dos estrategias para la evaluación donde se verifica los cumplimientos de la arquitectura y del rendimiento de despliegue en un entorno Localhost. En la Figura 56 se muestra la estructura del capítulo.

Figura 56. Estructura Captulo 3

3.1 Evaluación de la Arquitectura

Como método de evaluación se emplea un checklist tomando las principales cláusulas del estándar internacional ISO/IEC/IEEE 42010 como se presenta en la Tabla 19, de esta forma, se obtiene una guía para evaluar si la arquitectura cumple con las especificaciones relevantes de la norma. Este enfoque de validación garantiza que el diseño de la arquitectura cumpla con las pautas y practicas recomendadas por el estándar internacional.

Tabla 19. Checklist para evaluar el cumplimiento de principios de la norma ISO/IEC/IEEE 42010

Cláusula	Principio	Especificación	Cumple	
			SI	NO
Cláusula Nº 5	Descripción de la arquitectura	**5.2** Identificación y descripción general de la arquitectura	✓	
		5.3 Identificación de partes interesadas y preocupaciones	✓	

		5.4 Puntos de vista de la arquitectura		✓
		5.5 Vistas de la arquitectura	✓	
		5.6 Modelos de arquitectura	✓	
		5.7.1 Consistencia dentro de una descripción de arquitectura		✓
		5.7.2 Correspondencias	✓	
		5.7.3 Reglas de correspondencia	✓	
		5.8.1 Registro de razones	✓	
		5.8.2 Registro de decisiones	✓	
Cláusula Nº 6	Marcos de arquitectura y lenguajes de descripción de arquitecturas	**6.1** Marcos de arquitectura	✓	
		6.2 Adherencia de una descripción de arquitectura a un marco de arquitectura	✓	
		6.3 Lenguajes de descripción de arquitectura	✓	
Cláusula Nº 7	Puntos de vista de la arquitectura	**a)** Una o más preocupaciones enmarcadas por este punto de vista (según 5.3);	✓	
		b) Partes interesadas típicas para las preocupaciones enmarcadas por este punto de vista (por 5.3);	✓	
		c) Uno o más tipos de modelos utilizados en este punto de vista;		✓
		d) Para cada tipo de modelo identificado en c), los lenguajes, notaciones, convenciones, técnicas de modelización, métodos analíticos y/u otras operaciones		✓

		que se utilizarán en los modelos de este tipo;		
		e) Referencias a sus fuentes.	✓	

Después de validar la arquitectura mediante el checklist con las características destacadas de la norma ISO/IEC/IEEE 42010 (Tabla 19), se puede afirmar que el cumplimiento de la aplicación del estándar internacional es de 77,8 %, con un total de 16 aspectos cumplidos y 4 no cumplidos, como indica la Figura 57.

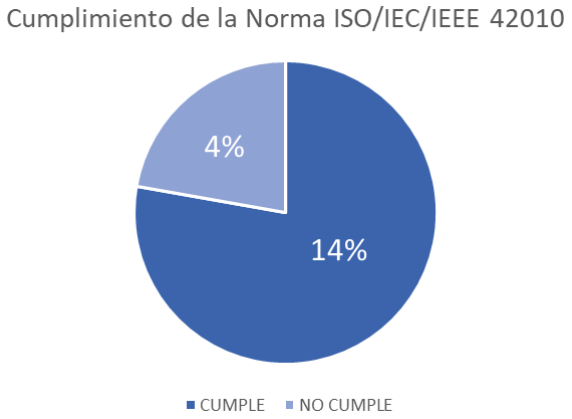

Figura 57. Gráfico del Cumplimiento de la Norma ISO/IEC/IEEE 42010

3.2 Pruebas de Rendimiento de la Arquitectura

Del mismo modo, como método de validación de la arquitectura, se desarrolló una prueba de concepto del módulo de consulta de cuentas y transferencias para dar cumplimiento a los objetivos del presente trabajo. Es así como se llevará a cabo pruebas de rendimiento o de estrés para medir el comportamiento del sistema.

Las pruebas de estrés tienen el objetivo de identificar la capacidad de respuesta y estabilidad del sistema bajo la arquitectura diseñada, simulando situaciones por las cuales el sistema se opera en su máxima capacidad, ya sea por usuarios concurrentes que realizan peticiones al servicio, volúmenes de datos, transacciones o carga de trabajo.

3.2.1 Instrumentación

Para ejecutar el experimento, se debe tomar como punto base a la infraestructura obtenida, puesto que dependiendo de los recursos computacionales pueden variar los resultados y calificaciones. En este caso, la infraestructura cuenta con los siguientes componentes:

Descripción de PC personal:

- Sistema Operativo: Linux Ubuntu 22.04 64-bit
- Procesador: Intel(R) Core (TM) i7-7500U CPU @ 2.70GHz 2.90 GHz
- Memoria: RAM de 8 GB

Herramienta para pruebas de rendimiento

La herramienta seleccionada para realizar las pruebas de estrés al sistema es JMeter. La aplicación Apache JMeter es un software de código abierto, que permite cargar, probar el comportamiento funcional y medir el rendimiento. Originalmente fue diseñado para probar aplicaciones web, pero desde entonces se ha expandido a otras funciones de prueba (Apache Software Foundation, 2023).

En el trabajo de investigación de (Padilla Jaramillo, 2019), menciona que en un estudio anterior, se realizó una comparativa entre varias herramientas para pruebas de rendimiento, teniendo a; ApacheBench, Httperf, Siege y Apache JMeter como las herramientas dominantes, y concluye que JMeter es quien pudo generar mayor cantidad de transacciones por segundo, teniendo un cálculo aproximado de 34.000.

3.2.2 Ejecución de Pruebas

Primero se obtuvieron 5 conjuntos de pruebas, donde cada conjunto ejecuta las 4 peticiones GET HTTP al servicio, tal y como se definieron en las HU (3, 4 y 5):

- 1era petición: Consulta de todos los usuarios registrados
- 2da petición: Consulta de un usuario por su cédula
- 3era petición: Consulta de las cuentas de un usuario
- 4ta petición: Consulta de los movimientos realizados por una cuenta de un usuario.

Cada conjunto de datos tenía un período de subida de 1 segundo, junto con un contador del bucle establecido en 5, mientras que el número de hilos se definió en; 10, 100, 200, 400 y 800 hilos. En la Figura 58 se observa la configuración de JMeter.

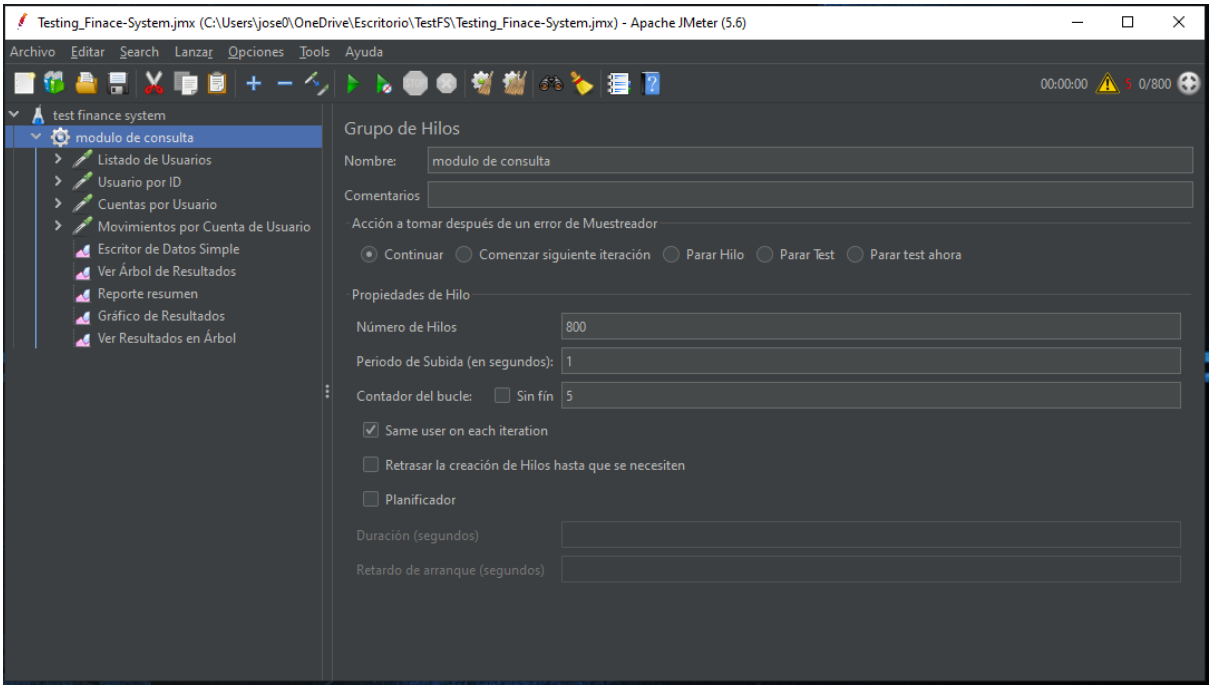

Figura 58. Configuración de JMeter

El número de hilos se refiere a la cantidad de usuarios virtuales que simula la herramienta para realizar peticiones simultáneamente en el entorno de prueba, la cantidad de hilos impacta directamente en la carga generada en la prueba, debido a que mayor número de hilos, mayor es la carga simulada. El período de subida se refiere al intervalo de tiempo que aumenta gradualmente la carga de usuarios virtuales. El contador de bucle es la cantidad de repeticiones que realiza el conjunto de prueba, es decir, el contador de bucle ejecutará el conjunto de acciones "**n**" veces antes de terminar la prueba.

Estas configuraciones básicas de JMeter permiten controlar como se incrementa la carga y como se repiten las acciones de la cantidad de usuarios virtuales que pretende tener el sistema. Estas pruebas ayudan a simular condiciones reales y recopilar datos para mantener un sistema escalable y sostenible.

Detalle de las Pruebas

Los conjuntos de pruebas se componen de la siguiente forma:

 # de Petición = (# de Hilos) * (Segundos) * (Bucle)

Tabla 20. Cojunto de datos

Primer conjunto de datos

	# de Hilos	Segundos	Bucle	Peticiones
1era petición	10	1	5	50
2da petición	10	1	5	50
3era petición	10	1	5	50
4ta petición	10	1	5	50
Total de ejecuciones				200

El proceso aplicado en la Tabla 20, se lo realiza para cada conjunto de datos, en los que se obtuvieron los siguientes resultados de ejecuciones:

- C1 con 10 hilos = 200 ejecuciones
- C2 con 100 hilos = 2000 ejecuciones
- C3 con 200 hilos = 4000 ejecuciones
- C4 con 400 hilos = 8000 ejecuciones
- C5 con 800 hilos = 16000 ejecuciones

3.2.3 Análisis y Resultado de las Pruebas

En esta sección, se presentan y describen los resultados conseguidos a partir de las pruebas de rendimiento realizadas con la herramienta JMeter con diferentes números de hilos. Esta herramienta proporciona la opción para generar un reporte en HTML con los resultados conseguidos, presentando figuras y tablas estadísticas de las pruebas realizadas que se presentan a continuación.

Conjunto de Prueba 1

. En las Figura 59 se presenta el rendimiento del sistema con 10 hilos por cada segundo en un bucle de 5 para cada petición, teniendo un total de 200 ejecuciones. Como métricas clave se consiguió; el tiempo promedio de respuesta de 8,18 ms, un porcentaje de error del 0%, 0 ejecuciones fallidas, es decir, todas las ejecuciones se completaron exitosamente, una mediana de 6.00 ms y un rendimiento de 189,04 transacciones/segundo. Estos resultados indican un alto rendimiento con el nivel de carga bajo.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**200**	**0**	**0.00%**	**8.18**	**2**	**22**	**6.00**	**16.00**	**17.00**	**21.99**	**189.04**	**213.15**	**23.95**
Cuentas por Usuario	50	0	0.00%	6.64	4	10	6.50	8.00	9.00	10.00	49.41	49.55	6.37
Listado de Usuarios	50	0	0.00%	4.40	2	21	4.00	6.00	10.70	21.00	48.54	80.31	5.74
Movimientos por Cuenta de Usuario	50	0	0.00%	15.18	11	22	14.50	19.00	21.45	22.00	49.26	62.15	6.88
Usuario por ID	50	0	0.00%	6.50	4	13	6.00	9.00	9.45	13.00	49.31	29.16	5.92

Figura 59. Estadísticas del conjunto 1 de prueba

Conjunto de Prueba 2

En las Figura 60 se presenta el rendimiento del sistema con 100 hilos por cada segundo en un bucle de 5 para cada petición, teniendo un total de 2000 ejecuciones. Se puede observar que existe un incremento en la simulación de la carga. Sin embargo, la tasa de error sigue manteniéndose en 0% al igual que las ejecuciones fallidas, el tiempo de respuesta aumentó a 291,19 ms, la mediana se obtuvo de 239,00 ms y el rendimiento fue de 299,76 transacciones/segundo. Estos resultados indican que el sistema puede funcionar perfectamente con un ligero aumento de carga.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**2000**	**0**	**0.00%**	**291.19**	**8**	**903**	**239.00**	**585.80**	**688.95**	**799.79**	**299.76**	**338.07**	**37.98**
Cuentas por Usuario	500	0	0.00%	212.71	8	367	210.50	292.90	304.90	356.97	78.15	78.38	10.07
Listado de Usuarios	500	0	0.00%	193.79	8	360	180.00	265.80	328.90	352.00	81.46	134.76	9.63
Movimientos por Cuenta de Usuario	500	0	0.00%	546.71	18	903	517.50	742.70	766.75	879.00	75.32	95.04	10.52
Usuario por ID	500	0	0.00%	211.54	8	363	212.50	293.90	303.85	354.99	79.59	47.14	9.56

Figura 60. Estadística del conjunto 2 de prueba

Conjunto de Prueba 3

En la Figura 61 refleja los resultados obtenidos al aumentar el número de hilos a 200 con un total de 4000 ejecuciones. Del mismo modo, se mantiene en 0 el porcentaje de error y de

ejecuciones fallidas, mientras que el tiempo promedio de respuesta es de 511,51 ms, la mediana es 374,00 ms y el rendimiento es de 355,46 transacciones/segundo. Estos resultados siguen siendo positivos al tener un alto nivel de rendimiento y un 0% de error.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**4000**	**0**	**0.00%**	**511.51**	**24**	**1306**	**374.00**	**995.90**	**1053.00**	**1140.97**	**355.46**	**400.85**	**45.04**
Cuentas por Usuario	1000	0	0.00%	357.29	24	567	357.50	403.00	419.95	548.99	95.91	96.20	12.36
Listado de Usuarios	1000	0	0.00%	348.63	24	566	343.00	428.00	534.95	555.00	101.66	168.17	12.01
Movimientos por Cuenta de Usuario	1000	0	0.00%	977.00	272	1306	960.50	1085.00	1123.00	1289.98	89.77	113.26	12.54
Usuario por ID	1000	0	0.00%	363.13	46	559	363.00	417.00	505.90	550.99	98.43	58.26	11.82

Figura 61. Estadística del conjunto 3 de prueba

Conjunto de Prueba 4

La Figura 62 muestra los resultados al ejecutar las pruebas con 400 hilos, dando un total de 8000 ejecuciones. Del mismo modo, la tasa de error es de 0% al igual que las ejecuciones fallidas, el tiempo promedio de respuesta aumenta a 923,85 ms, la mediana es 715,00 ms y el rendimiento es de 401,95 transacciones/segundo. Los resultados indican que hasta este nivel de carga se sigue manteniendo un buen rendimiento a pesar del aumento de carga.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**8000**	**0**	**0.00%**	**923.85**	**25**	**2405**	**715.00**	**1686.00**	**1824.00**	**1938.00**	**401.95**	**453.38**	**50.93**
Cuentas por Usuario	2000	0	0.00%	688.50	64	861	693.00	785.90	815.00	845.99	110.36	110.68	14.23
Listado de Usuarios	2000	0	0.00%	644.81	25	869	658.00	827.00	847.00	857.99	118.88	196.67	14.05
Movimientos por Cuenta de Usuario	2000	0	0.00%	1700.91	389	2405	1650.00	1876.00	1919.75	2309.48	101.14	127.61	14.12
Usuario por ID	2000	0	0.00%	661.17	32	869	661.00	809.00	823.00	848.99	114.08	67.64	13.70

Figura 62. Estadística del conjunto 4 de prueba

Conjunto de Prueba 5

La figura 63 refleja los resultados obtenidos de la prueba al aumentar la carga a 800 hilos, teniendo 16000 ejecuciones en total. En esta configuración, el tiempo promedio de respuesta disminuyo a 507,34 ms en comparación con el conjunto de pruebas anterior, esto se debe a que en este punto se obtuvo 8221 ejecuciones fallidas, es decir, que no se completaron, por lo que el tiempo que debió tomar esas ejecuciones fue mínimo. En consecuencia, la tasa de error incrementó a 51,38%. Estos hallazgos indican que el sistema ha alcanzado su límite, teniendo un nivel de carga extremo y una mayor proporción de errores bajo cargas más elevadas.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**16000**	**8221**	**51.38%**	**507.34**	**0**	**4356**	**134.00**	**1566.00**	**1717.95**	**3610.00**	**775.72**	**1449.53**	**47.91**
Cuentas por Usuario	4000	2051	51.27%	329.18	0	2651	103.50	720.90	776.00	1059.99	209.92	379.03	13.18
Listado de Usuarios	4000	2120	53.00%	474.05	0	3647	132.00	689.00	3471.85	3612.00	221.55	473.83	12.30
Movimientos por Cuenta de Usuario	4000	1938	48.45%	936.02	0	4356	1439.00	1873.00	1971.00	4268.99	194.92	369.34	14.03
Usuario por ID	4000	2112	52.80%	290.10	0	2887	66.00	648.00	727.00	798.94	216.15	353.47	12.25

Figura 63. Estadística del conjunto 5 de prueba

En resumen, los resultados de las pruebas de rendimiento indican que el sistema está apto para sobrellevar cargas moderadas a altas con un buen rendimiento, teniendo una carga de 400 usuarios que realizan 8000 peticiones por segundo. Sin embargo, a medida que la carga aumenta a 800 usuarios que realizan 16000 peticiones por segundo, se presenta una disminución significativa de más del 50% de rendimiento, ocasionado un bajo rendimiento y mayor cantidad de fallos con una tasa alta de errores. Hay que tomar en cuenta que se estableció un periodo de subida de 1 segundo para todas las pruebas, de modo que, si se aumenta los segundos se obtiene un rango de mayor cantidad de usuarios que puede soportar el sistema.

Conclusiones

- Se estableció un marco teórico conceptual de la literatura que consiste en realizar un proceso de planificación de la revisión bibliográfica, búsqueda y extracción de información sobre arquitecturas de software y el ciclo de vida de DevOps, de igual manera, se define una conceptualización de herramientas y técnicas que se pueden integrar en estos procesos para aumentar su rendimiento.

- En base a la conceptualización establecida sobre el ciclo de vida de DevOps, herramientas tecnológicas para el desarrollo, integración y despliegue de aplicaciones y sobre diseño de arquitecturas con TOGAF y la norma ISO/IEC/IEEE 42010, se diseñó una arquitectura de software utilizando la fase 3 y 4 de TOGAF que soporta la implementación de aplicaciones orientadas a microservicios y que cumple con el ciclo de vida de DevOps.

- Una vez diseñada la arquitectura, se realizó la validación mediante 2 técnicas. Se utilizó un checklist proporcionado por la norma ISO/IEC/IEEE 42010 que utiliza una serie de características que debe tener una arquitectura de software. Y se realizó un laboratorio de pruebas de rendimiento al sistema dividida en 5 grupos de ejecuciones, en donde los resultados muestran que la arquitectura puede funcionar adecuadamente sin presentar errores en las peticiones al servicio hasta con 400 usuarios simulados que realizan 8000 peticiones por segundo.

- Implementar sistemas de contenedores en una arquitectura de software es indispensable actualmente, debido a que ofrece portabilidad a las aplicaciones, reducción de costos en infraestructura y mejoras en tiempos de despliegue, ya que pueden ejecutarse de manera exitosa en diferentes entornos, sin tener que hacer cambios al código fuente o a la infraestructura. Es así como se establece una arquitectura escalable y portable.

Recomendaciones

- Para el diseño de una arquitectura, se recomienda conocer bien cuales son los requisitos base que se tiene y sobre qué tipo de infraestructura se pretende levantar la arquitectura, también es importante mantener una buena comunicación con las partes interesadas para satisfacer sus necesidades.
- Se recomienda impartir conocimientos a los estudiantes de la carrera de CSOFT sobre temas de DevOps y virtualización con contenedores debido a que son herramientas que permiten automatizar procesos del ciclo de vida del software, para disminuir tiempos y aumentar calidad. De la misma forma se debe manejar una buena planificación y un control de versionamiento de código.
- Existen varias técnicas y herramientas que permiten agregar calidad y confianza a un sistema, se recomienda utilizar estándares internacionales como la ISO/IEC/IEEE 42010 o el framework TOGAF para validar los diseños y emplear herramientas que faciliten el desarrollo y despliegue de aplicaciones.

Bibliografía

Abbass, M. K. A., Osman, R. I. E., Mohammed, A. M. H., & Alshaikh, M. W. A. (2019). Adopting continuous integeration and continuous delivery for small teams. *Proceedings of the International Conference on Computer, Control, Electrical, and Electronics Engineering 2019, ICCCEEE 2019*. https://doi.org/10.1109/ICCCEEE46830.2019.9070849

Alnafessah, A., Gias, A. U., Wang, R., Zhu, L., Casale, G., & Filieri, A. (2021). Quality-Aware DevOps Research: Where Do We Stand? *IEEE Access*, *9*, 44476–44489. https://doi.org/10.1109/ACCESS.2021.3064867

Apache Software Foundation. (2023). *Apache JMeter™*. https://jmeter.apache.org/

Atlassian. (2022). *Comparación de contenedores y máquinas virtuales*. Atlassian - Desarrollo de Software. https://www.atlassian.com/es/microservices/cloud-computing/containers-vs-vms

Bastidas Bastidas, E. A. (2020). *Desarrollo de un sistema web para la automatización del proceso de mapeo sistemático de la literatura, y validado mediante un marco de trabajo de calidad de uso basado en las normas ISO/IEC 25000 para mejorar el proceso de investigación en los docentes de* [Universidad Técnica del Norte]. http://repositorio.utn.edu.ec/handle/123456789/10254

Canabal, R., Cabarcas, A., & Martelo, R. J. (2017). Application of an Open Group Architecture Framework (TOGAF) to a Small Enterprise (SME) using Google Collaborative Applications. *Información Tecnológica*, *28*(4), 85–92. https://doi.org/10.4067/S0718-07642017000400011

Casalicchio, E., & Iannucci, S. (2020). The state-of-the-art in container technologies: Application, orchestration and security. *Concurrency and Computation: Practice and Experience*, *32*(17), e5668. https://doi.org/10.1002/CPE.5668

Casasola Girón, N. E. (2019). *Diseño de un modelo de integración continua utilizando Jenkins para proyectos de desarrollo en una empresa de telecomunicaciones para el módulo de contabilidad* [Universidad de San Carlos de Guatemala]. http://biblioteca.ingenieria.usac.edu.gt/

Click-IT. (2022, April 7). *Arquitectura monolítica vs arquitectura de microservicios: ¿cuál debo elegir? | Click-IT | Servicios tecnológicos y de consultoría*. Click-IT. https://click-it.es/arquitectura-monolitica-vs-arquitectura-de-microservicios-cual-debo-elegir/

Cuervo, V. (2019, March 15). *¿Qué es DevOps? – Arquitecto IT*. https://www.arquitectoit.com/devops/que-es-devops/

Dhir, S., Kumar, D., & Singh, V. B. (2019). Success and failure factors that impact on project implementation using agile software development methodology. *Advances in Intelligent Systems and Computing*, *731*, 647–654. https://doi.org/10.1007/978-981-10-8848-3_62/COVER

Díaz Cortés, E. A. (2019). *Mejora del proceso de desarrollo de una empresa de servicios de información mediante la incorporación DevOps* [Universidad de Chile]. https://repositorio.uchile.cl/handle/2250/170924

Eddy, B. P., Wilde, N., Cooper, N. A., Mishra, B., Gamboa, V. S., Shah, K. M., Deleon, A. M., & Shields, N. A. (2017a). A Pilot Study on Introducing Continuous Integration and Delivery into Undergraduate Software Engineering Courses. *Proceedings - 30th IEEE Conference on Software Engineering Education and Training, CSEE and T 2017*, *2017-January*, 47–56. https://doi.org/10.1109/CSEET.2017.18

Eddy, B. P., Wilde, N., Cooper, N. A., Mishra, B., Gamboa, V. S., Shah, K. M., Deleon, A. M., & Shields, N. A. (2017b). A Pilot Study on Introducing Continuous Integration and Delivery into Undergraduate Software Engineering Courses. *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE and T)*, *2017-Janua*, 47–56. https://doi.org/10.1109/CSEET.2017.18

edureka. (2022, November 22). *Docker Networking – Explore How Containers Communicate With Each Other*. Edureka Blog. https://www.edureka.co/blog/docker-networking/

Edureka. (2022, September 28). *DevOps Tutorial: Introduction to DevOps* . Edureka Blog. https://www.edureka.co/blog/devops-tutorial#WhatisDevOps?

El Khalyly, B., Belangour, A., Banane, M., & Erraissi, A. (2020). A new metamodel approach of CI/CD applied to Internet of Things Ecosystem. *2020 IEEE 2nd International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS)*. https://doi.org/10.1109/ICECOCS50124.2020.9314485

Galindo, J. A., Benavides, D., Trinidad, P., Gutiérrez-Fernández, A. M., & Ruiz-Cortés, A. (2018). Automated analysis of feature models: Quo vadis? *Computing*, *101*(5), 387–433. https://doi.org/10.1007/S00607-018-0646-1/METRICS

International Organization Of Standardization. (2011). ISO/IEC/IEEE 42010:2011 - Systems and software engineering -- Architecture description. *ISOIECIEEE 420102011E Revision of ISOIEC 420102007 and IEEE Std 14712000, 2011(March)*, 1–46. https://doi.org/10.1109/IEEESTD.2011.6129467

Iñiguez Sánchez, L. A. (2017). *Arquitectura tecnológica para la entrega continua de software con despliegue en contenedores* [Universidad de Cuenca]. http://dspace.ucuenca.edu.ec/handle/123456789/28529

Jamshidi, P., Pahl, C., Mendonca, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software*, *35*(3), 24–35. https://doi.org/10.1109/MS.2018.2141039

Jenkins. (2022). *Jenkins User Documentation*. https://www.jenkins.io/doc/

Kersten, M. (2018). A cambrian explosion of DevOps tools. *IEEE Software*, *35*(2), 14–17. https://doi.org/10.1109/MS.2018.1661330

Khan, M. S., Khan, A. W., Khan, F., Khan, M. A., & Whangbo, T. K. (2022). Critical Challenges to Adopt DevOps Culture in Software Organizations: A Systematic Review. *IEEE Access*, *10*, 14339–14349. https://doi.org/10.1109/ACCESS.2022.3145970

Kwon, S., & Lee, J. H. (2020). DIVDS: Docker Image Vulnerability Diagnostic System. *IEEE Access*, *8*, 42666–42673. https://doi.org/10.1109/ACCESS.2020.2976874

Life Art Tech. (n.d.). *ISO/IEC/IEEE 42010:2011 – Descripción de arquitectura – Parte I: Introducción | LIFE.ART.TECH*. Retrieved January 9, 2022, from https://lifeartech.wordpress.com/2017/08/16/isoiecieee-42010-descripcion-de-arquitectura-intro/

Lingayat, A., Badre, R. R., & Gupta, A. K. (2018). Performance Evaluation for Deploying Docker Containers on Baremetal and Virtual Machine. *Proceedings of the 3rd International Conference on Communication and Electronics Systems, ICCES 2018*, 1019–1023. https://doi.org/10.1109/CESYS.2018.8723998

Lu, Z., Xu, J., Wu, Y., Wang, T., & Huang, T. (2019). An Empirical Case Study on the Temporary File Smell in Dockerfiles. *IEEE Access*, *7*, 63650–63659. https://doi.org/10.1109/ACCESS.2019.2905424

LucusHost. (2022, February 7). *Qué es Node.js y para qué sirve*. Blog de LucusHost.

https://www.lucushost.com/blog/que-es-node-js/#Que_es_Nodejs

Maida, E. G., & Pacienzia, J. (2015). *Metodologías de desarrollo de software* [Pontificia Universidad Católica Argentina]. https://repositorio.uca.edu.ar/handle/123456789/522

Maigualca Toapanta, D. A., & Pillaga Zhagñay, L. A. (2021). *Desarrollo de un sistema web con metodologías DevOps, para optimizar la gestión de historias clínicas en el sistema integrado de salud de la Universidad de las Fuerzas Armadas - ESPE* [Latacunga: Universidad de la Fuerzas Armadas ESPE, 2021]. http://repositorio.espe.edu.ec/jspui/handle/21000/26028

Marijan, D., Liaaen, M., & Sen, S. (2018). DevOps Improvements for Reduced Cycle Times with Integrated Test Optimizations for Continuous Integration. *Proceedings - International Computer Software and Applications Conference*, *1*, 22–27. https://doi.org/10.1109/COMPSAC.2018.00012

Microsoft Learn. (2021, October 28). *Containers vs. virtual machines | Microsoft Learn*. Learn Microsoft. https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm

Microsoft Learn. (2022, April 13). *Docker terminology | Microsoft Learn*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-terminology

Navasa Martínez, A. (2008). *Marco de trabajo para el desarrollo de arquitecturas software orientado a aspectos* [Universidad de Extremadura]. https://dehesa.unex.es:8443/handle/10662/459

Osadhani, Y., Maulana, A., Rizkiputra, D., Kaburuan, E. R., & Sfenrianto. (2019). Enterprise Architectural Design Based on Cloud Computing using TOGAF (Case Study: PT. TELIN). *ICSECC 2019 - International Conference on Sustainable Engineering and Creative Computing: New Idea, New Innovation, Proceedings*, 111–115. https://doi.org/10.1109/ICSECC.2019.8907072

Padilla Jaramillo, J. M. (2019). *Diseño de una arquitectura institucional tecnológica basada en el estándar ISO/IEC/IEEE 42010 que permita diversificar el modelo educativo dentro de la Universidad Técnica del Norte* [Universidad Técnica del Norte]. http://repositorio.utn.edu.ec/handle/123456789/9537

Pahl, C. (2015). Containerization and the PaaS Cloud. *IEEE Cloud Computing*, *2*(3), 24–31.

https://doi.org/10.1109/MCC.2015.51

Palacios Acosta, M. A. (2022). *Diseño de una arquitectura tecnológica que permita fortalecer el proceso de analítica web en el status de portales educativos de alto tráfico.* [Universidad Técnica del Norte]. http://repositorio.utn.edu.ec/handle/123456789/12520

Proenca, D., & Borbinha, J. (2017). Enterprise architecture: A maturity model based on TOGAF ADM. *Proceedings - 2017 IEEE 19th Conference on Business Informatics, CBI 2017*, *1*, 257–266. https://doi.org/10.1109/CBI.2017.38

Rafi, S., Yu, W., Akbar, M. A., Alsanad, A., & Gumaei, A. (2020). Prioritization Based Taxonomy of DevOps Security Challenges Using PROMETHEE. *IEEE Access*, *8*, 105426–105446. https://doi.org/10.1109/ACCESS.2020.2998819

Red Hat. (2020, May 8). *¿Qué es una API de REST?* https://www.redhat.com/es/topics/api/what-is-a-rest-api

Retamal, J. (2019). *Plataforma de Desarrollo de Aplicaciones en el DCC Basada en Técnicas de DevOps* [Universidad de Chile]. https://repositorio.uchile.cl/bitstream/handle/2250/170717/Plataforma-de-desarrollo-de-aplicaciones-en-el-DCC-basada-en-técnicas-de-DevOps.pdf?sequence=1&isAllowed=y

Roldán Martínez, D., Valderas Aranda, P. J., & Torres Bosch, V. (2018). *Microservicios Un enfoque integrado*. RA-MA Editorial. https://books.google.com.ec/books?id=2o6fDwAAQBAJ&printsec=frontcover&hl=es&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game*.

Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, *5*, 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629

Srivastava, A., Bhardwaj, S., & Saraswat, S. (2017). SCRUM model for agile methodology. *Proceeding - IEEE International Conference on Computing, Communication and Automation, ICCCA 2017*, *2017-Janua*, 864–869. https://doi.org/10.1109/CCAA.2017.8229928

Universidad Técnica del Norte. (2022). *Perfil Profesional CSOFT*. Perfil Profesional de La Carrera

de Ingeniería de Software. https://software.utn.edu.ec/gestion-academica/perfil-profesional/

Vásconez Chávez, D. P. (2016). *Análisis de herramientas de desarrollo y su adaptación al modelo de integración continúa* [Universidad de las Fuerzas Armadas ESPE. Carrera de Ingeniería de Sistemas e Informática.]. http://repositorio.espe.edu.ec/jspui/handle/21000/11719

Vinueza Celi, O. S. (2021). *Entorno de Integración, Entrega y Despliegue Contínuo de Software en la Universidad Católica Sede Esmeraldas* [Universidad Católica Sede Esmeraldas]. https://repositorio.pucese.edu.ec/handle/123456789/2418

Visual Paradigm. (2022a). *What is ArchiMate?* https://www.visual-paradigm.com/guide/archimate/what-is-archimate/

Visual Paradigm. (2022b). *What is ArchiMate?* What Is ArchiMate? https://www.visual-paradigm.com/guide/archimate/what-is-archimate/

VMware. (2019, May 31). *Arquitectura de máquina virtual*. VMware Cloud Director. https://docs.vmware.com/es/VMware-Cloud-Director/10.0/com.vmware.vcloud.tenantportal.doc/GUID-8F806B38-2489-4D36-82FF-B23BAFC3B294.html

You, L., & Sun, H. (2022). Research and Design of Docker Technology Based Authority Management System. *Computational Intelligence and Neuroscience*, *2022*. https://doi.org/10.1155/2022/5325694

ANEXOS

Integración del Proyecto con Jenkins

El entorno construido en este trabajo utiliza las siguientes herramientas: Git para el control de versión del código fuente, GitHub para almacenamiento de código en la nube, Docker para la gestión y manejo de contenedores, Jenkins para los procesos de automatización de CI/CD y DockerHub para almacenar las versiones estables o releases del sistema.

El proyecto se lo realiza en el Sistema Operativo Linux Ubuntu 22.04.01, debido a su fácil integración con las herramientas mencionadas anteriormente, siendo Docker la base para levantar los servicios de Base de Datos y Jenkins dentro de contenedores. Cabe mencionar que estos servicios pueden ser instalados directamente en el Sistema Operativo o utilizar un proveedor externo, pero por motivos académicos y para utilizar mínimos recursos se crea este entorno.

Configuración de Jenkins

Jenkins cumple la función de orquestar la integración, entrega y despliegue continuo del sistema, además, se encarga de monitorear el código fuente y vincularse a DockerHub para publicar la última versión estable. A continuación, se muestran todas las configuraciones realizadas en el servidor de Jenkins.

El sistema desarrollado utiliza el lenguaje de programación NodeJs, por lo tanto, como buena práctica es necesario definir que versión de NodeJs se utiliza, esto evita que cuando más programadores trabajen en el sistema utilicen diferentes versiones, presentando errores en la compatibilidad del lenguaje. Los problemas más comunes pueden ser funciones o sintaxis obsoletas que fueron reemplazadas con nuevas características, este tema se lo amplía en la justificación del presente trabajo de titulación. Por ende, en las configuraciones de Jenkins es necesario instalar la versión de NodeJs a utilizar.

Primero nos dirigimos a la Opción de “Gestor de Plugins” e instalamos los Plugins necesarios, como se muestra en la Figura 64.

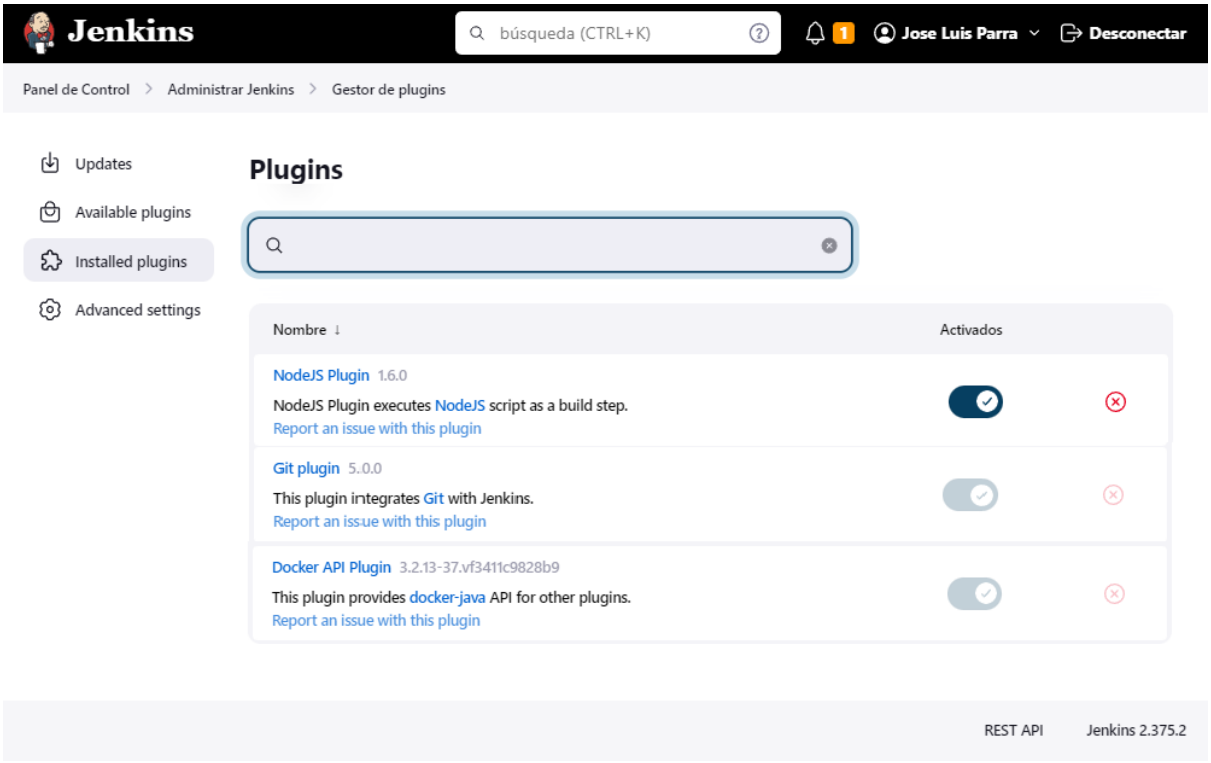

Figura 64. Plugins instalados en Jenkins

A continuación, en el Panel principal vamos a “Administrar Jenkins” y seleccionamos la opción “Global Tool Configuration”, nos muestran un listado de opciones y utilizaremos la opción de NodeJS y seleccionamos la opción “instalaciones de NodeJS” y llenamos los campos requeridos con la versión de NodeJs que necesitemos (podemos instalar más de una versión de NodeJs). La Figura 65 muestra la instalación de NodeJS versión 18.12.1.

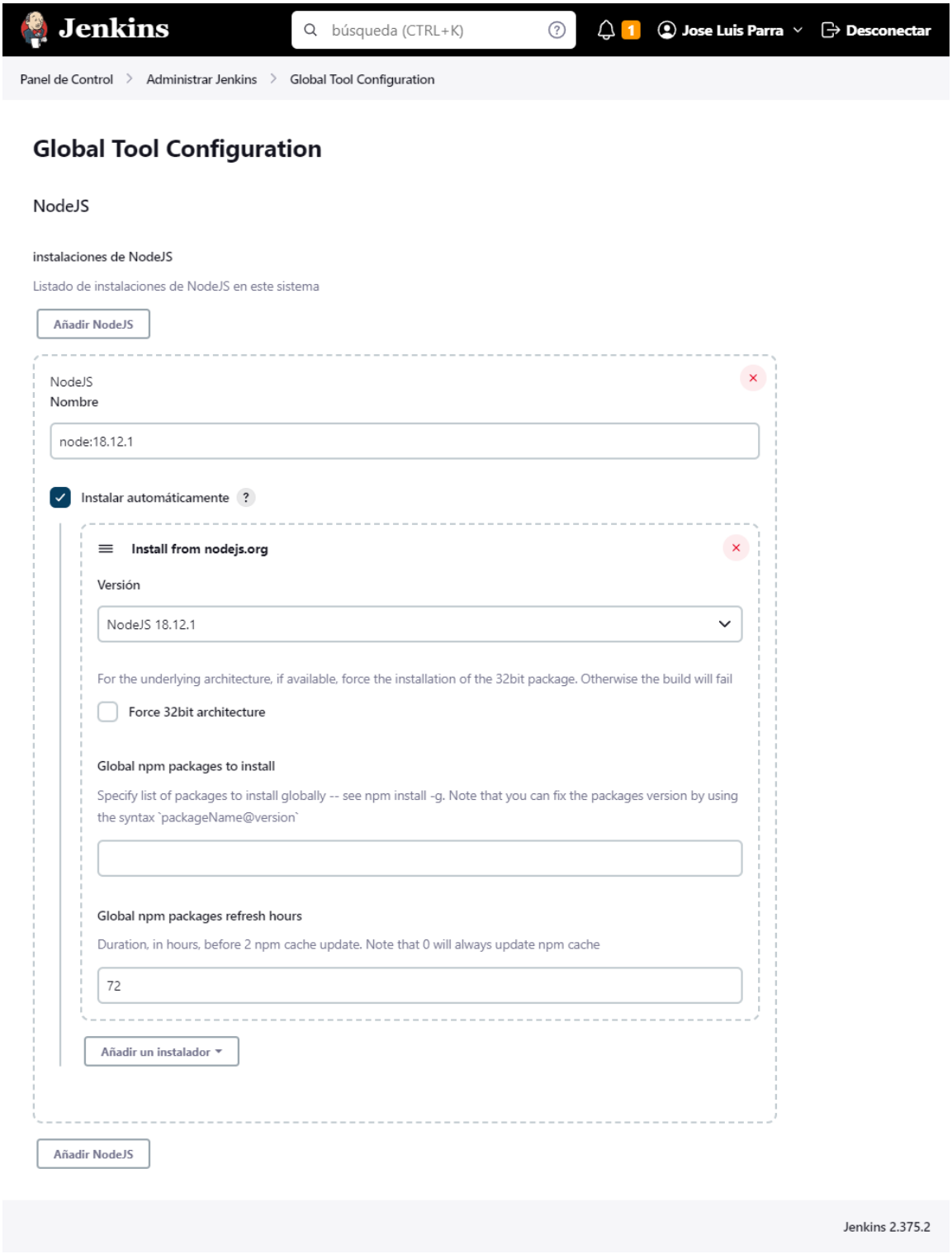

Figura 65. Instalación de NodeJs en Jenkins

Adicionalmente, necesitamos crear y configurar una credencial en Jenkins con el usuario y token provenientes de la cuenta de DockerHub, para ello se necesita crear una cuenta previamente y agregar un nuevo token, como se indica en la Figura 66.

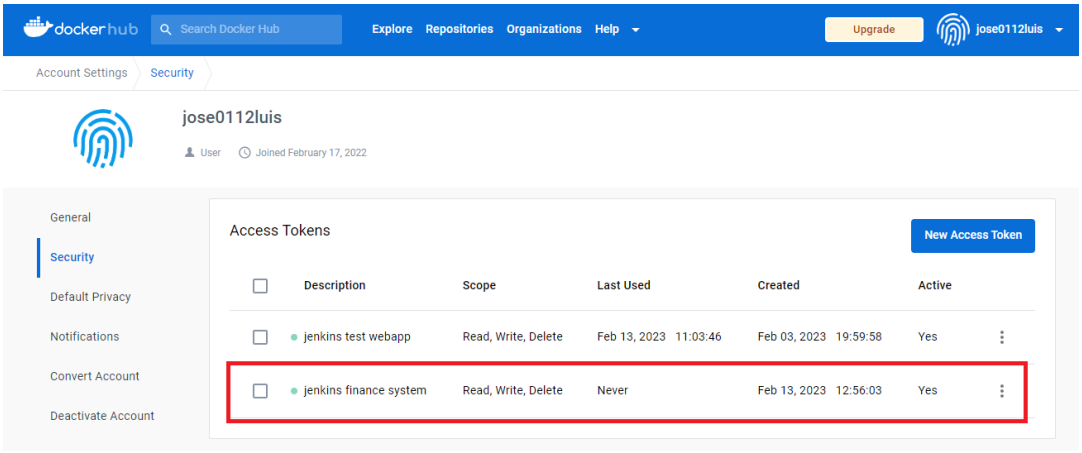

Figura 66. Token de DockerHub

Después de guardar el token de DockerHub, en el Panel de Control de Jenkins seleccionamos “Administrar Jenkins” y en el apartado de Seguridad seleccionamos la opción “Manage Credentials”. Dentro, podemos observar un listado de credenciales, seleccionamos “System” luego “Global crendetials” y agregamos una Nueva Credencial. La Figura 67 muestra las credenciales registradas en Jenkins.

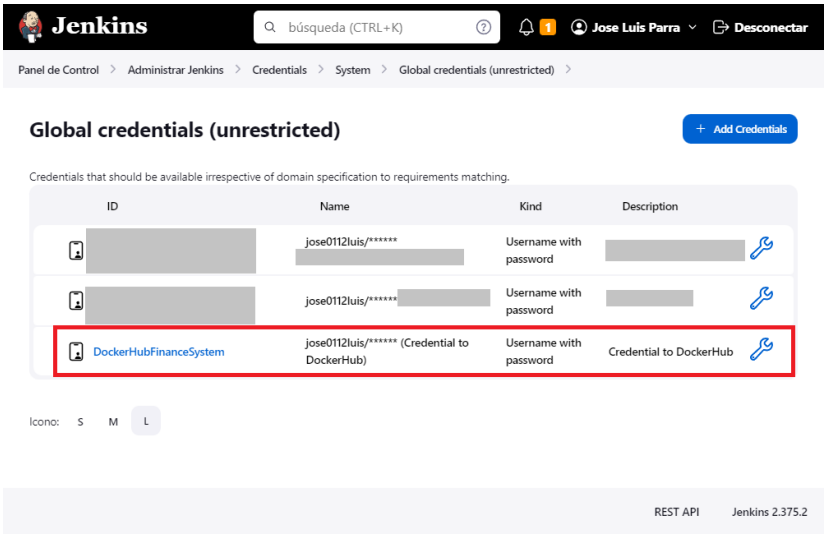

Figura 67. Credencial de DockerHub en Jenkins

Una vez completadas las configuraciones en Jenkins, podemos avanzar con el siguiente paso, el cual corresponde a la creación del Job o tarea, la cual se explica a continuación.

Creación y Configuración del Job

Para crear un nuevo Job o tarea, nos ubicamos en la “Panel de Control” y en las opciones de la parte izquierda seleccionamos “Nueva Tarea”. A continuación, definimos un nombre para la tarea y seleccionamos el tipo, en este caso utilizaremos la opción “Pipeline” como se muestra en la Figura 68.

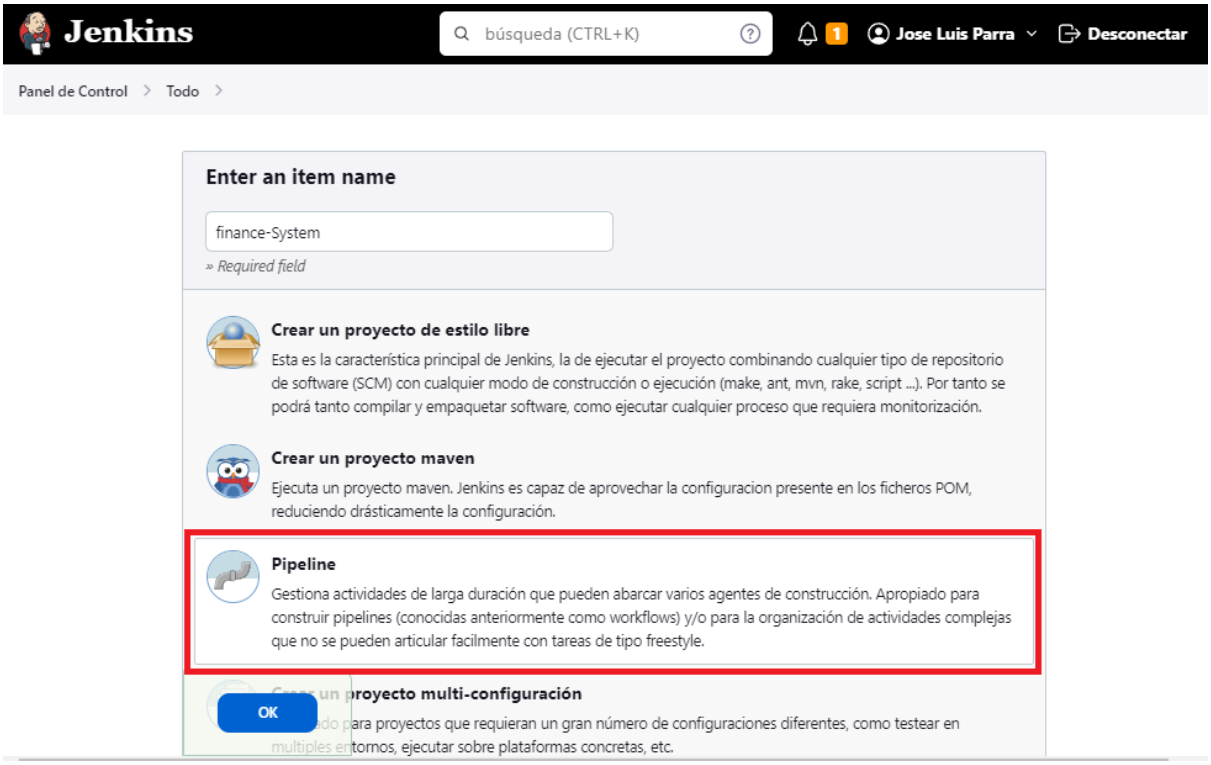

Figura 68. Creación del Job/Tarea

De la misma forma, el Job tiene sus respectivas configuraciones, en este ejemplo utilizaremos las configuraciones básicas. En configuración General únicamente marcamos la casilla que indica el uso de un repositorio de GitHub llamada “GitHub project” (opcional) como se indica en la Figura 69 y también, marcamos la casilla “GitHub hook trigger for GITScm polling” como se muestra en la Figura 70. Esta segunda opción se mantiene monitoreando el repositorio.

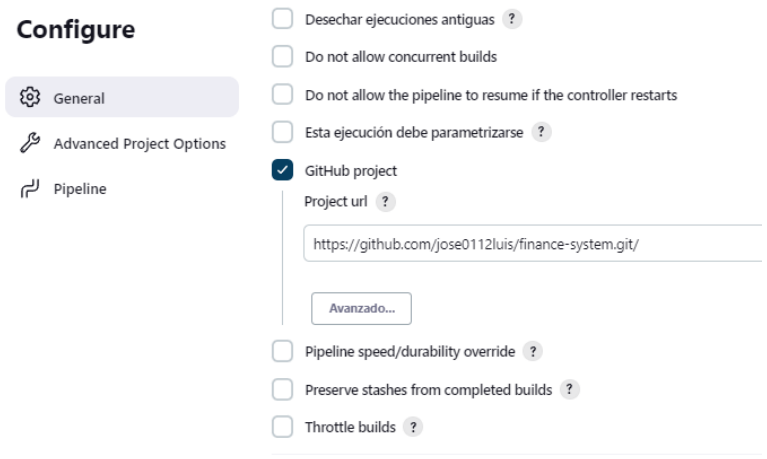

Figura 69. Configuración General del Pipeline parte 1

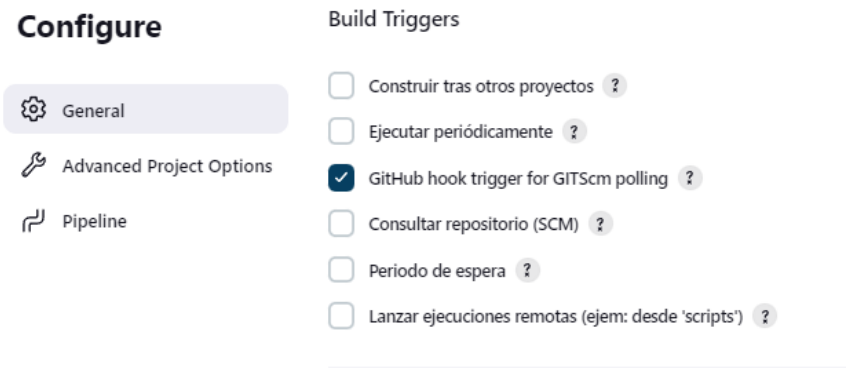

Figura 70. Configuración General del Pipeline parte 2

En la “configuración avanzada del proyecto” no aremos ninguna modificación, pero en la configuración “Pipeline” vamos a definir donde se encuentra el script. Primero especificamos que el Pipeline se encuentra en un repositorio externo y no se definirá un script local, para ello usamos la opción “Pipeline script from SCM” y adjuntamos la URL del repositorio. Al ser un repositorio público no se necesita agregar ninguna credencial y en cuanto a las ramas definiremos que revise todas con “*/*”, y finalmente ubicamos la ruta donde se encuentra el script, es decir, el archivo Jenkinsfile, en este caso se encuentra en la raíz del repositorio, por lo tanto, únicamente escribimos el nombre del archivo Jenkinsfile. La Figura 71 muestra toda la configuración Pipeline.

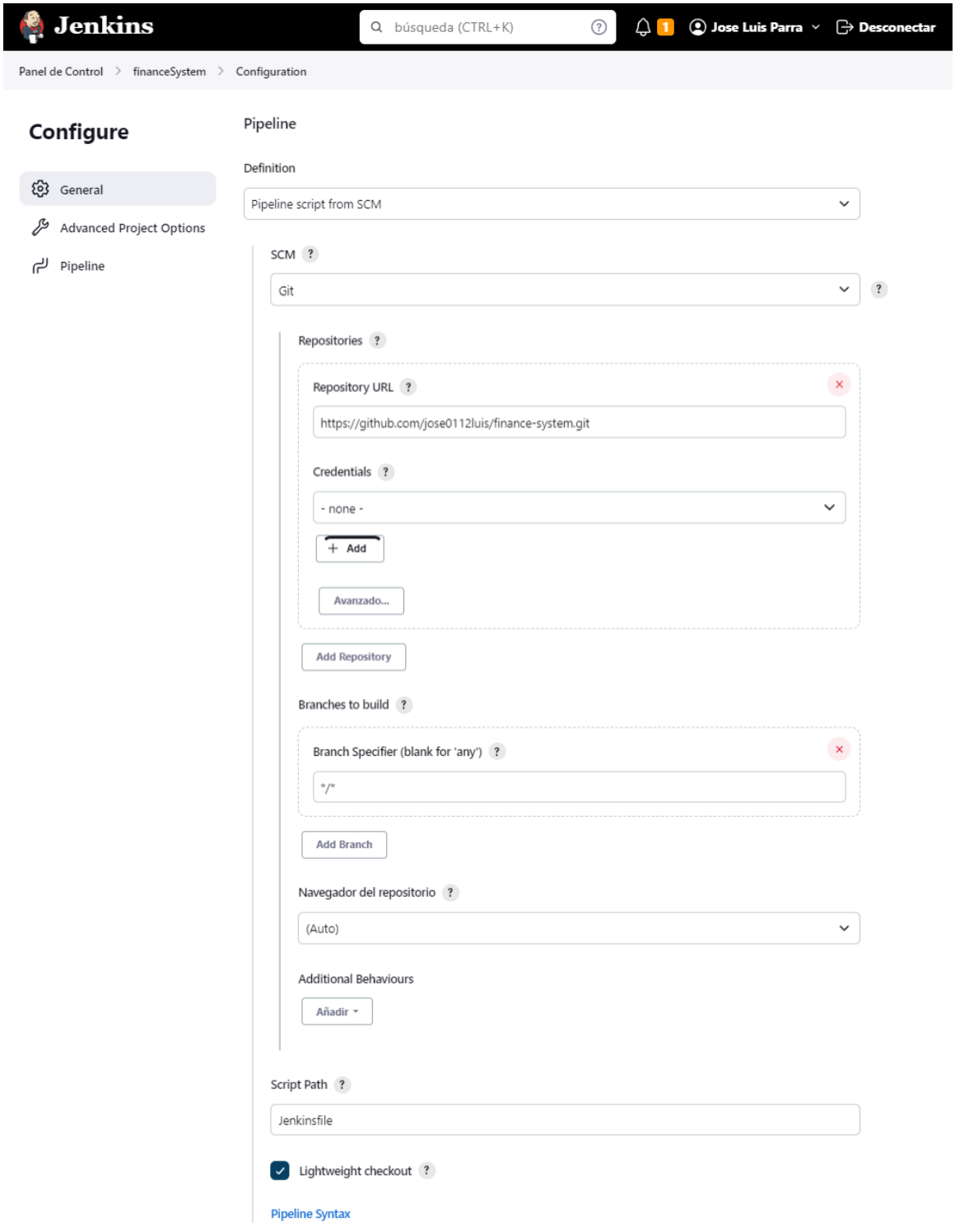

Figura 71. Configuración Pipeline

Después de realizar todos los cambios, guardamos y en el Panel de Control podemos observar todos los Jobs o tareas creadas, en este caso el Job financeSystem ha sido creado, pero no ha tenido ninguna ejecución hasta el momento, como podemos ver en la Figura 72.

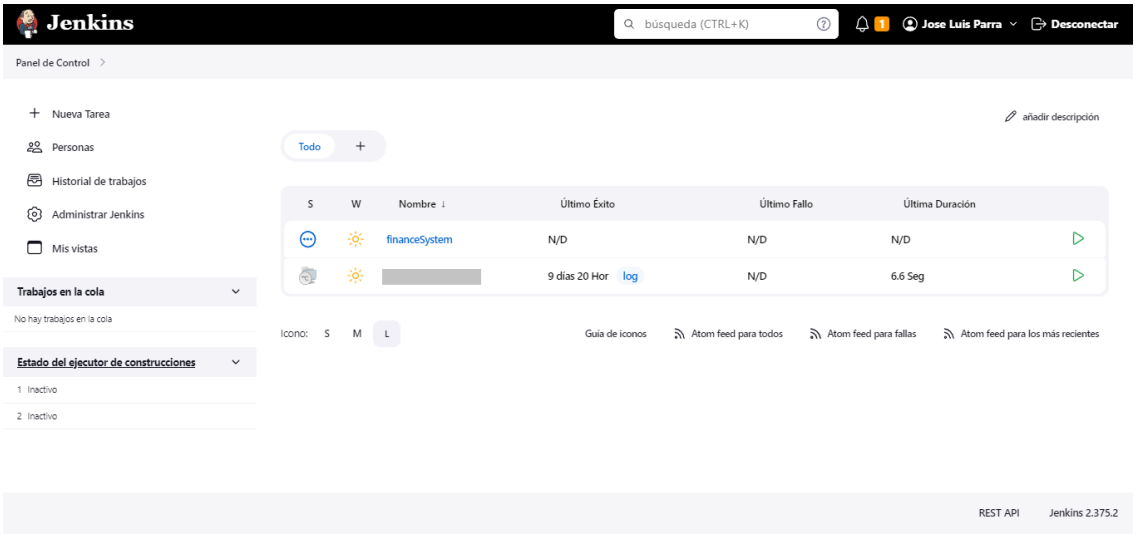

Figura 72. Listado de Jobs

Configuración del Pipeline

El Pipeline es un archivo con configuraciones secuenciales que se ejecutaran en orden, y dependiendo si cada estado o etapa es exitosa continúa a la siguiente, caso contrario se detiene el Job y lo marca como error. En la Figura 73 se observa el Pipeline.

```
pipeline {
    agent any

    options {
        timeout(time: 10, unit: 'MINUTES')
    }

    environment {
        ARTIFACT_ID = "jose0112luis/financesystem:${env.BUILD_NUMBER}"
    }

    stages {
        stage('Build') {
            steps {
                script {
                    dockerImage = docker.build "${env.ARTIFACT_ID}"
                }
            }
        }

        stage('Publish') {
            steps {
                script {
                    docker.withRegistry("", "DockerHubFinanceSystem") {
                        dockerImage.push()
                    }
                }
            }
        }
    }
}
```

Figura 73. Archivo Jenkinsfile

Ejecución del Job

El Job creado anteriormente se ha ejecutado y completado correctamente, pasando por todas las etapas configuradas. En la Figura 74 se observa que en la ejecución número 1, 3 y 4 se detuvo la ejecución debido a que el proceso de publicación en DockerHub sobrepasó el tiempo definido en el Pipeline.

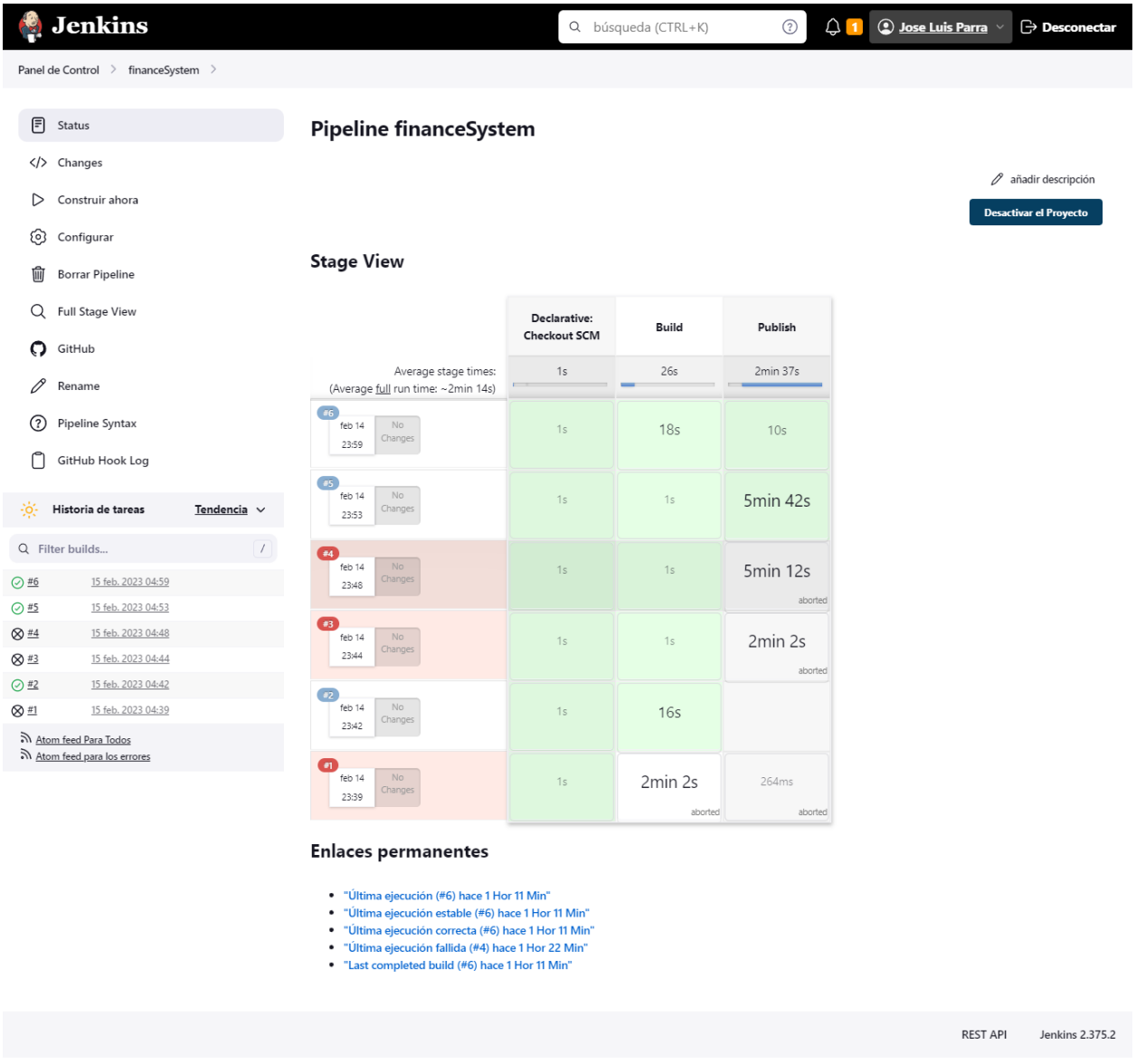

Figura 74. Ejecución del Job

Y finalmente al completar el último proceso, el Job realizó la publicación exitosa del sistema en DockerHub. En la Figura 75 se observan las imágenes cargadas a la cuenta de DockerHub.

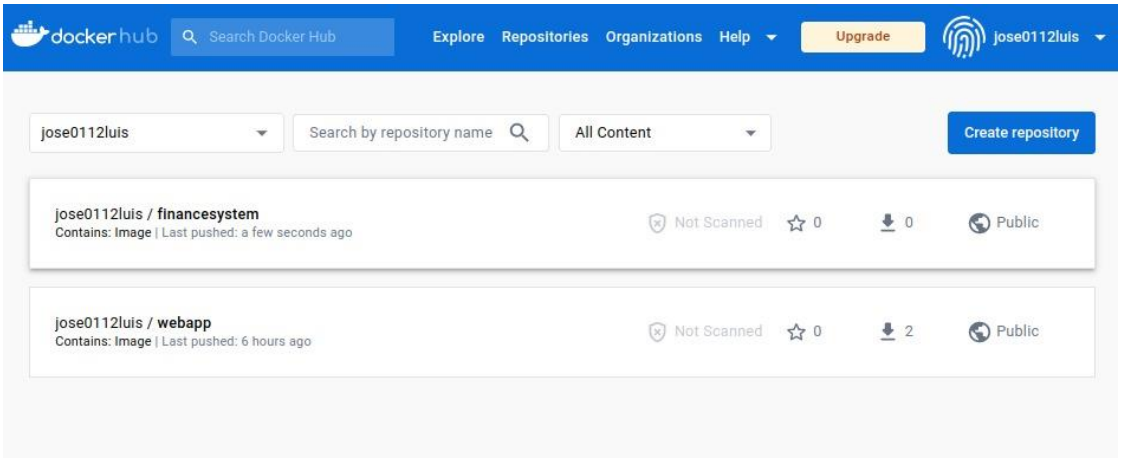

Figura 75. Repositorio de DockerHub