UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA DE COMPUTACIÓN

CONFIGURACIÓN DE UN AMBIENTE DE INTEGRACIÓN CONTINUA Y DESPLIEGUE CONTINUO PARA PROYECTOS DE DESARROLLO DE SOFTWARE EN LA CARRERA DE COMPUTACIÓN UNIVERSIDAD POLITÉCNICA SALESIANA SEDE QUITO

Trabajo de titulación previo a la obtención del
Título de Ingeniero e Ingeniera en Ciencias de la Computación

AUTORES: Jonathan Alfredo Campaña Mera

Libia Maribel Rivera Altamirano

TUTOR: Franklin Edmundo Hurtado Larrea

Quito - Ecuador

2024

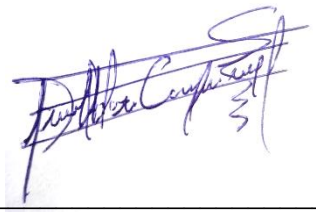
CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, Jonathan Alfredo Campaña Mera con documento de identificación N° 1719620864 y Libia Maribel Rivera Altamirano con documento de identificación N° 1727472001; manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Quito, 7 de agosto del 2024

Atentamente,



Jonathan Alfredo Campaña Mera	Libia Maribel Rivera Altamirano
1719620864	1727472001

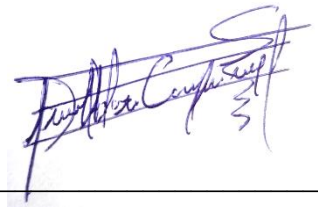
CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, Jonathan Alfredo Campaña Mera con documento de identificación N° 1719620864 y Libia Maribel Rivera Altamirano con documento de identificación N° 1727472001; expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Proyecto Técnico: "Configuración de un ambiente de integración continua y despliegue continuo para proyectos de desarrollo de software en la Carrera de Computación Sede Quito Universidad Politécnica Salesiana", el cual ha sido desarrollado para optar por el título de: Ingenieros en Ciencias de la Computación, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Quito, 7 de agosto del 2024

Atentamente,



Jonathan Alfredo Campaña Mera
1719620864

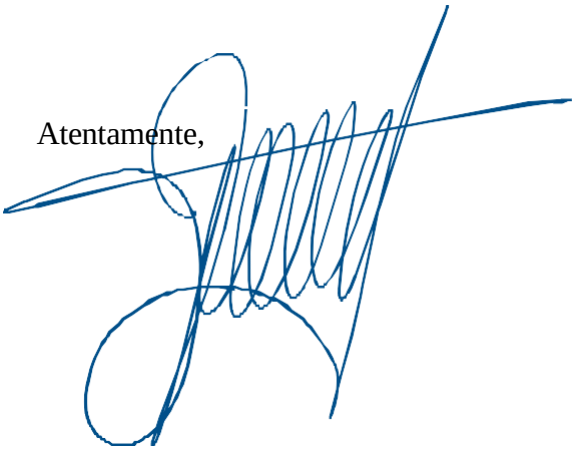
Libia Maribel Rivera Altamirano
1727472001

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Franklin Edmundo Hurtado Larrea, con documento de identificación N° 1713382016, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: CONFIGURACIÓN DE UN AMBIENTE DE INTEGRACIÓN CONTINUA Y DESPLIEGUE CONTINUO PARA PROYECTOS DE DESARROLLO DE SOFTWARE EN LA CARRERA DE COMPUTACIÓN UNIVERSIDAD POLITÉCNICA SALESIANA SEDE QUITO, realizado por Jonathan Alfredo Campaña Mera con documento de identificación N° 1719620864 y por Libia Maribel Rivera Altamirano con documento de identificación N° 1727472001, obteniendo como resultado final el trabajo de titulación, bajo la opción de Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Quito, 7 de agosto del 2024

Atentamente,



Ing. Franklin Edmundo Hurtado Larrea, MSc.

1713382016

DEDICATORIA

Dedicaré este proyecto a mis padres quienes con su incondicional apoyo y sacrificio han sido mi fuente principal de inspiración y fortaleza a mis hermanas por su confianza inquebrantable, por su motivación que posibilitaron que esto suceda.

Dedico a toda mi familia que estuvo para mí en los momentos que más les necesite brindándome su amor y sabiduría para seguir adelante en este difícil proceso académico. Sin ustedes este proyecto no sería posible sus consejos me han ayudado a mantenerme firme en mis objetivos y a no perder de vista mis metas.

Sin ustedes, este proyecto no sería posible porque cada uno de ustedes ha contribuido significativamente a este logro por ello, les dedico esta tesis con todo mi amor y gratitud.

Jonathan Alfredo Campaña Mera

AGRADECIMENTO

Empiezo agradeciendo a mi madre por su infinito cariño, apoyo y paciencia incondicional, a mi padre por sus consejos y sacrificios que ha hecho por mí y por toda mi familia. Este proyecto me ha enseñado a ser constante a pesar de todos los problemas presentados en mi vida.

Agradezco infinitamente a mis hermanas que han sido mi consuelo y aliento en todo momento, gracias por no dejarme recaer han sido mi guía en este proceso.

Agradezco a Dios por darme fuerzas para seguir adelante con mi carrera universitaria, gracias por la salud que es fundamental para que este proyecto se lleve a cabo.

A mis familiares más allegados que compartieron conmigo esta etapa académica que me ayudo a crecer personalmente y sobre todo en el ámbito profesional que sé que me ayudara de mucho.

Gracias a la Universidad por brindarme la oportunidad de crecer académicamente y a todos los docentes que contribuyeron a mi formación, brindándome sus conocimientos y compartiendo sus experiencias con nosotros.

Finalmente, a todas las aquellas personas que de una forma u otra me ayudaron a la realización de este trabajo, cada palabra y gesto de apoyo por más pequeño que sea ha sido de mucha ayuda para mí.

Jonathan Alfredo Campaña Mera

DEDICATORIA

A mis padres cuya guía, apoyo y amor han sido la base de todas mis aspiraciones y logros, ustedes me enseñaron el valor del esfuerzo, la perseverancia y la honestidad.

A mi hermano por su apoyo constante y por sus sabias enseñanzas que día a día me ayudaron a sobrellevar las situaciones complicadas en mi vida.

Este proyecto le dedico a mi familia por ser el pilar fundamental en mi vida, por su apoyo incondicional durante este proceso en la universidad y a todas aquellas personas que estuvieron ante cualquier situación ayudándome siempre, que confiaban en mí y en mis capacidades para poder alcanzar mis metas, y culminar con esta etapa de manera exitosa.

Libia Maribel Rivera Altamirano

AGRADECIMIENTO

Al concluir con esta etapa agradezco infinitamente a todos aquellos que han hecho posible la realización de este proyecto especialmente a Dios por ser mi guía, por la salud y sabiduría para poder seguir mi camino a pesar de todas las dificultades presentadas en este proceso.

Expreso mi más sincero agradecimiento a mi familia, pilar fundamental de mi vida y fuente de apoyo. A mis padres, por su amor incondicional, sus sacrificios y su fe en mí, gracias por los sabios consejos que han sido mi guía constante, le agradezco a mi hermano quien es mi compañero de vida gracias por brindarme su sabiduría, su apoyo en todo momento y por esas risas que aligeraron este arduo camino que me ayudaron a superar los problemas presentados.

Gracias a todas las personas cercanas a mí por brindarme su apoyo cada vez que he necesitado porque sin ustedes todo esto no sería posible, gracias con lo mucho o poco que me han ayudado fue muy importante para mí.

Quiero agradecer a nuestro tutor por guiarnos constantemente en esta tesis, que con su ayuda y colaboración hemos podido completar este proceso, le agradezco su paciencia y comprensión en todo momento.

Libia Maribel Rivera Altamirano

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

Este proyecto se enfocó en la configuración de un ambiente automatizado para la optimización del proceso de desarrollo de software, con concentración en aplicaciones Java. Esto incluye la compilación del código, la realización de pruebas y el despliegue de la aplicación, buscando una adecuada gestión de cambios y control de versiones, reduciendo así el tiempo de entrega de software funcional. Con esta automatización los desarrolladores, en este caso, estudiantes de la carrera de Computación de la Universidad Politécnica Salesiana, se podrán enfocar en tareas que agreguen mayor valor al producto. Este ambiente de Integración Continuo y Despliegue Continuo (IC/DC) incorpora las siguientes herramientas:

- Visual Studio Code que es el entorno de desarrollo, proporcionando un ambiente configurable que soporta Java.
- GitLab para la gestión de código fuente proporcionando un adecuado control de versiones, además de proporcionar la colaboración a través de sus funcionalidades de Merge Request.
- Jenkins como servidor de automatización y ejecución de tareas, permitiendo la IC/DC, esta herramienta automatiza varios procesos como construcción del proyecto, las pruebas y despliegue, verificando que el código se valide y se despliegue en todos los entornos.
- SonarQube para analizar el código, identificando errores que podrían causar fallos en el programa.
- JUnit para la realización de pruebas en las cuales el desarrollador coloca su propio código donde facilite la detección temprana de errores.

Palabras clave: Integración y Despliegue Continuo (IC/DC), configuración Merch Request, automatizar, versionado, pipeline.

ABSTRACT

This project focused on the configuration of an automated environment for the optimization of the software development process, concentrating on Java applications. This includes code compilation, testing and deployment of the application, looking for an adequate change management and version control, thus reducing the delivery time of functional software. With this automation, developers, in this case, students of the Computer Science program at the Salesian Polytechnic University, will be able to focus on tasks that add greater value to the product. This Continuous Integration and Continuous Deployment (CI/CD) environment incorporates the following tools:

- Visual Studio Code is the development environment, providing a configurable environment that supports Java.
- GitLab for source code management providing adequate version control, in addition to providing collaboration through its Merge Request functionality.
- Jenkins as an automation and task execution server, enabling CI/DC, this tool automates several processes such as project building, testing and deployment, verifying that the code is validated and deployed in all environments.
- SonarQube to analyze the code, identifying errors that could cause bugs in the program.
- JUnit for performing tests in which the developer places his own code where it facilitates the early detection of errors.

Keywords: Integration and Continuous Deployment (IC/DC), Merch Request configuration, automate, versioning, pipeline.

CAPÍTULO I

ANTECEDENTES Y GENERALIDADES

1.1. Introducción

La Integración Continua (IC) y el Despliegue Continuo (DC) han ido modificando la manera en la cual se desarrolla y despliega el software. En la actualidad la velocidad, la calidad y la eficiencia son esenciales, esta implementación se ha convertido en una automatización necesaria para las organizaciones que buscan mantenerse competitivas.

La investigación de conceptos básicos de IC/DC, junto con la exploración de herramientas open source establece una base para la implementación exitosa de este ambiente, propósito de este proyecto. La selección de herramientas como GitLab para el control de versionado, Jenkins para IC/DC y herramientas de prueba constituyen la base para la configuración de un ambiente de desarrollo automatizado. ilimit (2020).

1.2. Problema de estudio

La Integración Continua (IC) y Despliegue Continuo (DC) son procedimientos y tecnologías que ayudan a incrementar la eficiencia de los equipos de desarrollo, en un entorno académico como el de los estudiantes de la carrera de Computación de la Sede Quito, esto sería de vital importancia tanto para el aprendizaje de estas tecnologías como para optimizar el tiempo en sus desarrollos.

La automatización de procesos relacionados con IC/DC es el enfoque central en este proyecto de manera que no solo se vea una mejoría en la eficiencia sino una mejor colaboración de los equipos de desarrollo.

Ambientes automatizados como este son ampliamente utilizados en la industria del software, aprender estas tecnologías aportará a que los estudiantes estén mejor preparados para las expectativas y demandas del mercado laboral actual, mejorando la posibilidad de empleabilidad. En el entorno educativo no solo se esperaría que proporcione a los estudiantes habilidades y experiencias en el desarrollo de software, sino que también se optimice el tiempo de sus desarrollos.

Permite a los estudiantes experimentar un flujo de trabajo más ágil lo que hace valioso en un entorno académico donde el tiempo para desarrollar proyectos suele ser limitado. Atlassian (2024)

1.3. Delimitación geográfica

Este proyecto está alojado en el Data Center de la carrera de Computación de la Universidad Politécnica Salesiana Campus Sur.

1.4. Justificación

La implementación de este entorno ofrece beneficios como detectar errores mediante pruebas automatizadas con el fin de evitar posibles fallos en el despliegue del aplicativo. La IC ayuda a integrar de manera colaborativa el código y se pruebe de manera constante, reduciendo la posibilidad de conflictos y errores inesperados. La automatización de un flujo de tareas como la compilación, las pruebas y el despliegue liberaría tiempo para que los estudiantes se enfoquen en aspectos más creativos e innovadores que aporten más al desarrollo de software.

Los ciclos de desarrollo más cortos y eficientes permiten una entrega más rápida de software funcional, lo cual es importante en entornos académicos con plazos más estrictos. Aprender y aplicar IC/DC prepararía a los estudiantes con habilidades y experiencias directamente aplicables en el mercado laboral, este tipo de entornos permiten a los estudiantes experimentar con nuevas ideas y enfoques sin el riesgo de interrumpir el desarrollo principal.

1.5. Objetivos

1.5.1. Objetivo general

- Configurar un ambiente de IC/DC utilizando herramientas de distribución libre para apoyar la automatización de los procesos de compilación, pruebas y despliegue de aplicaciones de software desarrolladas en la carrera de Ciencias de la Computación.

1.5.2. Objetivos específicos

- Investigar y analizar conceptos básicos de IC/DC además de herramientas libres y prácticas de automatización para eliminar procesos manuales en IC/DC.
- Configurar un ambiente de IC/DC para automatizar la compilación, ejecución de pruebas y despliegue de los proyectos de desarrollo de software.
- Realizar pruebas del ambiente IC/DC configurado.

1.6. Alcance

El proyecto se enfoca en seleccionar y utilizar herramientas libres para la configuración de un ambiente de IC/DC. Se integran herramientas para el proceso de IC/DC, incluyendo IDEs de programación, repositorio, ambiente de testing, y despliegue.

Realizar este proceso implica varios pasos esenciales para garantizar una implementación exitosa, para ello se debe tomar en cuenta lo siguiente:

- IDE (Entorno de Desarrollo Integrado)

El IDE seleccionado debe ser compatible y que permita integración con el resto de las herramientas que se utilizarán dentro de este ambiente IC/DC. Debe permitir a los

desarrolladores trabajar eficientemente en colaboración con el repositorio escogido, facilitar la escritura de pruebas y scripts de automatización, y ofrecer cambios inmediatos.

- Test (Pruebas):

Las pruebas se deben integrar en este ambiente con el fin de detectar errores y no comprometer el ambiente desplegado y monitorear que los cambios introducidos no rompan la funcionalidad existente.

Las herramientas de prueba seleccionadas deben ser compatibles e integrables con las demás herramientas utilizadas en este ambiente. Estas herramientas pueden incluir marcos de pruebas unitarias, marcos de pruebas de integración, herramientas de pruebas de rendimiento, entre otras. Es esencial que estas herramientas puedan ejecutarse de manera automática como parte del proceso de IC/DC.

Algunas consideraciones importantes al seleccionar herramientas de prueba son la capacidad de escribir y ejecutar pruebas de forma rápida y eficiente, la generación de detalles sobre el estado de las pruebas y la integración con el proceso de construcción automatizado.

- Despliegue (Ciclo final):

Al finalizar el ciclo de desarrollo, integración y pruebas. Después de que el código ha sido escrito, probado y confirmado como apto para ser implementado, el proceso de despliegue se ejecuta para llevar el software desde el entorno de desarrollo hasta el entorno de producción.

Se prueba la funcionalidad del proceso de IC/DC, asegurando que la configuración del entorno sea correcta y que los flujos de trabajo se ejecuten según lo esperado.

Se verifica que las pruebas automatizadas se ejecuten correctamente y que los despliegues automáticos se realicen sin problemas.

CAPÍTULO II

MARCO TEÓRICO

2.1.Bases históricas

El desarrollo de software ha sido impulsado por la demanda de productos de software más rápidos y eficientes. A lo largo de la historia, varias tecnologías han surgido para abordar estos problemas. Por la alta competencia y el aumento de demanda en productos digitales, las organizaciones comenzaron a comprender la importancia de implementar cambios y mejoras de manera rápida y eficiente.

Martin Fowler y otros expertos empezaron a popularizar el concepto de IC/DC a mediados de la década de 2000. Este ambiente automatiza la integración de código de varios desarrolladores en un mismo repositorio, lo que ayuda a detectar y corregir errores de integración. A medida que la IC se popularizaba, surgieron herramientas como Jenkins en 2005, que permitían a los equipos implementar prácticas de IC de manera más eficiente y efectiva. El Despliegue Continuo, que automatiza la entrega de software a entornos de producción después de las pruebas, se agregó a este ambiente IC. Esto permite la entrega de cambios en el software de manera más rápida. Tecdelainfosite (2017)

Es importante entender los siguientes conceptos para crear procesos más fluidos, reducir errores e intentar disminuir el tiempo de entrega de software.

2.1.1. Integración en el contexto de desarrollo de software

La integración es la unión entre algunos sistemas o herramientas, de forma que varias áreas trabajen de manera asociada y automatizada. Es muy importante en el desarrollo de software y se puede dividir en varios enfoques como:

I. Integración de equipo

Se refiere a las prácticas y herramientas utilizadas para mejorar la colaboración entre los miembros de un equipo de desarrollo esto incluye:

- La comunicación
- Gestión de proyectos
- Colaboración en documentos

II. Integración de aplicaciones

Se refiere al proceso de enlazar diferentes aplicaciones de software con el fin de que puedan trabajar juntas y compartir datos, que pueden ser:

- APIs
- Servicios web
- Integración de datos

III. Integración de código

Esta integración se centra en combinar el código de diferentes desarrolladores y verificar que todo funcione correctamente. Que puede incluir:

- Control de versiones
- Integrar cambios en el código
- Desplegar cambios en un entorno de producción
- Revisión de código. sydle (2022)

2.1.2. Integración Continua (IC)

La IC es automatizar los cambios de código por parte de desarrolladores en un solo proyecto, es una práctica recomendada de DevOps que permite a los desarrolladores cambiar el código de un repositorio de donde posteriormente se hará la compilación y se realizará

pruebas. Las herramientas automatizadas sirven para verificar que los cambios de código sean correctos antes de que se integre con más herramientas. MAX (2024)

2.1.3. Despliegue Continuo (DC)

El DC es una práctica de desarrollo de software que consiste en liberar automáticamente cada cambio válido en el código directamente a producción. En este enfoque los cambios pasan por un proceso automatizado de pruebas y verificaciones, permite entregas más rápidas y frecuentes, reduciendo el tiempo entre el desarrollo y el feedback de los usuarios. ibm (s.f.)

2.2.IC/DC

La IC/DC son componentes importantes del desarrollo de software que impulsan una entrega de código más rápida, reducen riesgos y mejoran la colaboración entre equipos de desarrollo. Al implementar este ambiente IC/DC y aprovechar las herramientas adecuadas, se puede mantener un panorama competitivo optimizando el proceso de desarrollo de software con el fin de satisfacer de manera más efectiva las demandas de los usuarios. redhat (2022).

Algunos de los conceptos que se debe tomar en cuenta son:

2.2.1. Rama

Una rama en el contexto de IC/DC se refiere a una versión separada del código fuente dentro del mismo proyecto. En un flujo de IC/DC la gestión de ramas asegura una integración ordenada y un despliegue eficiente. Se utilizan varios tipos de ramas para gestionar el desarrollo y la implementación del código. Algunos tipos de ramas son:

1. **Main**

    - Es la rama principal del proyecto.
    - Contiene código estable y listo para producción.

- Se actualiza generalmente después de que las características han sido probadas y aprobadas.

2. **Feature:**

    - Estas ramas son utilizadas para desarrollar nuevas funcionalidades.
    - Se ramifican desde la rama Main y de fusionan de vuelta una vez completadas.
    - Además, permiten el desarrollo en paralelo de múltiples características.

3. **Develop**

    - Es la rama de integración para varias características.
    - Sirve como base para el desarrollo continuo.
    - Se fusiona con la rama Main cuando esta lista para lanzamiento.

4. **Release**

    - Esta rama es preparada para un nuevo lanzamiento
    - Es utilizada para pruebas finales y correcciones de varios errores antes de fusionarse con una rama Main.

Algunos de los beneficios de la gestión de ramas en IC/DC:

- Al trabajar en ramas específicas, se asegura que el código se integre de manera ordenada y estructurada.
- Facilita la colaboración entre desarrolladores al definir roles y reglas claras para cada tipo de rama.
- Permite ejecutar pruebas automatizadas en diferentes fases del desarrollo y despliegue, verificando que el código cumpla con los estándares establecidos.

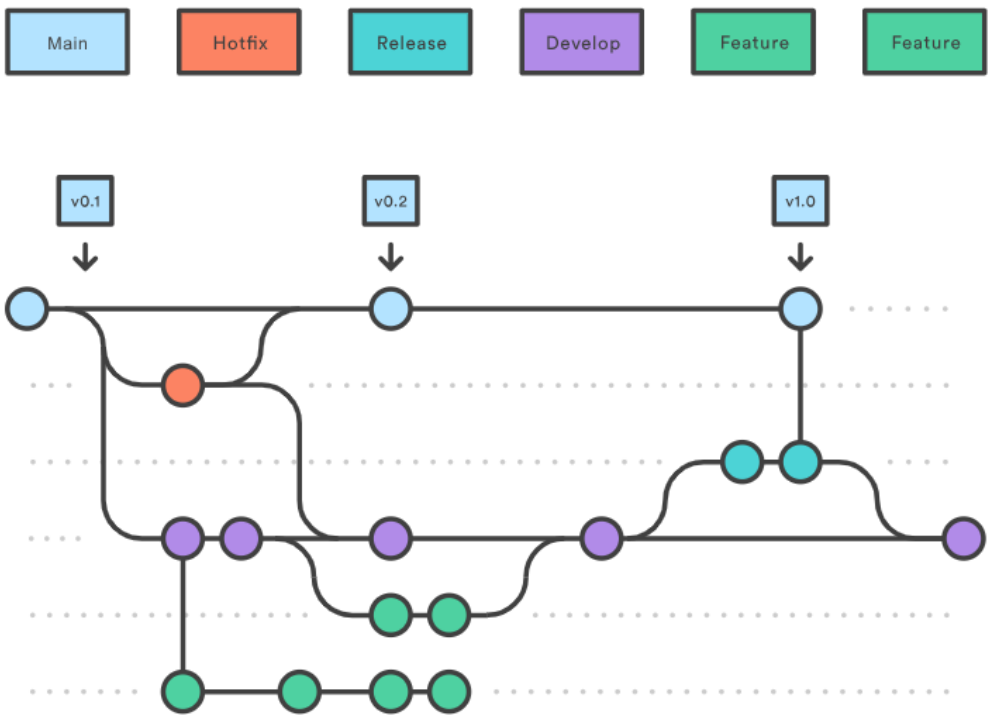
GitFlow es popular por promover un flujo continuo de integración y por simplificar el lanzamiento y mantenimiento de nuevas versiones de software de forma estructurada.

Gitflow y su gestión de ramas ayudan a mantener un flujo de trabajo organizado y eficiente en este tipo de entornos IC/DC, promoviendo el mantenimiento de nuevas versiones de manera estructurada. Atlassian (2024)

La **Figura 1** muestra un diagrama del método Gitflow, que es una estrategia de ramificación para el manejo de repositorios Git en el desarrollo de software.

Figura 1

Método Gitflow



Nota. Método Gitflow. Fuente*:* Atlassian (2024)

2.2.2. Pipeline

Un pipeline es una serie automatizada de varios procesos que permite a los programadores probar, compilar y desplegar su código eficientemente. Es un conjunto de datos de pasos automatizados que hacen que el código pase desde el repositorio hasta la producción.

Su propósito es reducir la intervención manual y asegurar que cada cambio pase por el mismo proceso, permite detectar y corregir errores. Walther (2023)

Funciones de un pipeline

Automatizar una serie de etapas que se ejecutan en secuencia de acuerdo a los requerimientos establecidos.

Permiten que los cambios de código sean integrados y desplegados automáticamente hasta que llegue a la etapa de producción. Borges (2023)

2.2.3. Merge request

También conocido como Pull Request es una característica de las plataformas de gestión de código como GitLab, GitHub y Bibucket. Un Merge Request es una solicitud para fusionar cambios desde una rama de trabajo (branch) en otra, generalmente la rama principal como Main o Master.

- Permite a otros desarrolladores revisar el código propuesto antes de que sea integrado a la rama principal.
- Permite que los cambios se prueben automáticamente mediante herramientas de IC antes de fusionarse.

Funcionamiento

1. El desarrollador construye una nueva rama en el repositorio.
2. El desarrollador realiza los cambios en la rama y los confirma (commit) regularmente.

3. Una vez que los cambios están listos, el desarrollador abre un Merge Request en la plataforma de gestión de código.
4. Otros desarrolladores revisan el Merge Request. Pueden comentar, sugerir mejoras o solicitar cambios adicionales.
5. Las herramientas de IC ejecutan pruebas automatizadas para asegurarse de que los cambios no destruyan lo que ya está planteado.
6. Si todo está bien, los revisores aprueban el Merge Request.
7. Los cambios se fusionan en la rama principal del proyecto. Esto a menudo se realiza automáticamente después de la aprobación y las pruebas exitosas. Primo (2022)

2.2.4. Stage

Se refiere a un entorno o etapa especifica en el flujo y despliegue del desarrollo, es un paso importante antes de llevar el código a la etapa de producción.

Ayudan a organizar diferentes etapas del pipeline de IC/DC, permitiendo una ejecución ordenada y controlada de tareas como por ejemplo compilación, pruebas, etc. Cada uno de estos puede contener varios jobs es decir varias tareas que se ejecutan en ese contexto. b2core (s.f.)

Algunos de los stage más comunes son:

1. Build(construcción)

   Compila el código fuente y genera paquetes
2. Test (pruebas)

   Ejecuta pruebas automatizadas para asegurar la calidad de código
3. Deploy (despliegue)

   Despliegue a entornos de desarrollo, pruebas, Staging o producción.

4. Stage

   Despliega el software en un entorno igual al de producción con el fin de realizar pruebas finales.

5. Production (producción)

   Despliega la aplicación en el entorno de ejecución.

2.2.5. Maven

Aunque Maven no actúa como una librería desempeña un rol importante en la gestión de librerías y dependencias para el proyecto. Maven se encarga de:

- Descargar e incluir librerías y frameworks necesarios para el proyecto.
- Actualizar las versiones de las librerías de manera sencilla mediante la modificación del pom.xml.

Es decir que es una herramienta de construcción y gestión que facilita la administración de dependencias y la automatización del proceso de construcción en proyectos Java. Aguín (2022)

2.2.6. Devops

DevOps integra el desarrollo de software y las operaciones de TI con el objetivo de mejorar la colaboración y cumplir con un proceso IC/DC. Se dió como una solución a la necesidad de superar los desafíos enfocados al desarrollo y al equipo de desarrollo.

DevOps busca:

- La cooperación entre los integrantes del equipo de desarrollo y operaciones.

- Automatizar procesos durante el ciclo de vida del desarrollo de software desde la etapa de construcción hasta el paso de implementación.
- Entrega continua mediante el proceso IC/DC.
- Mejora continua donde los equipos requieren optimizar procesos.

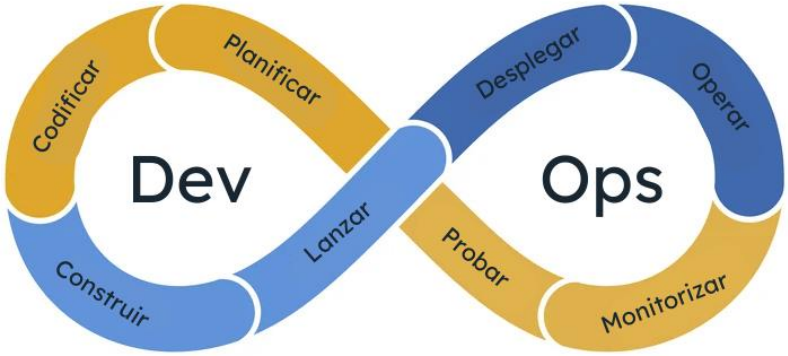
Ventajas de DevOps

- Ayuda a identificar y resolver problemas en la fase de desarrollo, mejorando la calidad de software.
- Reduce el riesgo de errores humanos, lo que permite que exista menos fallas y problemas al momento de estas en la etapa de producción.
- Automatiza tareas repetitivas y manuales
- Facilita la escalabilidad de aplicaciones y la infraestructura, asegurando una mayor disponibilidad de servicios.
- Permite un flujo continuo de feedback entre los usuarios, lo que ayuda a ir mejorando constantemente los procesos. unity (2024)

La **Figura 2** muestra el ciclo de vida de DevOps, destacando el proceso IC/DC en el desarrollo de software. El flujo está dividido en dos partes principales: "Dev" (Desarrollo) y "Ops" (Operaciones), que reflejan la colaboración y la automatización entre estos dos equipos.

Figura 2

Flujo del ambiente IC/DC



Nota. Flujo de ambiente IC/DC. Fuente: Duran (2023)

2.2.7. Conceptos de las herramientas para el ambiente

A continuación, se describen las herramientas que se requiere para la creación del ambiente automatizado.

- **Visual Studio Code (Vs Code)**

Editor de código que se ha convertido en uno de los IDEs (Herramienta que proporciona un entorno para desarrollo de varias aplicaciones) más populares por su flexibilidad y rendimiento. Algunas de sus características son:

1. VS Code soporta algunos lenguajes de programación incluido Java. Permite resaltado de la sintaxis y completado de código.
2. Es un sistema de extensiones que permite agregar varias funciones adicionales como soporte para varios lenguajes, depuración, herramientas de desarrollo web, etc.
3. Git integrado: permitiendo a los desarrolladores clonar repositorios, gestionar commits, etc. Flores (2022)

- **GitLab**

Repositorio de código que controla las versiones y facilita el trabajo colaborativo, brindando control y seguimiento de versiones de un proyecto. Permite gestionar distintas líneas de desarrollo, realizar revisiones de código, volver a versiones anteriores y tener avances del proyecto. Los cambios se realizan mediante ramas, donde los desarrolladores pueden trabajar de manera independiente y publicar en una rama separada sin afectar a las demás. Javier (2024)

- **Jenkins**

Herramienta de automatización que permite la ejecución de varias tareas repetitivas y la gestión del software. Facilita el IC, permitiendo realizar pruebas automáticas cada vez que se actualiza el código en el repositorio GitLab. SENTRIO (2021)

La herramienta ofrece automatización de tareas de desarrollo, ejecución de pruebas y notificación de resultados. Su alta configurabilidad y capacidad de integrarse con varias herramientas y servicios permiten a los equipos adaptar su ambiente de trabajo según sus necesidades específicas. Jenkins también fomenta la colaboración al proporcionar una interfaz centralizada para monitorear el estado de las compilaciones e implementaciones, así como acceso a registros y resultados de pruebas. Esto promueve la veracidad del progreso del desarrollo, mejorando la comunicación y la eficiencia del equipo. Inesdi (2022)

- **SonarQube**

Es una herramienta open source que verifica la calidad de código usando herramientas de análisis, y para ello es necesario su ejecución con el fin de obtener resultados que nos permitan ver los errores y poder corergirlos.

Proporciona métricas ayudando a mejorar la calidad de código, facilitando a los miembros del equipo de desarrollo monitorear y detectar errores manteniendo así el código limpio. SENTRIO (2021)

Algunos aspectos que SonarQube tiene son:

- Proporciona métricas sobre el código que se repite.
- Código inaccesible: Identifica el código que no será ejecutado.
- Errores (Bugs): Detecta problemas que impiden que el software funcione correctamente.
- Complejidad ciclomática: Evalúa el número de posibles caminos en el código. A mayor complejidad, menor comprensión.

2.3.IC/DC en entornos académicos

Incluso en los ambientes académicos, la adopción de estos conceptos puede cambiar la manera en que se gestionan los sistemas y aplicaciones, a medida que las universidades e instituciones buscan modernizar sus infraestructuras tecnológicas, la integración, la IC y DC se convierten en herramientas de ayuda para lograr los objetivos. ibm (s.f.)

Los estudiantes podrán experimentar con la gestión de versiones y el control de código fuente, aprendiendo a colaborar en proyectos grupales. La implementación de IC/DC en el ámbito académico no solo proporciona a los estudiantes habilidades y conocimientos de estos temas, sino que también los prepara para enfrentar posteriormente el mundo laboral brindándoles así una ventaja competitiva, mejorando su empleabilidad y capacidad para contribuir efectivamente en sus futuros roles profesionales. Chacón (2024)

CAPÍTULO III

METODOLOGÍA

A continuación, se describe la metodología que se usó en este proyecto, dividida en varias fases.

3.1.Investigación y análisis:

En esta fase, se investigó sobre los conceptos básicos de IC/DC, así como las herramientas disponibles para automatizar estos procesos. Se analizaron las herramientas de código abierto como GitLab, Jenkins y otras herramientas de prueba y despliegue continuo, utilizando criterios como distribución, capacidad de integración, etc.

3.2.Diseño y planificación:

Fase enfocada al diseño del ambiente de IC/DC. Esto incluyó la identificación de las necesidades del ambiente, herramientas que sean más adecuadas y la planificación de cómo se integran y configuran para automatizar este ambiente.

3.3.Implementación:

En esta fase, se lleva a cabo la configuración del ambiente IC/DC de acuerdo con el diseño elaborado en la fase anterior. Se seleccionan y configuran las herramientas identificadas para automatizar este flujo.

3.4.Pruebas del ambiente configurado:

Una vez completada la implementación del ambiente de IC/DC, se realizan pruebas para validar su funcionamiento y rendimiento. Esto incluye pruebas funcionales para verificar que la integración, compilación, pruebas y despliegue se realicen correctamente.

3.5.Documentación y corrección de problemas:

En este punto, los resultados de las pruebas se documentan, identificando cualquier problema encontrado durante el proceso de pruebas. Se corrige estos problemas y posterior a esto se realiza pruebas adicionales para garantizar que el ambiente de IC/DC esté listo para su implementación.

A lo largo de todo el proceso, se mantendrá la flexibilidad para adaptarse a cambios y ajustar según sea necesario.

CAPÍTULO IV

DESARROLLO

En este capítulo se describen todos los pasos ejecutados para la configuración del ambiente.

4.1.Investigación y análisis

Se compararon las herramientas más utilizadas en el área de IC/DC.

4.1.1. Análisis de herramientas para el ambiente automatizado

Se tomó en cuenta como punto principal que las herramientas sean open source y que se integren la una con la otra, para que al momento de configurar no haya problemas de compatibilidad.

A continuación, se detallan los criterios que se tomaron en cuenta para realizar el análisis y la posterior selección de los repositorios.

Plataforma web porque proporciona una interfaz más accesible desde cualquier dispositivo permitiendo a los desarrolladores gestionar repositorios.

Que se pueda alojar en la nube para que estén a disposición en cualquier momento.

De colaboración avanzada como los pull request (solicitudes para que se revisen y eventualmente se fusionen cambios en el código), Code Reviews (proceso de revisar el código escrito por otro desarrollador antes de que se integre en el proyecto principal) que son importantes para que faciliten la revisión de software y la integración de cambios.

IC/DC integrado porque permite automatizar la construcción, la realización de pruebas para desplegar el aplicativo, lo que acelera la entrega de nuevas funcionalidades.

Gestión de problemas con el fin de permitir al desarrollador gestionar nuevas funcionalidades directamente dentro del repositorio.

- Para escoger la herramienta de revisión de código se realizó un análisis en base a las siguientes características:

Herramienta Gratuita: Debe ser una herramienta libre de pago para que estudiantes de la carrera de computación puedan acceder a este ambiente IC/DC además de poder ser integrada con demás herramientas de este ambiente.

Análisis Dinámico: para evaluar el comportamiento del software en tiempo de ejecución

Integración IC/DC: La fácil integración con pipelines de IC/DC es importante para automatizar las pruebas y el despliegue, mejorando la eficiencia del flujo de trabajo.

Despliegue en la Nube: Las soluciones basadas en la nube ofrecen flexibilidad y escalabilidad, eliminando la necesidad de infraestructura propia y permitiendo acceso remoto.

Finalmente, para escoger la herramienta que se va a utilizar como orquestador se realizó un análisis en base a las siguientes características:

Alojamiento Local: La capacidad de alojar la herramienta localmente es importante para proyectos que requieren mayor control con restricciones en la red.

Configuración YAML: La configuración mediante archivos YAML facilita la definición de pipelines y tareas de manera estructurada y legible. Esto mejora la reproducibilidad y la gestión de configuraciones.

Plugins Extensos: La disponibilidad de plugins extensos permite personalizar y ampliar las funcionalidades de la herramienta para satisfacer necesidades específicas del proyecto.

Paralelización: permite ejecutar múltiples tareas simultáneamente, lo que permite acelerar el tiempo de ejecución de los pipelines

Repositorios

Los repositorios mejoran la eficiencia del desarrollo de software y la colaboración en equipo.

En la **Tabla 1** se puede ver la comparación de repositorios para integrar al ambiente automatizado basándose en las características ya descritas anteriormente.

Tabla 1

Comparación de repositorios

Característica	GitHub	Git	GitLab
Plataforma web	SI	NO	SI
Alojamiento en la nube	SI	NO	SI
Colaboración avanzada	SI	NO	SI
CI/CD integrado	SI	NO	SI
Gestión de problemas	SI	NO	SI

Nota. La tabla indica las características de los repositorios. Elaborado por: Los autores

Como se puede ver en la **Tabla 1** La elección entre GitHub y GitLab puede depender de las preferencias del proyecto, el flujo de trabajo del equipo y la integración con las demás herramientas usadas en este ambiente. Git de acuerdo con las características analizadas es una herramienta que no ofrece alojamiento en la nube, no tiene las características avanzadas de colaboración que ofrecen las plataformas web y no tiene características de CI/CD integradas así que para este caso no es una buena opción.

En este caso escogemos GitLab como plataforma de gestión de repositorios debido a sus funcionalidades integradas. Permite una gestión eficiente del desarrollo de software hasta la implementación.

Herramientas de revisión de código

Estas herramientas son esenciales para mantener la calidad del software.

En la **Tabla 2** se puede ver la comparación de herramientas para revisar la calidad de software dentro del ambiente utilizando las características ya descritas anteriormente.

Tabla 2

Comparación de herramientas para la calidad de software

Característica	CodeClimate	Coverity	SonarQube
Múltiples Lenguajes Soportados	SI	SI	SI
Análisis dinámico	NO	SI	NO
Integración IC/DC fácil	SI	NO	SI
Herramienta gratuita	NO	SI	SI
Despliegue en la nube	SI	SI	SI

Nota. La tabla indica las características de las herramientas para verificar la calidad de software. Elaborado por: Los autores

Según la **Tabla 2**, SonarQube es la herramienta que más se ajusta para utilizar en este ambiente ya que se integra fácilmente y es flexible para poder desplegarse en la nube. Coverity es una excelente opción si se necesita análisis dinámico además del estático y si la integración IC/DC no es una prioridad principal. CodeClimate es una opción si se busca una integración IC/DC sencilla y no es un problema pagar por la herramienta.

Es decir que SonarQube será la que utilice para el análisis de código, lo que identifica y corrige posibles problemas de calidad en el código fuente. Su integración con GitLab garantiza una revisión continua del código.

Orquestador

Estas herramientas permiten automatizar la implementación, gestión y coordinación de aplicaciones.

En la **Tabla 3** se puede ver la comparación de herramientas para orquestar el flujo utilizando las características ya descritas anteriormente.

Tabla 3

Comparación de herramientas para orquestar el flujo

Característica	Travis CI	CircleCI	Jenkins
Alojamiento local	NO	SI	SI
Configuración YAML	SI	SI	SI
Plugins Extensos	NO	NO	SI
Paralelización	SI	SI	SI, con plugins
Soporte de contenedores	SI	SI	SI, con plugins

Nota. La tabla indica las características de los orquestadores para la construcción de este ambiente. Elaborado por: Los autores

Según los datos que se muestran en la **Tabla 3**, Jenkins es la opción más completa y flexible debido a sus plugins extensos, alojamiento local, y configuraciones avanzadas. Es ideal para equipos que requieren personalización y extensibilidad. CircleCI es buena opción intermedia con soporte para alojamiento local y configuraciones mediante YAML, adecuada para la mayoría de los flujos de trabajo de IC/DC. Travis CI es adecuado para proyectos que

buscan una solución simple y eficaz de un ambiente automatizado sin la necesidad de alojamiento local.

Es decir que Jenkins se escoge como orquestador para automatizar las fases de compilación, pruebas y despliegue del desarrollo. Su integración con las demás herramientas permite orquestar y coordinar eficientemente las diferentes etapas del pipeline de DC.

4.2.Diseño y planificación

La tarea de programar es compleja ya que requiere una considerable inversión de tiempo porque se necesita realizar un análisis y planificación del problema planteado, y posterior a esto es necesario realizar pruebas funcionales, es por esto que un ambiente de IC/DC es beneficioso para los desarrolladores ya que este entorno permite automatizar muchas tareas repetitivas, como la integración de código y las pruebas, liberando así tiempo que los desarrolladores pueden dedicar a actividades más específicas y de mayor valor. Además de esto es importante tomar en cuenta los siguientes aspectos:

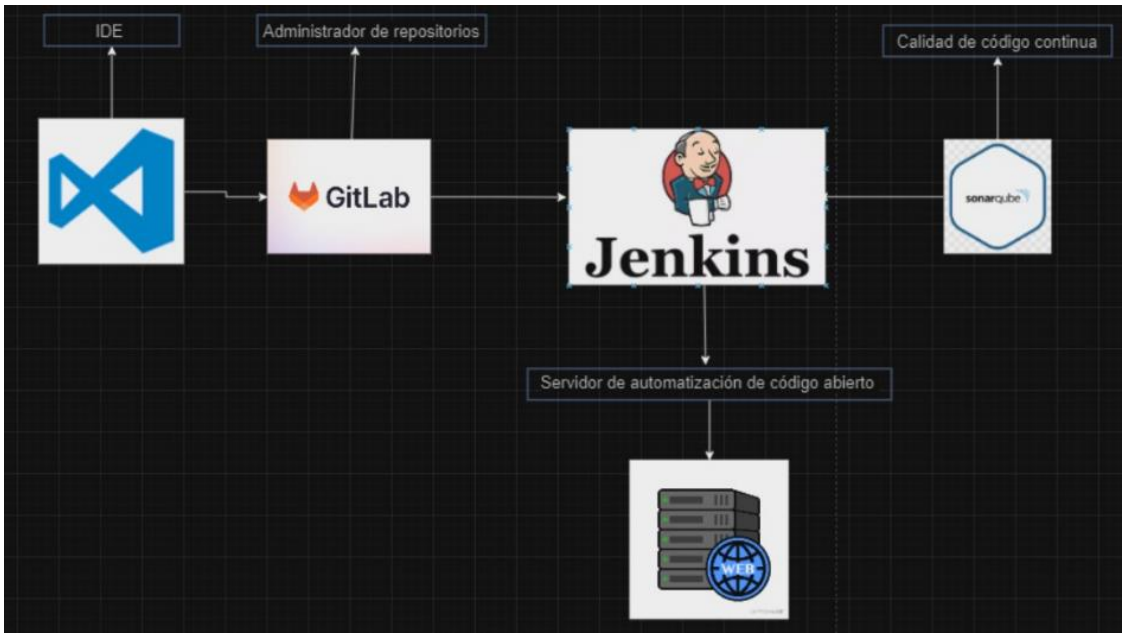
- La máquina virtual debe estar configurada con los recursos (CPU, memoria, almacenamiento) adecuados para soportar las operaciones de IC/DC y manejar la carga de trabajo generada por la integración y los despliegues automáticos.
- Para poder integrar el código en la máquina se configurará el repositorio de código con las ramas Main y Develop. La rama Develop se utilizará para las integraciones y pruebas continuas, mientras que Main contendrá el código listo para producción.
- Las herramientas utilizadas para IC/DC, como Jenkins y SonarQube, se configurarán para trabajar con aplicaciones Java ya que para este lenguaje hay una gran cantidad de herramientas y frameworks que facilitan el proceso de IC/DC, por ejemplo, Jenkins y Apache Maven que son ampliamente usadas para este tipo de flujos.

- Herramienta de Revisión de Código: Para la identificación de errores de código se implementará SonarQube que permita detectar estos problemas antes de que llegue a la etapa de despliegue.
- Herramienta de Rendimiento: Se instalará JMeter para realizar pruebas de rendimiento y carga con el fin de que nos notifique cuales son los problemas y poder realizar correcciones rápidamente.

A continuación, en la **Figura 3**, se muestra el diseño con las herramientas elegidas para crear el entorno automatizado.

Figura 3

Flujo de herramientas



Nota. Se presenta flujo de herramientas para IC/DC. Elaborado por: Los autores

En la **Figura 3** se muestra un ambiente automatizado de IC/DC utilizando herramientas open source donde los componentes del sistema son: un IDE donde los desarrolladores pueden editar su código, GitLab en este caso actúa como administrador de repositorios mediante el

cual pueden almacenar y gestionar el código, Jenkins es el servidor para automatizar las fases del ciclo de vida del aplicativo.

Jenkins toma el código desde GitLab para realizar construcciones y despliegues automáticos, después envía el código a SonarQube para el respectivo análisis de calidad, Jenkins despliega el código a un servidor web, completando así el procedimiento DC.

Pipeline

Un pipeline es un flujo de trabajo en el contexto de IC/DC es una serie de pasos automatizados que permiten la construcción, prueba y despliegue de software de manera continua y consistente. A continuación, el detalle de cómo sería el flujo de trabajo en el ambiente diseñado.

Obtención de código fuente:

Inicia con la obtención del código fuente desde un repositorio centralizado como en este caso GitLab.

Build (Construcción):

Se ejecuta la construcción del proyecto utilizando herramientas como Maven o scripts de construcción específicos. Este paso compila el código fuente y genera el artefacto binario (por ejemplo, un archivo JAR, WAR).

Pruebas unitarias:

Después de la construcción, se ejecutan pruebas unitarias automatizadas para verificar la funcionalidad a nivel de componente o módulo, cabe recalcar que estas pruebas deben estar dentro del proyecto desarrollado, es decir que son escritas por el programador. Para este propósito, se utilizó un framework para JUnit, lo que permite asegurar que cada unidad de código funcione correctamente y cumpla con los requisitos establecidos.

Análisis de código:

En algunos casos, se puede integrar un análisis estático de código para verificar estándares de codificación, identificar posibles problemas de seguridad o detectar código duplicado utilizando SonarQube.

Despliegue en entorno de desarrollo:

Una vez que las pruebas son exitosas, el artefacto construido se despliega en un entorno de desarrollo o preproducción. Aquí se verifica que la integración del nuevo código no afecte negativamente el funcionamiento general del sistema.

Despliegue en entorno de producción:

Después de pasar las pruebas de aceptación, el artefacto se despliega automáticamente en el entorno de producción. Este despliegue puede ser gradual o completo, dependiendo de la estrategia de despliegue configurada.

Todos los pasos descritos anteriormente están completamente automatizados utilizando herramientas de IC/DC como Jenkins y GitLab. Estas herramientas ejecutan cada paso en respuesta a eventos como cambios en el repositorio de código (pushes, pull requests) según una planificación definida.

Orquestación: El pipeline está orquestado de manera que cada paso depende del éxito del paso anterior. Si alguna etapa falla se detiene el proceso y se notifica al equipo para corrección.

Visibilidad y control: El equipo de desarrollo tienen acceso en tiempo real al estado del pipeline a través de paneles de control.

Iteración y mejora continua: El pipeline no es estático; se puede iterar y mejorar continuamente. Los equipos pueden agregar nuevos pasos, ajustar las configuraciones de prueba, o mejorar los tiempos de ejecución para optimizar el flujo de trabajo general.

4.3.Implementación

4.3.1. Instalación de las herramientas escogidas para el ambiente automatizado

De acuerdo con el análisis realizado anteriormente se procede a instalar y configurar las herramientas en una máquina virtual del DataCenter de la Universidad Politécnica Salesiana la cual consta con las siguientes especificaciones:

- Sistema operativo: Linux / Ubuntu
- Núcleos: 32
- Memoria RAM: 32GB
- Almacenamiento: 100 GB

En el desarrollo del proyecto se instalaron varias herramientas de integración dentro de ellas se instaló la versión 17.1.0 de GitLab como repositorio del código fuente del proyecto por lo cual es necesario constar con mínimo 2 núcleos y 8 Gb de RAM, se instalaron varias dependencias para el funcionamiento del repositorio, que se emplea para la transmisión de datos mediante URL.

Para la descarga de proyectos desde servidores remotos, se utilizó openssh-server para establecer conexiones seguras mediante el protocolo ssh, certificados de autoridad confiables, entre otras.

Se configuraron las respectivas reglas del firewall, adicionalmente se estableció el puerto 81 sobre el cual se encuentra disponible la interfaz web, y por último se configuró el usuario y contraseña mediante la interfaz web para el acceso a la herramienta.

A continuación, en la **Figura 4** se muestra la ejecución de comandos para actualizar el sistema y preparar la instalación de algunos paquetes necesarios.

Figura 4

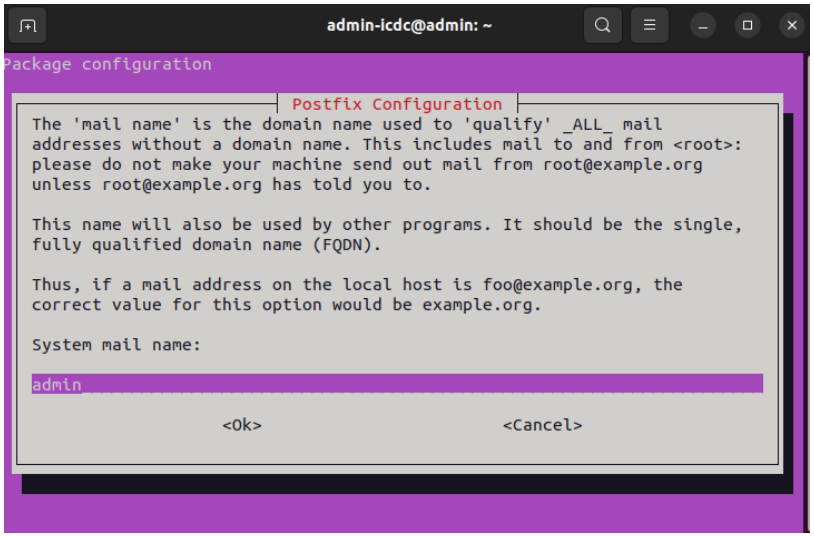
Comandos para actualizar los paquetes

Nota. Ventana de actualización de paquetes. Elaborado por: Los autores

La **Figura 5** muestra la configuración de Postfix en la terminal del servidor, donde se especifica el nombre del correo del sistema. Este nombre se utiliza para calificar todas las direcciones de correo sin un nombre de dominio completo.

Figura 5

Configuración Postfix Gitlab

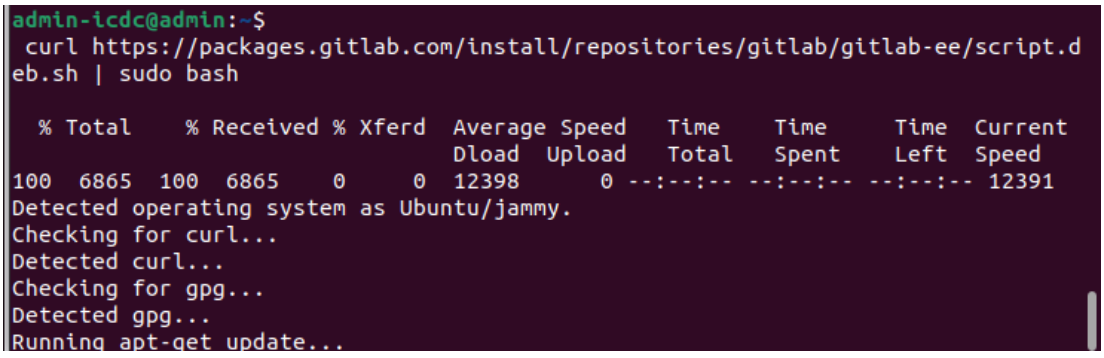


Nota. Ventana de configuración inicial. Elaborado por: Los autores.

En la **Figura 6** se muestra la descarga de un script desde el repositorio de paquetes de GitLab y ejecutándolo con privilegios de administrador para instalar GitLab Enterprise Edition.

Figura 6

Repositorio Gitlab

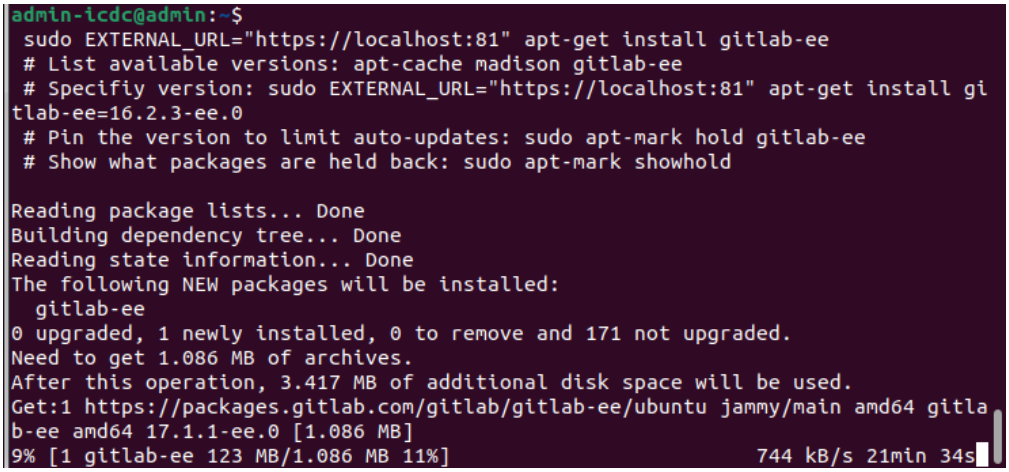


Nota Se descarga el repositorio de GitLab desde la URL directa. Elaborado por: Los autores.

Finalmente, en la **Figura 7** se muestra la instalación exitosa de GitLab Enterprise Edition en el servidor, y a la vez se configura el puerto por el cual se comunicará.

Figura 7

Configuración IP para interfaz web

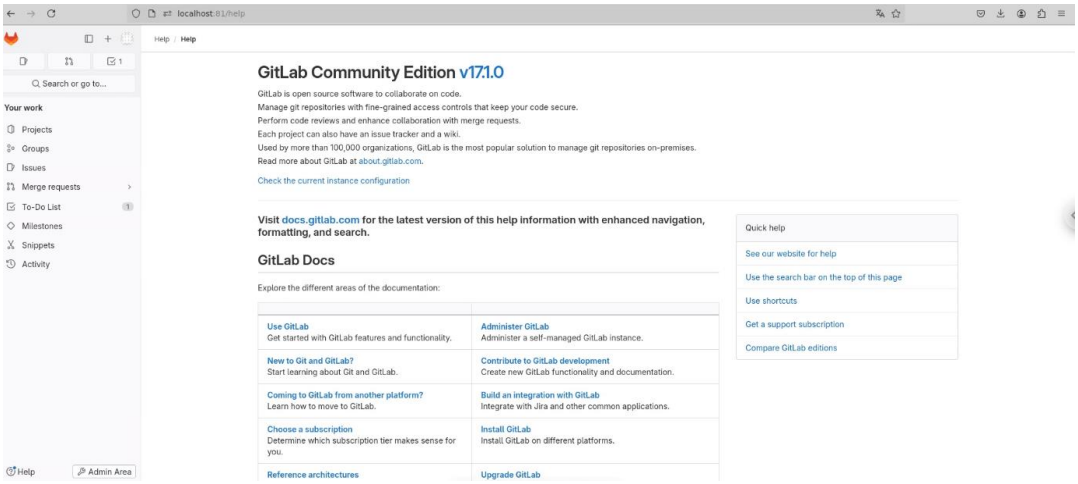


Nota. Se selecciona la dirección a la que se va a dirigir el servicio de GitLab. Elaborado por: Los autores

Una vez finalizada la instalación en la **Figura 8** se muestra la versión instalada de GitLab para la configuración de este ambiente IC/DC

Figura 8

Versión del GitLab utilizada en este ambiente

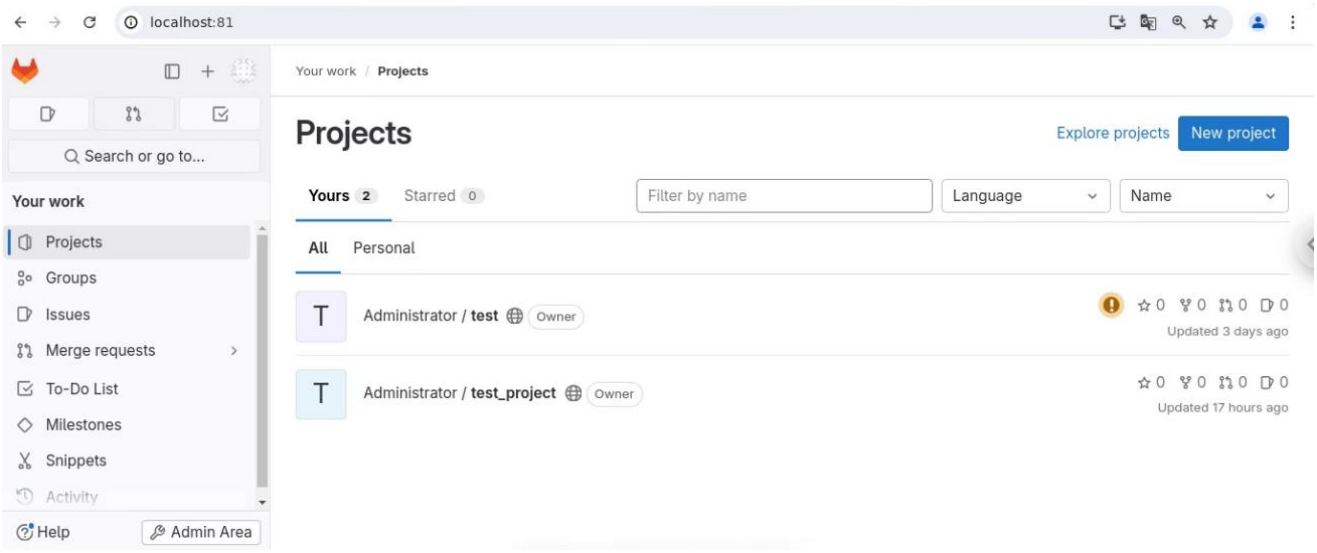


Nota. Se muestra la versión instalada de Gitlab. Elaborado por: Los autores

En la **Figura 9** se muestra la interfaz gráfica de GitLab que cumplirá con la función de integración de código en el repositorio, y para su posterior configuración con otras herramientas.

Figura 9

Ventana principal Gitlab



Nota. Se muestra la ventana principal de GitLab. Elaborado por: Los autores.

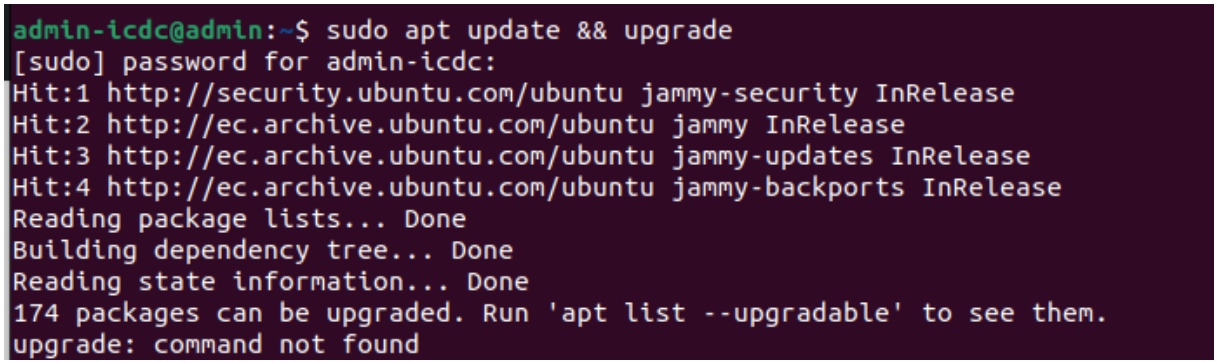
Instalación Jenkins

Para la instalación del motor de automatización Jenkins en su versión 2.452.2 se necesita de una versión de JDK superior a la 11. Una vez verificados los prerrequisitos se descargó e instaló con la clave pública del repositorio de Jenkins, esta clave certifica que la versión de la herramienta sea legitima del fabricante, adicional se agrega el repositorio a la lista para poderla instalar.

Para la instalación del motor de automatización, nuestro orquestador de herramientas se realizó una actualización de los paquetes del sistema, como se muestra en la **Figura 10**.

Figura 10

Actualización de paquetes para instalación de Jenkins



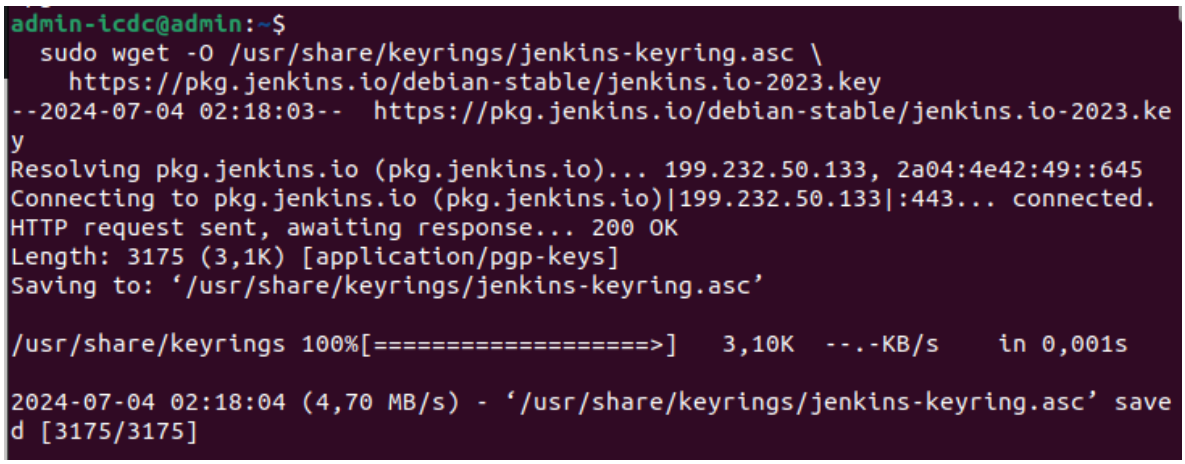
Nota. Se muestra el comando ejecutado para la actualización de paquetes de Jenkins.

Elaborado por: Los autores.

A continuación, como se muestra en la **Figura 11** se añadió el repositorio de Jenkins para instalar la herramienta desde su sitio oficial y con la versión más actual al momento.

Figura 11

Repositorio de Jenkins

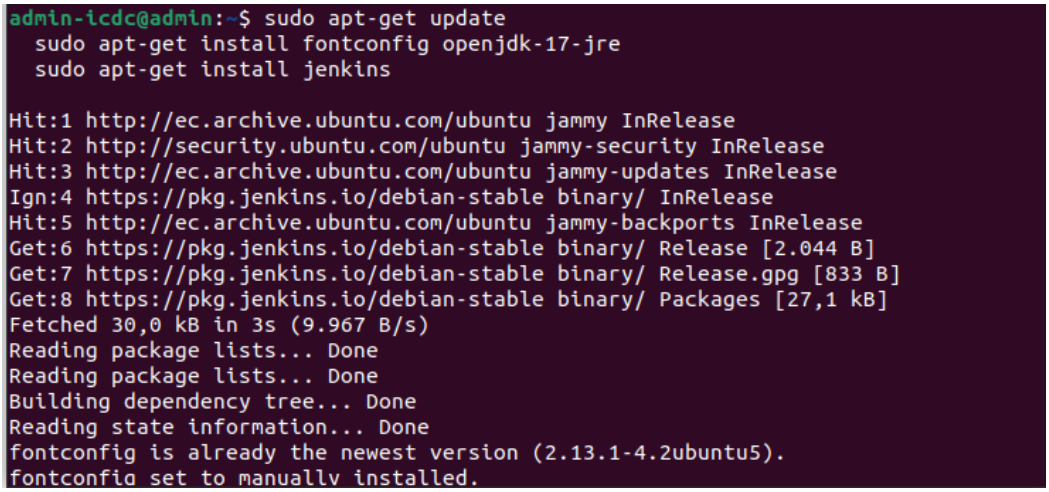


Nota. Agregación del repositorio de Jenkins. Elaborado por: Los autores.

Como requisito para la instalación de esta herramienta, se necesita agregar los paquetes e instalar una versión de fontconfig y openjdk-17-jre para la instalación de Jenkins como se puede mostrar en la **Figura 12**.

Figura 12

Comando para instalación de JDK y Jenkins

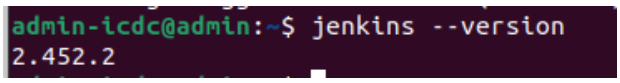


Nota. Se muestra la instalación de prerrequisitos para la instalación de Jenkins. Elaborado por: Los autores.

Se verifica la versión que se instaló en Jenkins.

Figura 13

Comando para verificar la versión de Jenkins



Nota. Se verifica que el motor este instalado correctamente mediante línea de comando. Elaborado por: Los autores.

Una vez instalada la herramienta se habilitó el puerto 8080 para la interfaz web.

A continuación, se muestra la pantalla inicial de configuración de Jenkins, específicamente la página "Desbloquear Jenkins" en la **Figura 14**.

Figura 14

Configuración de contraseña inicial de Jenkins

Nota. Ventana de desbloqueo de configuración de Jenkins. Elaborado por: Los autores

En la **Figura 15** se muestra el directorio en el que se encuentra la contraseña inicial.

Figura 15.

Contraseña ubicada en el directorio indicado

```
admin-icdc@admin:/var/lib/jenkins$ sudo su
root@admin:/var/lib/jenkins# cd secret
secret.key                secret.key.not-so-secret  secrets/
root@admin:/var/lib/jenkins# cd secretc
bash: cd: secretc: No such file or directory
root@admin:/var/lib/jenkins# ls
config.xml                     nodeMonitors.xml         secrets
hudson.model.UpdateCenter.xml  plugins                  updates
jenkins.telemetry.Correlator.xml  secret.key            userContent
jobs                           secret.key.not-so-secret  users
root@admin:/var/lib/jenkins# cd secrets
root@admin:/var/lib/jenkins/secrets# cat initialAdminPassword
80c5400e741a400eb0e37b3be412324d
root@admin:/var/lib/jenkins/secrets#
```

Nota. Se colocó la contraseña inicial para continuar con la instalación, alojada en el directorio "**/var/lib/jenkins/secrets/initialAdminPassword**". Elaborado por: Los autores.

Una vez colocada la contraseña inicial se inició con la instalación por defecto de los plugins los cuales se utilizó para la integración con las demás herramientas como se muestra en la **Figura 16**.

Figura 16

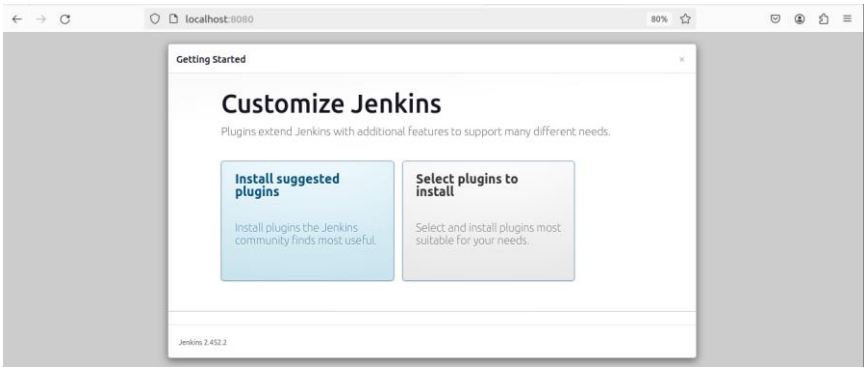
Verificación de contraseña

Nota. Verificación de contraseña de inicio de Jenkins. Elaborado por: Los autores.

Después de la verificación de la contraseña aparece una ventana de Jenkins para la instalación que en este caso se escogió la instalación sugerida como en la **Figura 17.**

Figura 17

Ventana de personalización inicio de Jenkins

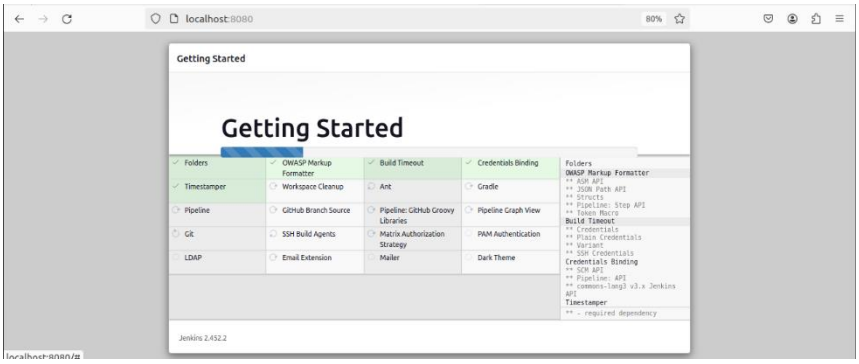


Nota. Se muestra la ventana de selección de configuración predeterminada o personalizada. Elaborado por: Los autores

En la **Figura 18** se muestra los plugins instalados definidos por defecto de la herramienta al escoger la opción de instalación de complementos sugeridos.

Figura 18

Instalación de los plugin Jenkins

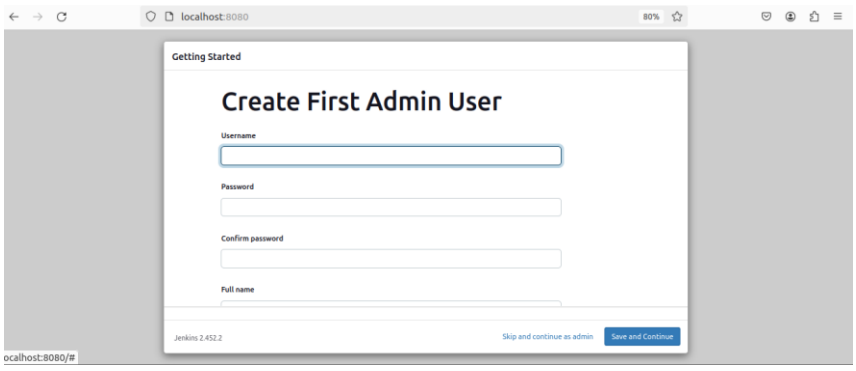


Nota. Instalación de plugins. Elaborado por: Los autores.

A continuación, en la **Figura 19** se visualiza la creación del usuario inicial en Jenkins con su respectiva contraseña.

Figura 19

Creación de usuario en Jenkins

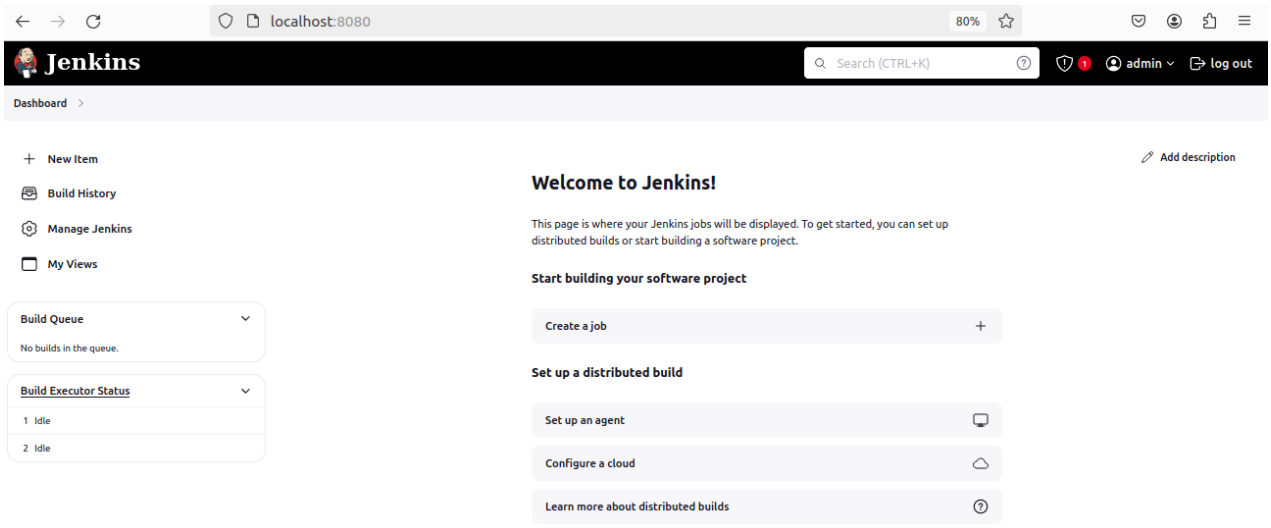


Nota. Al finalizar con la instalación de los plugins se procede con la creación de usuario y contraseña para el acceso a interfaz web. Elaborado por: Los autores.

Una vez instalado se despliega la ventana de bienvenida a la interfaz gráfica de Jenkins como se muestra en la **Figura 20.**

Figura 20

Ventana inicial de Jenkins



Nota. Ventana de inicio de Jenkins. Elaborado por: Los autores.

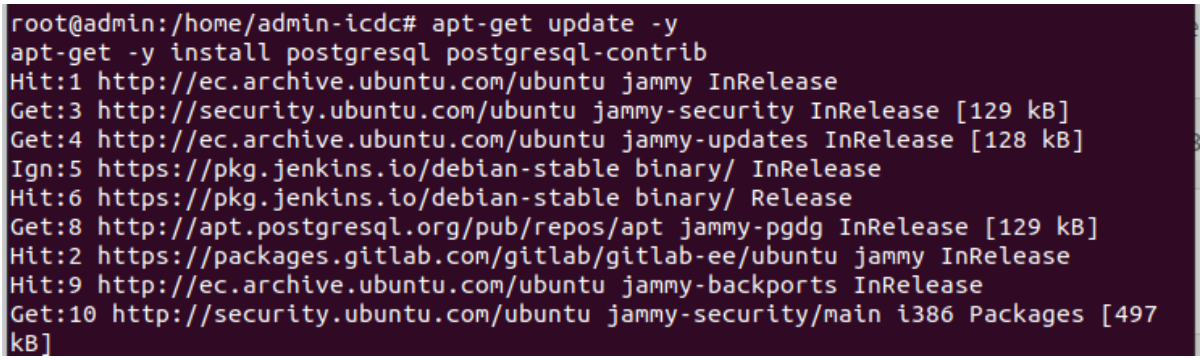
Instalación SonarQube

Para el análisis de calidad de código se utilizó SonarQube la cual para su instalación se requirió de una versión de PostgreSQL 14.12 y una versión de openjdk 17.0.11.

Para la instalación de la herramienta SonarQube se actualiza los paquetes instalados con el comando apt-get update -y como se muestra en la **Figura 21.**

Figura 21

Actualización e instalación de requisitos para la instalación de SonarQube



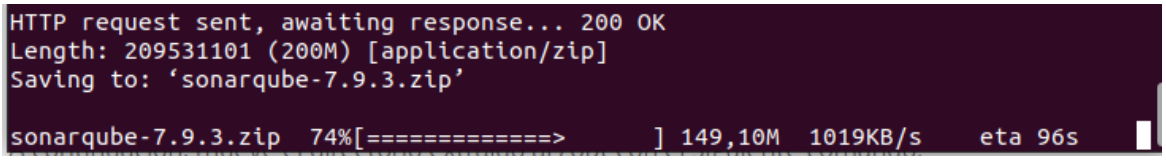
Nota. Se muestra la actualización de paquetes para la instalación de SonarQube. Elaborado por: Los autores.

Una vez con los requisitos previos listos se procedió a la actualización de todos los paquetes e instalación de la última versión de SonarQube en este caso la 10.5.1, se agrega el usuario independiente para SonarQube y se otorgan permisos al directorio /opt/sonarqube.

A continuación, se procede a configurar el usuario y contraseña para la administración de la herramienta desde la interfaz web, se verificó que todos los cambios se hayan realizado con normalidad. Para finalmente proceder con la instalación de SonarQube como se muestra en la **Figura 22**.

Figura 22

Descarga e instalación de SonarQube

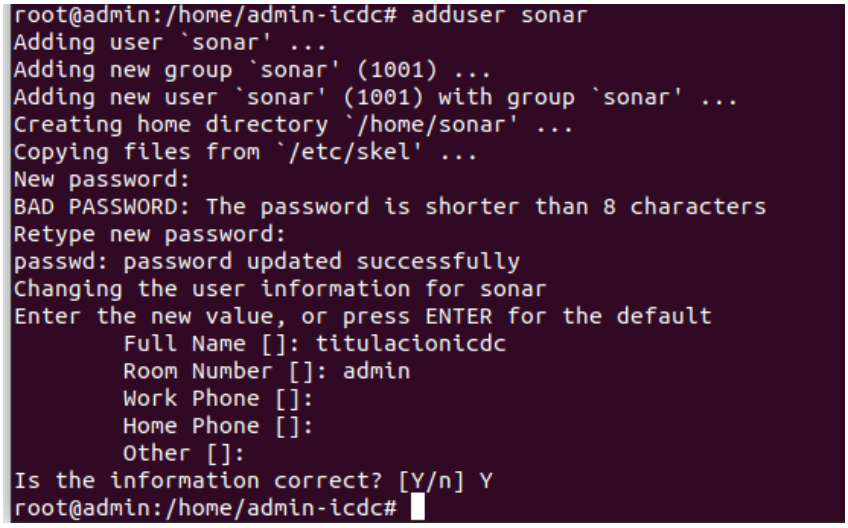


Nota. Se muestra la instalación de SonarQube. Elaborado por: Los autores.

A continuación de muestra en la **Figura 23** la creación del usuario en la herramienta SonarQube. Se verifica la información colocando la letra Y (yes).

Figura 23

Usuario SonarQube

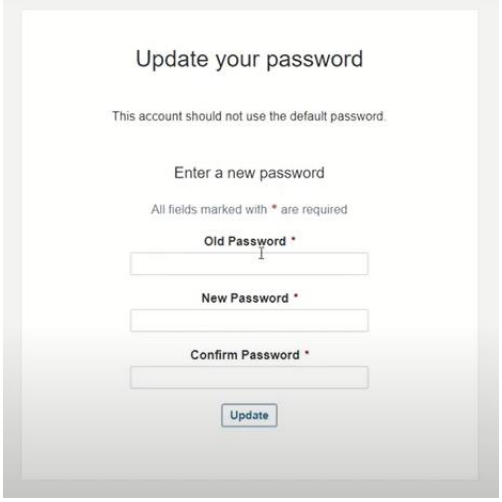


Nota. Se muestra la creación de usuario de SonarQube. Elaborado por: Los autores.

Para iniciar en la interfaz web de SonarQube, la primera vez para ingresar el usuario y contraseña por defecto siempre será user: admin, password: admin, una vez ingresada las credenciales por primera vez se procede a cambiar la contraseña y se definen las credenciales como se visualiza en la **Figura 24.**

Figura 24

Primer inicio de sesión y cambio de contraseña en Jenkins

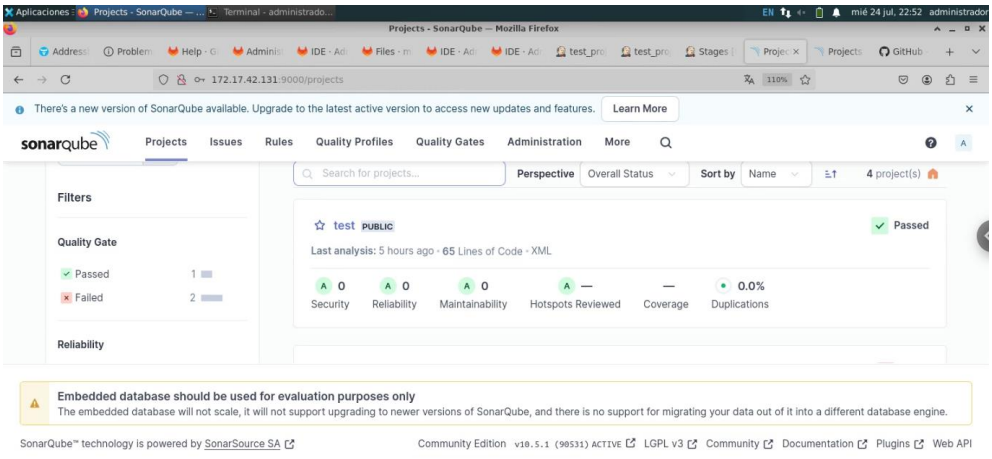


Nota. Actualización de Contraseña por primera vez SonarQube. Elaborado por: Los autores

Se verifica la versión instalada del SonarQube en la parte inferior como se muestra en la **Figura 25.**

Figura 25

Versión SonarQube

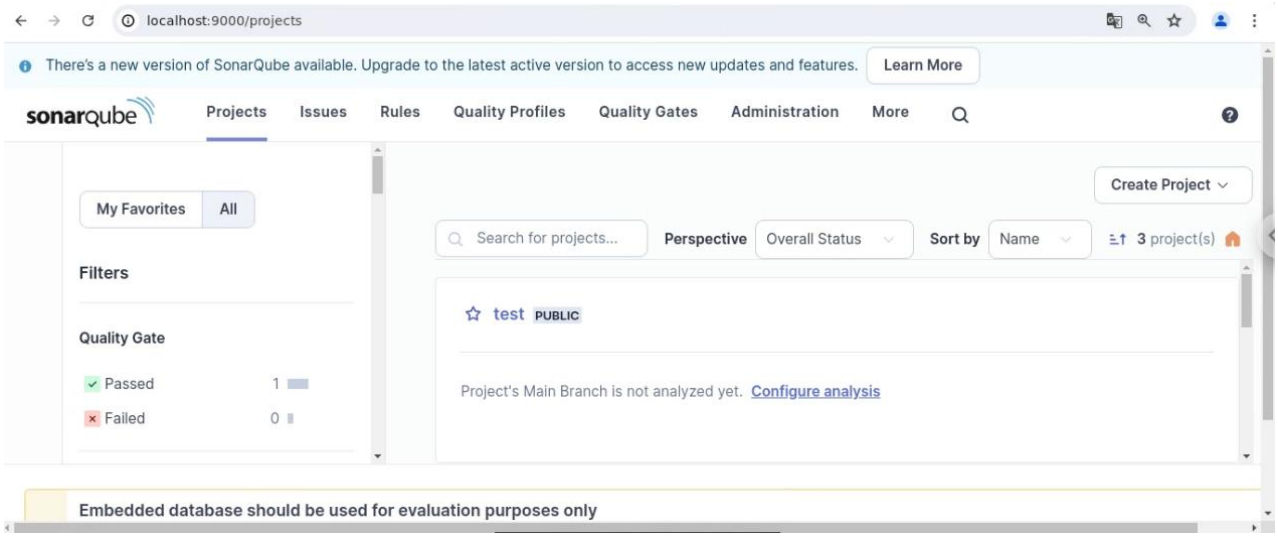


Nota. Versión de SonarQube instalada en la interfaz web. Elaborado por: Los autores

Se obtiene una pantalla de inicio como se muestra en la **Figura 26.**

Figura 26

Ventana inicial SonarQube



Nota. Se muestra menú principal de SonarQube. Elaborado por: Los autores.

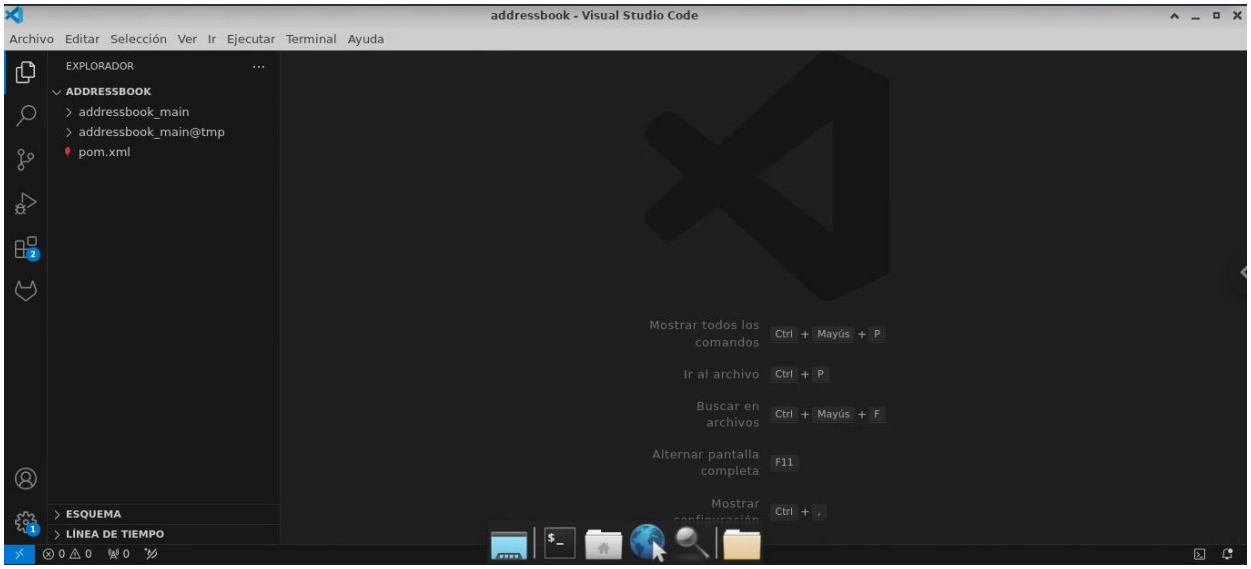
Visual Studio Code

Como IDE editor de código abierto se instaló VScode como se muestra en la **Figura 27**.

Para su configuración se inició y revisó el estado del servicio de Jenkins, se instalaron los plugins necesarios para la integración con las demás herramientas como SonarQube y GitLab.

Figura 27

Ventana Visual Studio Code



Nota. Ventana principal Visual Studio Code editor de código. Elaborado por: Los autores

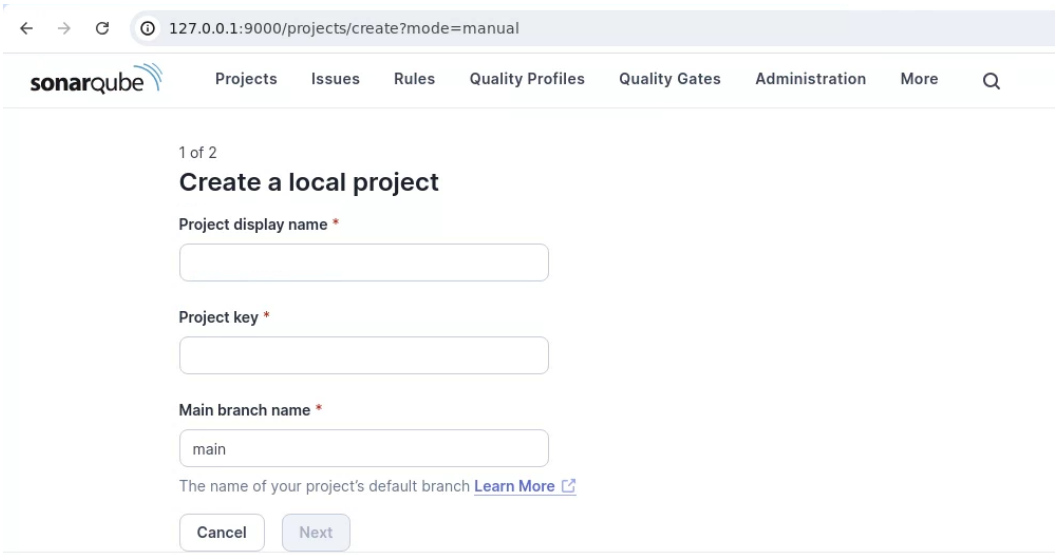
4.3.2. Configuración de las herramientas escogidas para el ambiente automatizado

Configuración de SonarQube

Se creó un nuevo proyecto en la herramienta, se le asignó un nombre y se le proporciona las configuraciones globales como se muestra en la **Figura 28.**

Figura 28

Creación del proyecto SonarQube



Nota. Se muestra la creación de un nuevo proyecto en la herramienta SonarQube. Elaborado por: Los autores

Una vez creado el proyecto se configuró la herramienta mediante la cual se procede a integrar el análisis de código, en este caso Jenkins que ejecuta la tarea desde la construcción de su pipeline para que el código sea analizado por SonarQube.

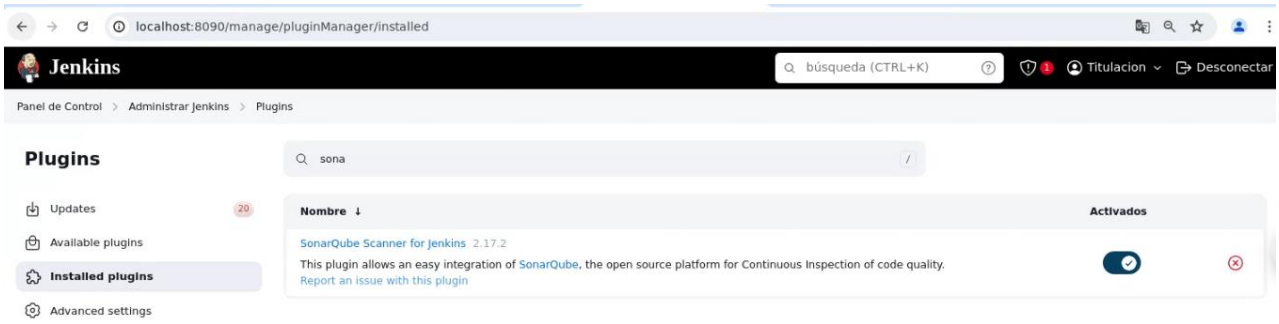
Para que se pueda integrar Jenkins con SonarQube se genera un token desde la configuración de seguridad del perfil de la herramienta donde se coloca un nombre, se escoge el tipo como token de usuario y un tiempo de expiración de 30 días. El token generado se copiará posteriormente en la configuración del sistema de Jenkins.

Configuración de Jenkins

Para la configuración e integración de las herramientas instaladas dentro del servidor Jenkins, se instaló el plugin fundamental para la integración, por lo cual se seleccionó SonarQube Scanner for Jenkins como se muestra en la **Figura 29**.

Figura 29

Descarga de plugin SonarQube Scanner



Nota. Se muestra la ventana donde se realiza la instalación de SonarQube Scanner para Jenkins. Elaborado por: Los autores

Una vez instalados los plugins necesarios se procede con la configuración del sistema para el servidor de SonarQube, se habilita la variable de entorno y se coloca un nombre, la dirección de la URL de SonarQube, y el token de usuario generado anteriormente. Una vez integrada dicha herramienta se creó un nuevo trabajo, pipeline.

Dentro de nuestra nueva Pipeline se ingresa los diferentes stages que serán ejecutados al momento de enviar a construir el pipeline donde se encontrara:

- La descarga del proyecto
- La instalación de las librerías
- Compilación del código
- Test
- Calidad de código
- Empaquetado del proyecto
- Preparación del ambiente de despliegue
- Despliegue del proyecto

- Pruebas de carga

Cada stage ejecuta una tarea distinta dentro del flujo del pipeline

Integración

Jenkins

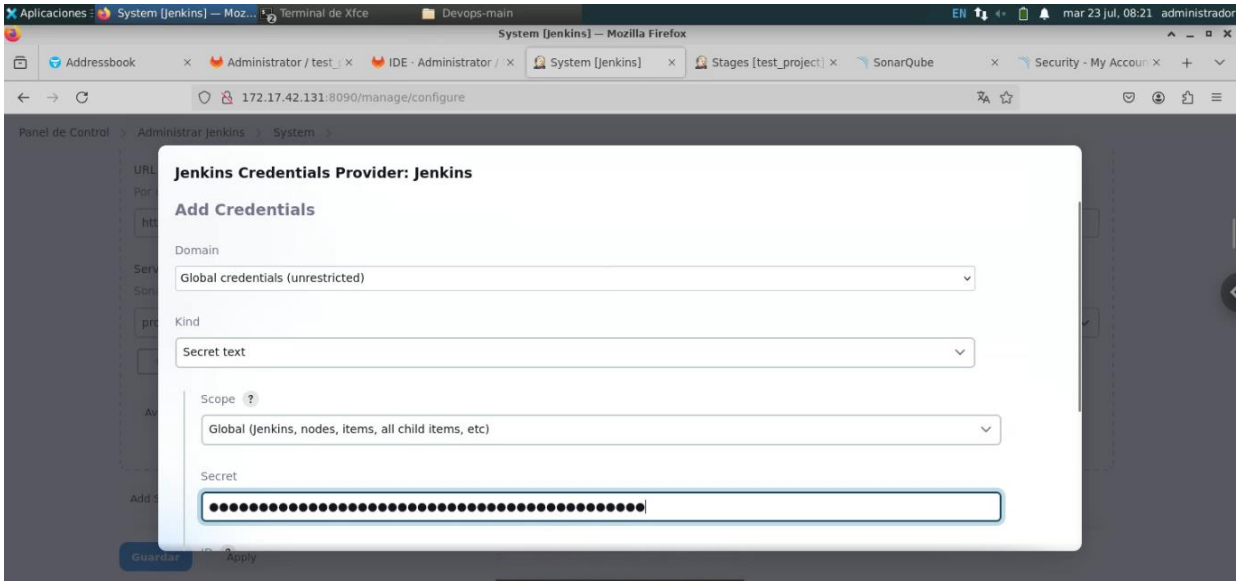
Para la integración con SonarQube se instala un plugin llamado SonarQube Scanner, dentro de la configuración del sistema de Jenkins se habilita las variables de entorno del servidor de SonarQube y se coloca un nombre, la URL y se agrega un nuevo token de autenticación al servidor con el texto secreto proporcionado por la misma herramienta de análisis de código.

A continuación, se crea un nuevo ítem en Jenkins con un pipeline y un proyecto nuevo en SonarQube con las configuraciones globales.

Antes de configurar en SonarQube, se instala en la herramienta de Jenkins el respectivo plugin de SonarQube Scanner, y en la configuración del sistema de Jenkins, se habilita las variables de entorno de SonarQube donde se configuró la URL del entorno de SonarQube, adicional se creó un token en SonarQube.

Figura 30

Credenciales de Jenkins



Nota. Configuración token para conexión a SonarQube Elaborado por: Los autores

Configuración del pipeline

Para la configuración del pipeline se inició con la creación de variables donde se guardaron las rutas y los datos más importantes que vamos a utilizar durante todo el desarrollo como la versión del servidor de aplicaciones y librerías utilizadas dentro de este proyecto, rutas de los directorios donde se encuentran instalados los mismos y ruta de los archivos utilizados por JMeter como se muestra en la **Figura 31**.

Figura 31

Creación de variables para rutas

```
pipeline {
    agent any

    environment {
        TOMCAT_VERSION = '8.5.24'
        MAVEN_VERSION = '3.9.4'
        MAVEN_DIR = "/var/lib/jenkins/apache-maven-${MAVEN_VERSION}"
        JMETER_DIR = '/var/lib/jenkins/workspace/apache-jmeter-5.6.3/bin'
        JMETER_TEST_FILE = "${JMETER_DIR}/Jmeter_test.jmx"
        JMETER_RESULTS_FILE = "${JMETER_DIR}/resultados.jtl"
        JMETER_REPORT_DIR=
'/var/lib/jenkins/workspace/test_project/results/report'
        COMMIT_PORT = '8081'
        MERGE_PORT = '9091'
    }
```

Nota. Configuración de variables y rutas que se va a utilizar en el desarrollo. Elaborado por: Los autores

A continuación, en la **Figura 32**, se procedió con la creación del primer stage el cual se encarga de clonar o actualizar el código del repositorio y realizar ciertas acciones relacionadas con el control de versiones.

Figura 32

Stage checkout code

```
stages {

        stage('CHECKOUT CODE') {
                steps {
                checkout scmGit(branches: [[name: '*/main']], extensions: [],
userRemoteConfigs: [[url: 'http://localhost:81/root/test_project.git']])
                 echo env.GIT_BRANCH
                }
        }
```

Nota. Configuración del stage para extraer el código del repositorio de la rama main. Elaborado por: Los autores

Esta parte el stage se encarga de asegurar que Apache Maven esté instalado en el entorno de Jenkins como se muestra en la **Figura 33**. En caso de que Maven no esté presente lo descarga, descomprime, mueve e instala, asegurando que Jenkins pueda utilizarlo para la construcción del proyecto. Si Maven ya está instalado, la etapa simplemente imprime un mensaje indicando que no es necesario realizar ninguna acción.

Figura 33

Stage de instalación de Maven

```
stage('Instalar Herramienta de Construcción Maven') {
            steps {
                script {
                    def mavenInstalled = fileExists("${MAVEN_DIR}/bin/mvn")
|| sh(script: 'which mvn', returnStatus: true) == 0

                    if (!mavenInstalled) {
                        echo 'Maven no está instalado. Procediendo con la
instalación...'
                        sh 'wget https://dlcdn.apache.org/maven/maven-
3/3.9.4/binaries/apache-maven-3.9.4-bin.tar.gz'
                        sh 'tar -xzvf apache-maven-3.9.4-bin.tar.gz'
                        sh "sudo mv apache-maven-${MAVEN_VERSION}
${MAVEN_DIR}"
                        sh "sudo chown -R jenkins:jenkins ${MAVEN_DIR}"
                    } else {
                        echo 'Maven ya está instalado.'
                    }
                }
            }
        }
```

Nota. Este stage verifica que Maven este instalado y configurado correctamente. Elaborado por: Los autores

A continuación, el siguiente stage se encarga de compilar el código fuente del proyecto ubicado en el directorio addressbook/addressbook_main. Primero cambia el directorio de trabajo a la ubicación del proyecto, luego ejecuta el comando Maven para compilar el código fuente del proyecto. Si la compilación falla, se imprime un mensaje de error en la consola de

Jenkins, lo que facilita la identificación y solución de problemas de compilación como se muestra en la **Figura 34**.

Figura 34

Stage para compilar la aplicación

```
stage('Compilar Aplicación de Ejemplo') {
            steps {
                dir('addressbook/addressbook_main') {
                    sh "${MAVEN_DIR}/bin/mvn compile"
                }
            }
            post {
                failure {
                    echo 'Error en la compilación de la aplicación de ejemplo'
                }
            }
        }
```

Nota. Configuración del stage para compilar el código fuente del directorio seleccionado. Elaborado por: Los autores

El stage de test que se muestra en la **Figura 35** se encarga de ejecutar las pruebas unitarias del proyecto ubicado en el directorio addressbook/addressbook_main. Primero, cambia el directorio de trabajo al directorio del proyecto, luego ejecuta el comando Maven para correr las pruebas unitarias. Si alguna prueba falla, se imprime un mensaje de error en la consola de Jenkins para ayudar a diagnosticar problemas en las pruebas.

Figura 35

Stage de pruebas de la aplicación

```
  stage('Pruebas de la Aplicación de Ejemplo') {
            steps {
                dir('addressbook/addressbook_main') {
                    sh "${MAVEN_DIR}/bin/mvn test"
                }
            }
            post {
                failure {
                    echo 'Error en las pruebas unitarias de la aplicación
de ejemplo'
                }
            }
        }
```

Nota. Stage encargado de ejecutar pruebas unitarias del directorio. Elaborado por: Los autores

En la **Figura 36** se muestra el stage que está diseñado para ejecutar un análisis de calidad del código utilizando SonarQube. Dentro de esta etapa, el pipeline:Cambia al directorio donde está ubicado el código fuente del proyecto. Configura el entorno para usar SonarQube con las credenciales y la configuración especificadas. Ejecuta un comando Maven que limpia, verifica y realiza el análisis de SonarQube del proyecto, enviando los resultados al servidor de SonarQube.

Figura 36

Stage de análisis con SonarQube

```
stage('Análisis con SonarQube') {

            steps {
                dir('addressbook/addressbook_main') {
                    withSonarQubeEnv(credentialsId: 'sonar_tokenN',
installationName: 'sonar') {
                        sh "${MAVEN_DIR}/bin/mvn clean verify sonar:sonar
-Dsonar.projectKey=test_projectN -Dsonar.projectName='test_projectN'"
                    }
                }
            }
        }
```

Nota. Configuración del stage para el análisis de código utilizando SonarQube. Elaborado por: Los autores

El siguiente stage que se muestra en **Figura 37** se encarga de crear un archivo empaquetado a partir del código fuente del proyecto, este proceso cambia al directorio del

proyecto donde está ubicado el archivo pom.xml y el código fuente y ejecuta el comando Maven package, que compila el código, ejecuta las pruebas y empaqueta el proyecto en un archivo distribuible WAR. Este archivo empaquetado es el resultado final de la construcción del proyecto.

Figura 37

Stage para empaquetar la aplicación

```
stage('Empaquetar Aplicación de Ejemplo') {
            steps {
                dir('addressbook/addressbook_main') {
                    sh "${MAVEN_DIR}/bin/mvn package"
                }
            }
        }
```

Nota. Configuración del stage para crear un archivo empaquetado del proyecto. Elaborado por: Los autores

A continuación, en la **Figura 38** se muestra el stage que se encarga de preparar el entorno para la ejecución de Tomcat instalando los componentes necesarios. En este caso, se actualiza la lista de paquetes disponibles y se instala el entorno de ejecución de Java 17 (openjdk-17-jre), que es un requisito para que Tomcat funcione, ya que Tomcat es una aplicación basada en Java y necesita una JVM (Java Virtual Machine) para ejecutarse.

Figura 38

Stage de instalación de prerrequisitos de Tomcat

```
stage('Instalación de Prerrequisitos de Tomcat') {
            steps {
                sh 'sudo apt update'
                sh 'sudo apt install openjdk-17-jre -y'
            }
        }
```

Nota. Instalación de componentes para la ejecución de Tomcat. Elaborado por: Los autores

La etapa Install Apache Tomcat Server de la **Figura 39** está diseñada para configurar el servidor Apache Tomcat en función del contexto del build (commit o merge). Dependiendo del tipo de evento, si es un commit (env.GIT_COMMIT está definido), configura Tomcat cambiando los permisos, copiando archivos de configuración, y ajustando el puerto del servidor, de lo contrario es un merge (cuando env.GIT_BRANCH es main o master), realiza configuraciones similares, pero con un puerto diferente.

Figura 39

Stage de instalación de Apache Tomcat Server

```
stage('Install Apache Tomcat Server') {
            steps {
                script {
                    echo env.GIT_COMMIT
                    if (env.GIT_COMMIT) {
                        // Ejecución en caso de commit
                        sh 'sudo chown -R jenkins:jenkins apache-tomcat-
8.5.24'
                        sh 'sudo cp tomcat-users.xml apache-tomcat-
8.5.24/conf/tomcat-users.xml'
                        sh 'sudo cp context.xml apache-tomcat-
8.5.24/webapps/manager/META-INF/context.xml'
                        sh "sudo sed -i 's/8080/${env.COMMIT_PORT}/g'
apache-tomcat-8.5.24/conf/server.xml"
                    } else if (env.GIT_BRANCH == 'main' || env.GIT_BRANCH
== 'master') {
                        // Ejecución en caso de merge
                        sh 'sudo chown -R jenkins:jenkins apache-tomcat-
8.5.24'
                        sh 'sudo cp tomcat-users.xml apache-tomcat-
8.5.24/conf/tomcat-users.xml'
                        sh 'sudo cp context.xml apache-tomcat-
8.5.24/webapps/manager/META-INF/context.xml'
                        sh "sudo sed -i 's/9090/${env.MERGE_PORT}/g'
apache-tomcat-8.5.24/conf/server.xml"
                    }
                }
            }
        }
```

Nota. Configuración del servidor Apache Tomcat. Elaborado por: Los autores

En la **Figura 40** el stage está diseñado para desplegar la aplicación web en el servidor Tomcat y verificar que el despliegue haya sido exitoso.

Figura 40

Stage de despliegue de la aplicación en el servidor Tomcat

```
stage('Deploy Sample Application To Tomcat Server') {
            steps {
                sh 'sudo cp
addressbook/addressbook_main/target/addressbook.war apache-tomcat-
8.5.24/webapps/'
                sh 'sudo runuser -l jenkins -c
"/var/lib/jenkins/workspace/test_project/apache-tomcat-
8.5.24/bin/startup.sh"'
            }
        }

        stage('Verificar Despliegue') {
            steps {
                script {
                    def appUrl = 'http://localhost:8081/addressbook'
                    def response = sh(script: "curl -s -o /dev/null -w
'%{http_code}' ${appUrl}", returnStdout: true).trim()
                    if (response != '200' && response != '302') {
                        error "La aplicación no se está ejecutando
correctamente. Código de respuesta HTTP: ${response}"
                    } else {
                        echo 'La aplicación se está ejecutando
correctamente.'
                    }
                }
            }
        }
```

Nota. Configuración para el despliegue de la aplicación web. Elaborado por: Los autores

A continuación, el stage de la **Figura 41** verifica la existencia del directorio de JMeter y el archivo de prueba, construye el comando para ejecutar JMeter, ejecuta las pruebas de rendimiento, captura la salida del comando y finalmente imprime los resultados de la ejecución.

Figura 41

Stage de ejecución de pruebas de JMeter

```
        stage('Run JMeter Tests') {
            steps {
                script {
                    def jmeterDir = env.JMETER_DIR
                    def jmeterTestFile = env.JMETER_TEST_FILE
                    def jmeterResultsFile = env.JMETER_RESULTS_FILE

                    if (!fileExists(jmeterDir)) {
                        error "El directorio de JMeter (${jmeterDir}) no
existe."
                    }

                    if (!fileExists(jmeterTestFile)) {
                        error "El archivo de prueba de JMeter
(${jmeterTestFile}) no existe."
                    }

                    def jmeterCommand = "${jmeterDir}/jmeter -n -t
${jmeterTestFile} -l ${jmeterResultsFile}"
                    def jmeterOutput = sh(script: jmeterCommand,
returnStdout: true).trim()

                    println "Resultado de la ejecución de
JMeter:\n${jmeterOutput}"
                }
            }
        }
```

Nota. Configuración del stage para ejecutar pruebas de rendimiento utilizando JMeter. Elaborado por: Los autores

4.4.Pruebas del ambiente configurado

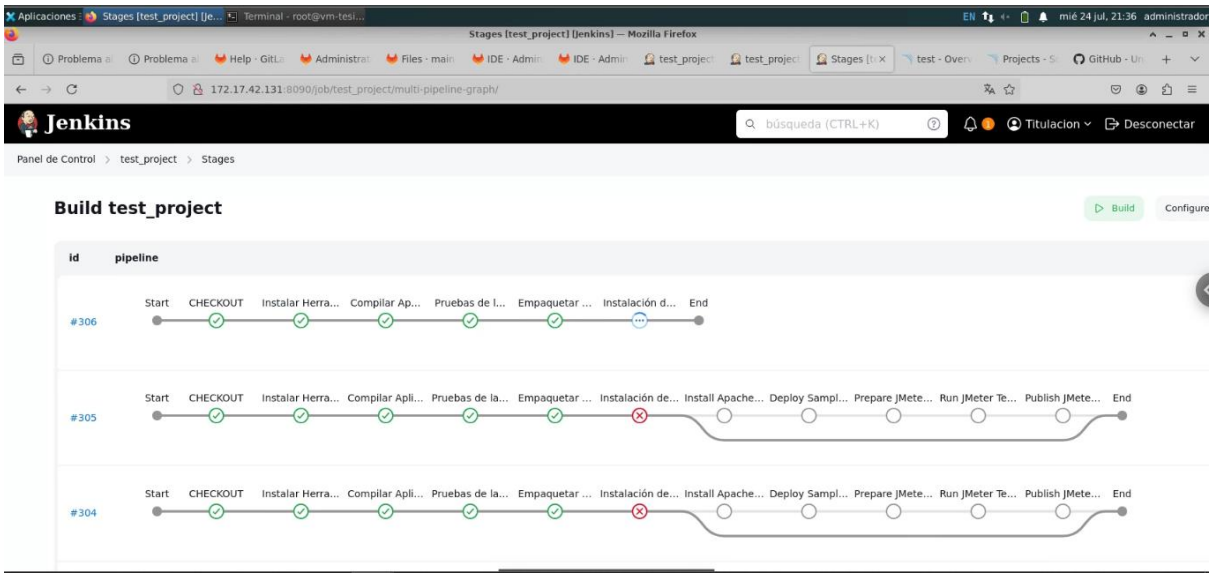
Caso funcional

4.4.1. Resultados de la construcción del pipeline en Jenkins

Una vez ejecutados todos los stages del pipeline se obtiene como resultado la correcta ejecución de cada una de las fases desde la descarga y construcción del proyecto hasta el despliegue como se puede ver en la **Figura 42.**

Figura 42

Construcción del proyecto



Nota. Despliegue correcto del proyecto en Jenkins. Elaborado por: Los autores

4.4.2. Resultados SonarQube

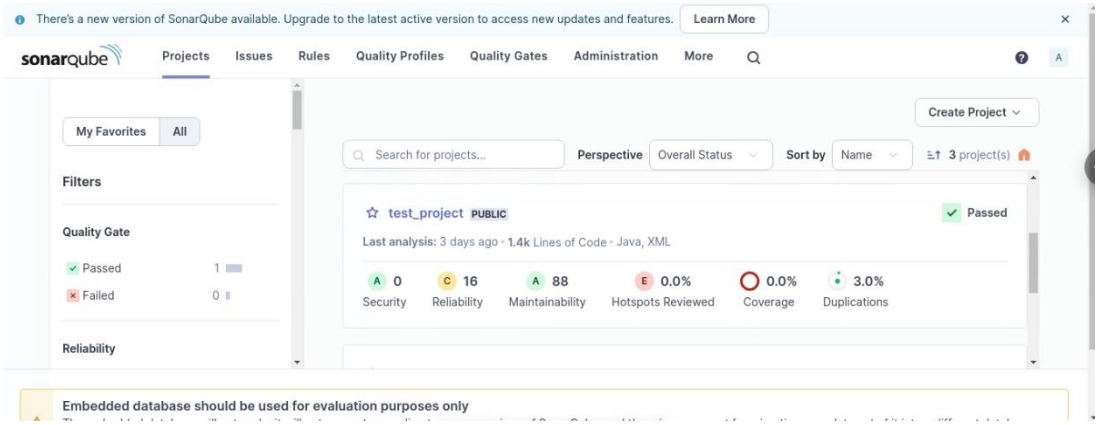
Las etapas que se visualiza en esta herramienta al momento de analizar la calidad son:

- **Security**: que analiza el código desde el punto de vista de seguridad con respecto a vulnerabilidades de seguridad.
- **Reliability:** Este parámetro mide la estabilidad del código al momento de ejecutarse, fallas y excepciones.
- **Maintainability:** Esta variable indica que tan difícil es mantener y evaluar el código para posterior a esto.
- **Hotspots Reviewed:** Esto indica las áreas críticas dentro del código fuente.
- **Coverage:** Muestra la calidad de código.
- **Duplications:** Indica el porcentaje de código duplicado en el proyecto, lo cual puede ser un indicador de código que necesita ser remplazado para mejorar la disponibilidad y reducir fallos.

A continuación, se muestra en la **Figura 43** los resultados de SonarQube

Figura 43

Resultados SonarQube



Nota. Resultados de SonarQube. Elaborado por: Los autores

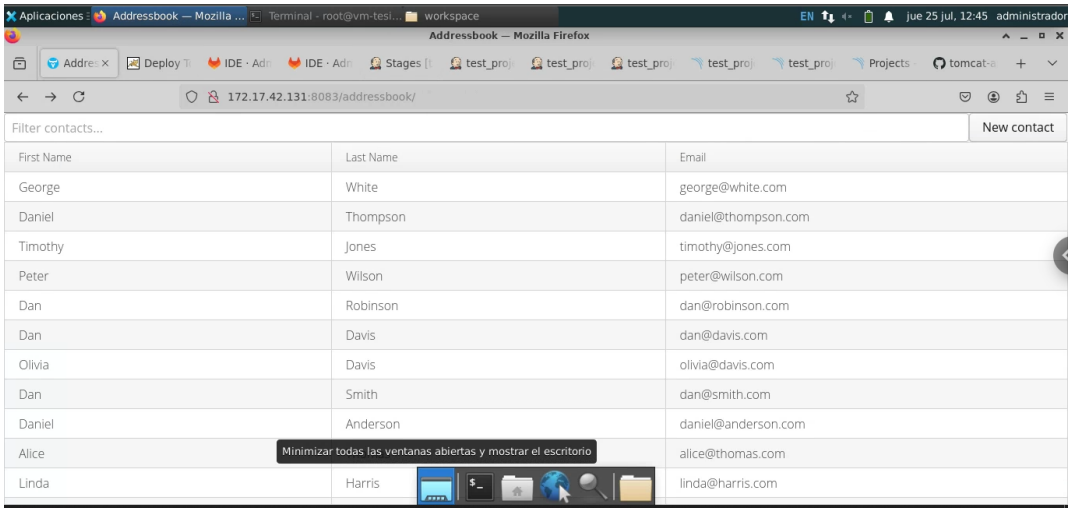
4.4.3. Resultados del despliegue

Por último, se obtuvo el resultado final del despliegue realizado por Tomcat el cual se encuentra desplegado localmente en el puerto 8083 como se muestra en la **Figura 44.**

En la dirección http://localhost:8083/addressbook/

Figura 44

Ventana del proyecto desplegado



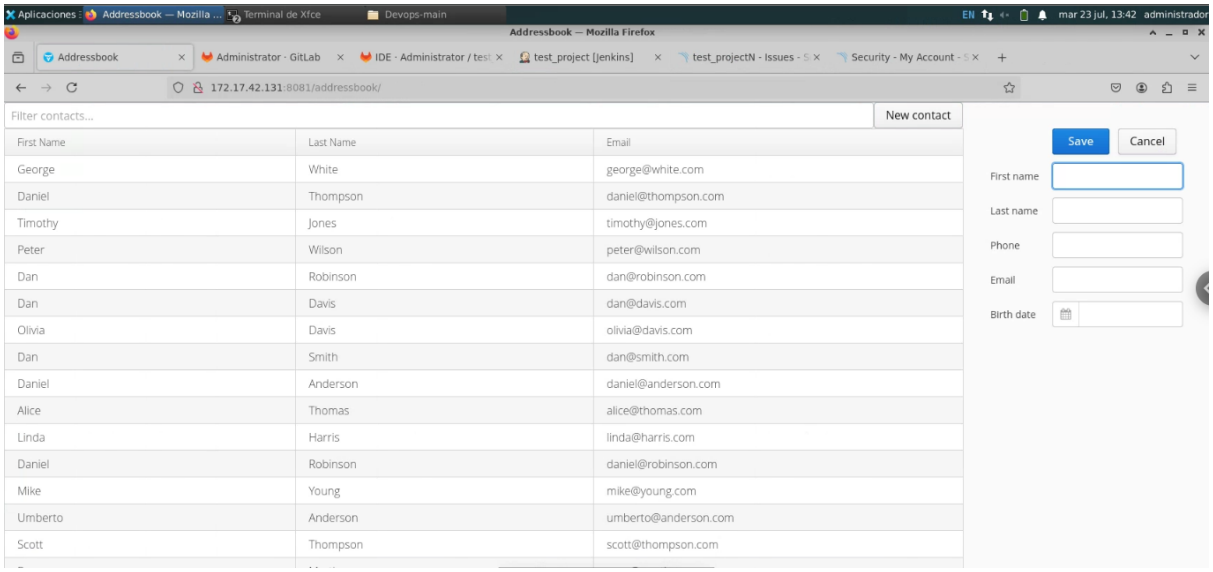
Nota. Ventana del proyecto. Elaborado por: Los autores

Cambio a realizar

En la ventana principal de la aplicación desplegada se muestra una de las funciones para agregar un nuevo contacto dentro de la lista de contactos, como se visualiza en la **Figura 45** se muestra que la ventana emergente para añadir los datos se encuentra a lado derecho.

Figura 45

Ventana de la aplicación desplegada



Nota. Aplicación principal con la ventana emergente del lado derecho. Elaborado por: Los autores

A continuación, se realizará un cambio para que la ventana emergente se muestre a lado izquierdo.

Una vez realizado el cambio dentro del código realizamos un commit para que se registre el cambio y se actualice el código para que nos muestre un mensaje informando que los cambios realizados fueron exitosos como se muestra en la **Figura 46** ya que al realizar un cambio el pipeline se dispara automáticamente al realizarse el commit.

Figura 46

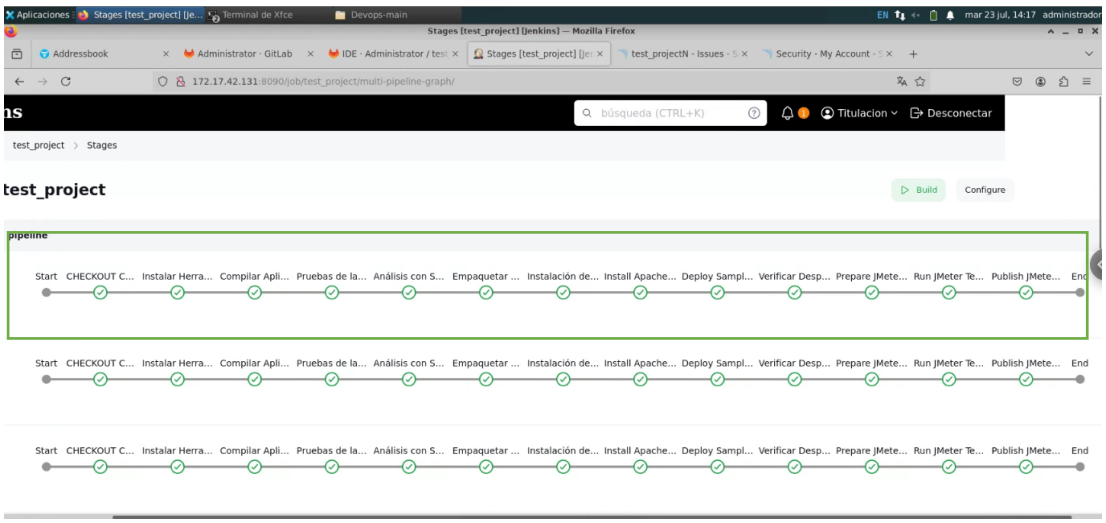
Ventana de ejecución del pipeline

Nota. Ejecución de cada stage del pipeline en proceso. Elaborado por: Los autores

A continuación, se visualiza que se ejecutó el pipeline exitosamente como se muestra en la **Figura 47**, el pipeline termino su ejecución pasando por cada uno de los stages.

Figura 47

Ventana del Jenkins con la ejecución del pipeline



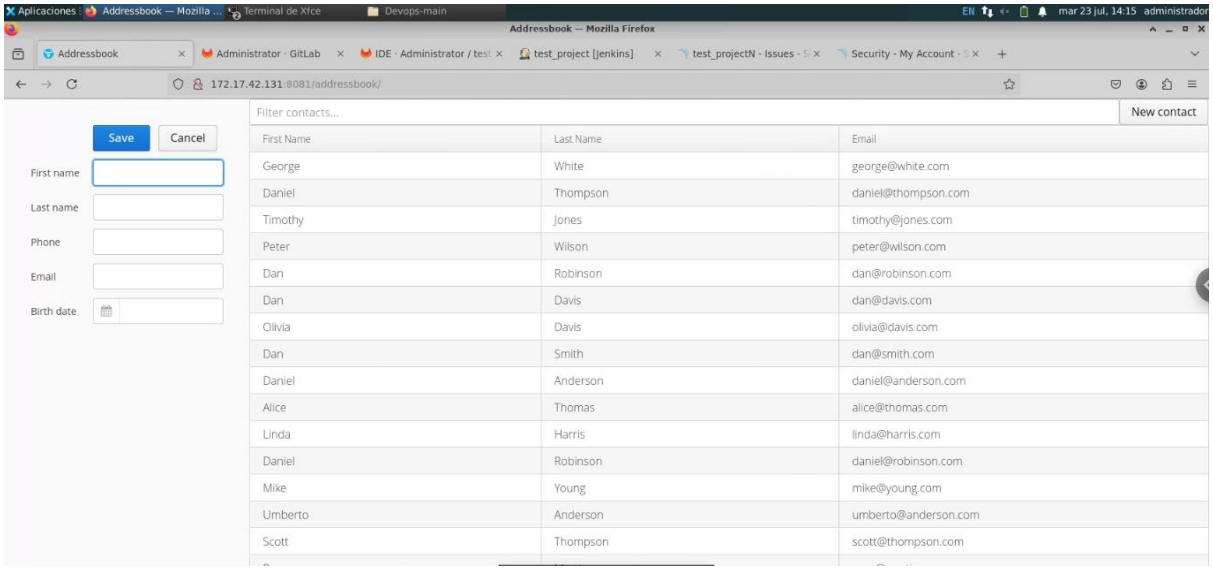
Nota. Ejecución exitosa de cada etapa del pipeline dentro de Jenkins. Elaborado por: Los autores

Cambio realizado

Como resultado se obtiene el cambio realizado y se muestra en la **Figura 48** la ventana emergente a lado izquierdo la cual se realizó el cambio desde el editor de código como se comentó anteriormente.

Figura 48

Ventana del aplicativo con el cambio realizado

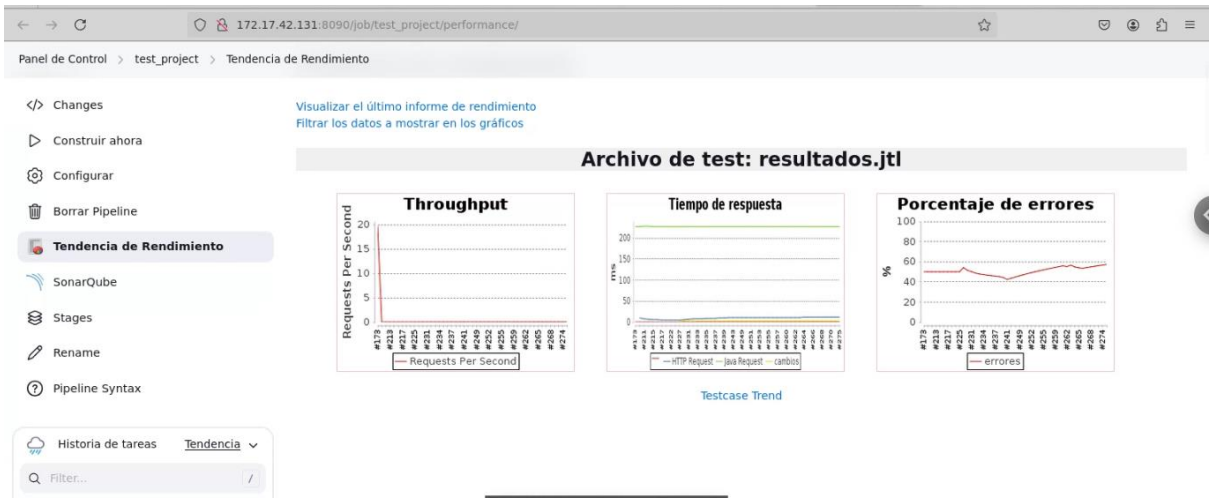


Nota. Aplicación principal con la ventana emergente del lado izquierdo. Elaborado por: Los autores

La **Figura 49** muestra los resultados de la prueba de rendimiento del JMeter, representados en tres gráficos:

Figura 49

Ventana de resultados de JMeter



Nota. Visualización del informe de rendimiento. Elaborado por: Los autores

Throughput (Rendimiento): Muestra la cantidad de solicitudes procesadas por segundo. En este caso se observa una fluctuación con picos y valles, la tendencia indica variabilidad en el rendimiento del sistema.

Tiempo de respuesta: Muestra el tiempo que tarda el sistema en responder a las solicitudes. En este caso HTTP Request, Java Request y Cambios, el tiempo de respuesta se mantiene bastante bajo y constante para todos los tipos de solicitudes, lo que indica un buen desempeño en términos de latencia.

Porcentaje de errores: Muestra el porcentaje de errores en las solicitudes. Se observa un aumento en el porcentaje de errores es decir que el sistema está experimentando problemas que necesitan ser corregidos.

Caso de prueba

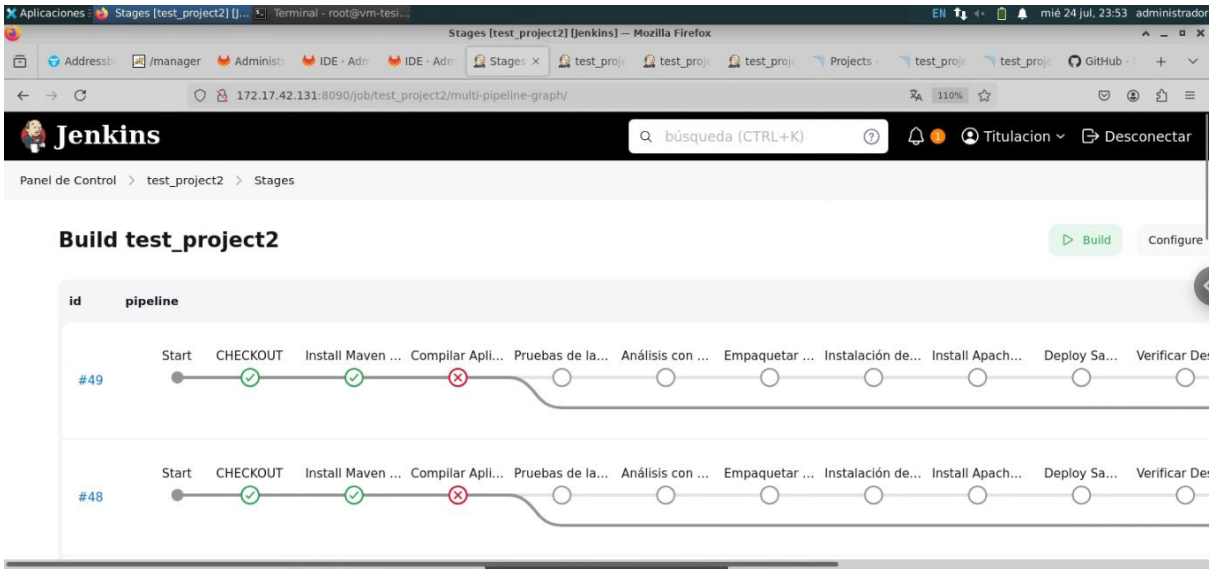
Se preparó un nuevo despliegue en el ambiente configurado, para lo cual se eligió un nuevo proyecto en Java y se procedió a realizar las pruebas verificando la compatibilidad de nuevo proyecto con el ambiente.

Al momento de intentar probar y ejecutar el pipeline con otro proyecto del repositorio se obtuvo un error de compilación.

En la **Figura 50** se muestra la ejecución del pipeline con la ejecución de la etapa de compilación de la aplicación fallida, la etapa checkout y la instalación del Maven se completó con éxito y en la etapa de compilación de la aplicación falló por lo tanto las etapas posteriores no se lograron ejecutar, este error se presentó debido a un error de sintaxis que se encontró en el stage de compilación, principalmente en el nombre del directorio donde se ejecuta esta fase lo cual hace que el pipeline se detenga y finalice con el proceso.

Figura 50

Ventana del stage de compilación fallido



Nota. Visualización de cada etapa compilándose. Elaborado por: Los autores

Es importante mencionar que se debe colocar correctamente el nombre de los directorios, variables, y sintaxis del pipeline para que al momento de ejecutar no se detenga.

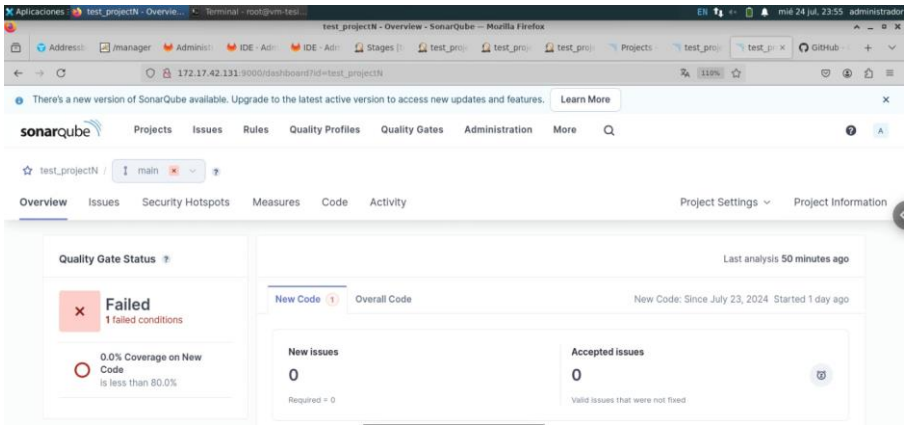
Dentro de las pruebas que se ejecutaron, encontramos que el pipeline se detuvo por un error en el despliegue de la aplicación, esto debido a que se el puerto por el que se estaba

comunicando el servidor de aplicaciones Tomcat ya se encontraba en uso del primer despliegue antes mostrado.

A continuación, como se muestra en la **Figura 51** se muestra la notificación de SonarQube indicando que hubo fallas al momento de la ejecución.

Figura 51

Ventana de notificación de SonarQube

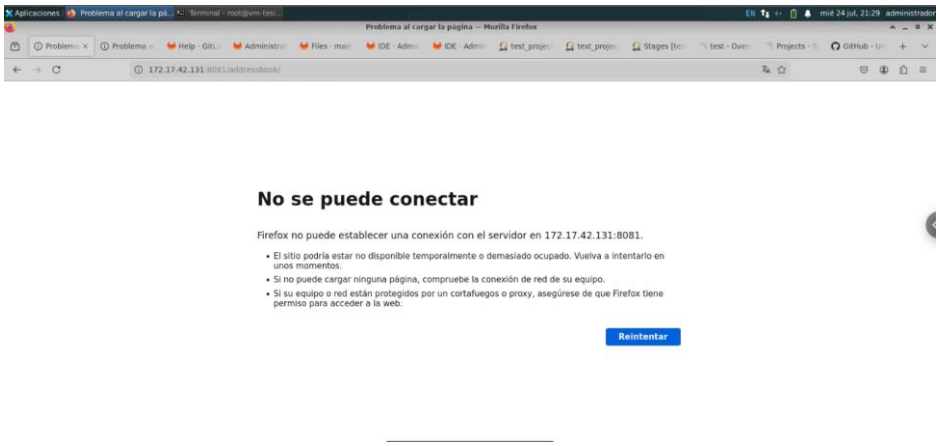


Nota. Visualización de la falla en SonarQube. Elaborado por: Los autores

A continuación, se muestra en la **Figura 52** que el aplicativo no se desplegó.

Figura 52

Ventana del navegador con un mensaje de error

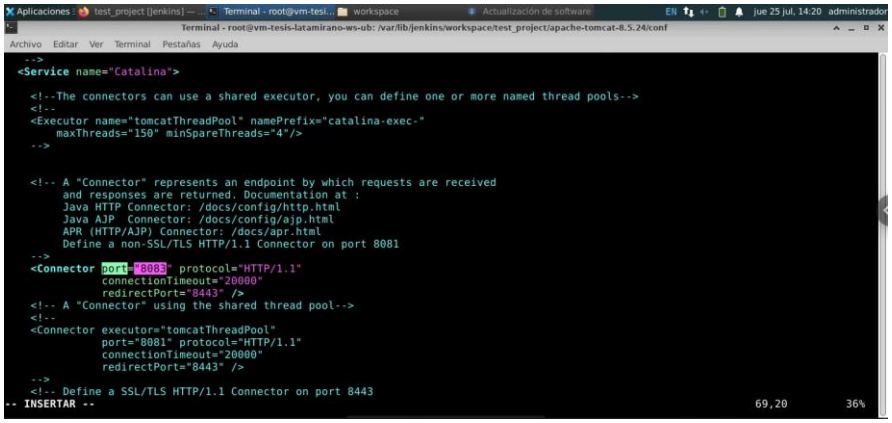


Nota. Ventana de error de conexión. Elaborado por: Los autores

Para este caso se realizó el cambio del puerto por el cual ApacheTomcat se comunicaba y se cambió el puerto desde el archivo de configuración del servidor de aplicaciones y se colocó el puerto 8083 como se muestra en la **Figura 53.**

Figura 53

Ventana de terminal con el archivo de configuración de Apache Tomcat.

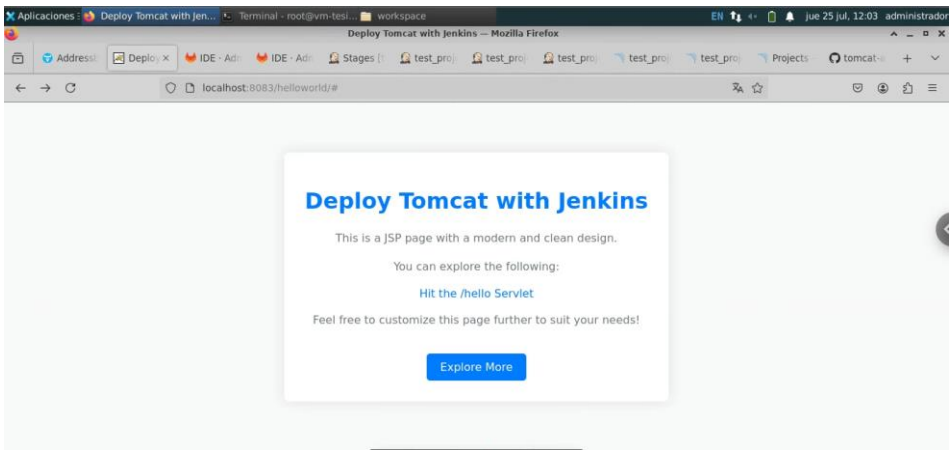


Nota. Tomcat está configurado para escuchar en el puerto 8083. Elaborado por: Los autores

Finalmente, como se muestra en la **Figura 54** se muestra el nuevo proyecto desplegado en el puerto 8083.

Figura 54

Página web desplegado Tomcat utilizando Jenkins.

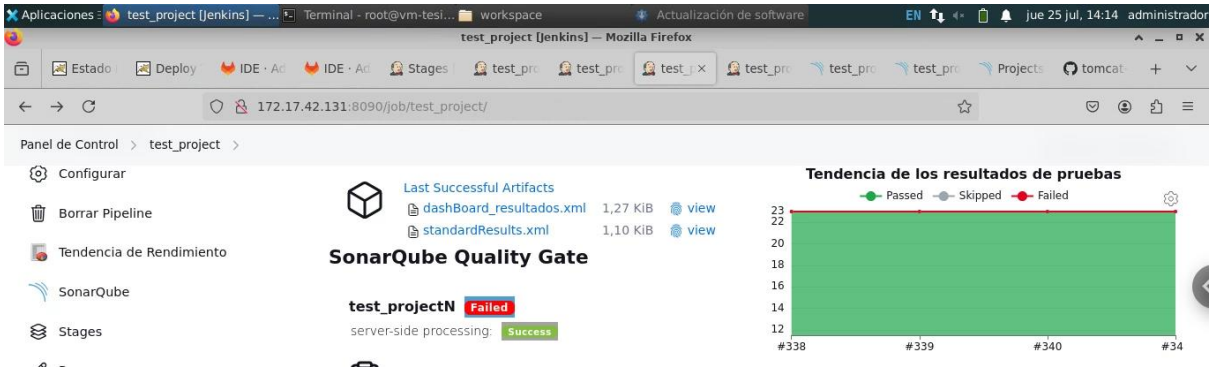


Nota. Ventana del proyecto desplegado. Elaborado por: Los autores

A continuación, se realizó una prueba, para validar que el pipeline se detiene cuando el proyecto llamado addresbook no se despliega correctamente, para lo cual se realizó un cambio en el código intencionalmente, como se muestra en la **Figura 55**, Jenkins muestra una falla debido a la inconsistencia en el código.

Figura 55

Interfaz web de Jenkins



Nota. Ventana de Jenkins con el análisis de SonarQube Elaborado por: Los autores

CONCLUSIONES

La investigación y el análisis respectivo ha permitido una comprensión más específica del funcionamiento de cada herramienta, destacando en cada una de ellas la integración con las demás para poder crear un flujo automatizado.

Se logró configurar un ambiente IC/DC que automatiza procesos de despliegue de proyectos de software, la integración de GitLab, Jenkins y SonarQube ha permitido un flujo de trabajo el cual gestiona el control del código fuente, la ejecución de pruebas continuas y la evaluación de la calidad del código.

En las pruebas realizadas sobre este ambiente se comprueba que la integración de estas herramientas fue compatible ya que al momento de su configuración estas se acoplaron correctamente y para comprobar se cargó un programa para que pase por este flujo y que al finalizar se obtenga buenos resultados.

RECOMENDACIONES

Escoger un IDE que sea compatible con las demás herramientas y lenguajes de programación usados en el ambiente que se requiera en el proyecto.

Gitlab es una de las mejores opciones para proyectos que requiera una solución robusta y que sea gratuita para la gestión de código, Jenkins es una herramienta flexible para automatizar procesos como en este caso IC/DC y SonarQube que es una herramienta de las más completas para la revisión continua de código.

Asegurarse de que las herramientas escogidas sean integrables entre ellas para que más adelante no exista problemas de configuración y flujo.

Documentar cada proceso y configuración que se vaya utilizando en el entorno automatizado para que en caso de que se requiera realizar cambios no dañar el ambiente.

Para futuros trabajos es recomendable incorporar escaneo de vulnerabilidades y pruebas de seguridad automatizadas ya que esto no solo mejoraría la calidad de código, sino que también prepararía a los estudiantes para abordar los crecientes temas de ciberseguridad en el desarrollo de software moderno.

REFERENCIAS

Amendola, M. (19 de agosto de 2022). *openwebinars*. Top herramientas DevOps: Integración y Despliegue Continuo: https://openwebinars.net/blog/top-herramientas-devops-integracion-y-despliegue-continuo/

aplyca. (07 de 12 de 2022). Integración continua y entrega continua (CI/CD): https://www.aplyca.com/blog/integracion-continua-y-entrega-continua-cicd

atlassian. (2024). Automatización de la implementación: qué es y cómo empezar: https://www.atlassian.com/es/devops/frameworks/deployment-automation

inesdi. (28 de 06 de 2022). ¿Qué es Jenkins y por qué es clave en todo pipeline?: https://www.inesdi.com/blog/que-es-jenkins-por-que-es-clave/

redhat. (11 de mayo de 2022). La integración y la distribución continuas (CI/CD): https://www.redhat.com/es/topics/devops/what-is-ci-cd

Rolfo, M. (5 de septiembre de 2022). *codigoencasa*. Integraciones continuas ¿Qué son y para que sirven?: https://codigoencasa.com/integraciones-continuas-que-son-y-para-que-sirven/

SENTRIO. (16 de 09 de 2021). Introducción a Jenkins: ¿qué es, para qué sirve y cómo funciona?: https://sentrio.io/blog/que-es-jenkins/

tecdelainfosite. (2 de Agosto de 2017). Evolución del Software: https://tecdelainfosite.wordpress.com/2017/08/02/evolucion-del-software/

tibco. (2024). ¿Qué es el despliegue continuo?: https://www.tibco.com/es/reference-center/what-is-continuous-deployment

Torres, A. (12 de diciembre de 2021). *excentia*. Qué es Jira y para qué sirve: https://www.excentia.es/que-es-jira-para-que-sirve-funcionalidades

unity. (2024). Explicación de CI/CD: https://unity.com/es/solutions/what-ci-cd#:~:text=La%20CI%2FCD%2C%20o%20integraci%C3%B3n,posible%20gracias%20a%20la%20automatizaci%C3%B3n.

Vega, M. (2020 de julio de 8). *ideasatcloud*. ¿Qué es DevOps?: https://ideasatcloud.azurewebsites.net/que-es-devops/

viewnext. (20 de marzo de 2019). Integracion Continua, Entrega Continua y Despliegue continuo: https://www.viewnext.com/integracion-continua/

Atlassian. (2024). Flujo de trabajo de Gitflow: https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow#:~:text=%C2%BFQu%C3%A9%20es%20Gitflow%3F,vez%20y%20quien%20lo%20popularizó.

aws.amazon. (2023). ¿Qué es un entorno de desarrollo integrado (IDE)?: https://aws.amazon.com/es/what-is/ide/#:~:text=Un%20entorno%20de%20desarrollo%20integrado%20(IDE)%20es%2

0una%20aplicaci%C3%B3n%20de,una%20aplicaci%C3%B3n%20f%C3%A1cil%20de%20usar.

b2core. (s.f.). ¿Qué es un entorno de staging?: https://b2core.com/es/news/what-is-a-staging-environment/#:~:text=Los%20entornos%20de%20staging%20sirven,software%20antes%20de%20su%20lanzamiento.

Borges, V. (4 de agosto de 2023). *community*. Cómo crear un pipeline con Azure DevOps y Spring Boot: https://community.listopro.com/como-crear-un-pipeline-con-azure-devops-y-spring-boot/

Chacón, C. (2024). *ucenfotec*. Integración y Despliegue Continuo: https://ucenfotec.ac.cr/ecommerce/producto/integracion-y-despliegue-continuo/?v=3fd6b696867d

Durán, M. (24 de agosto de 2023). *blog.hubspot*. Qué es CI/CD y por qué es importante: https://blog.hubspot.es/website/que-es-ci-cd

Flores, F. (22 de julio de 2022). *openwebinars*. Qué es Visual Studio Code y qué ventajas ofrece: https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/

ibm. (s.f.).¿Qué es el despliegue continuo?: https://www.ibm.com/es-es/topics/continuous-deployment

ilimit. (10 de diciembre de 2020). Integración continua, entrega continua y despliegue continuo: https://ilimit.com/blog/integracion-continua-entrega-continua-despliegue-continuo/

Javier. (29 de enero de 2024). *Formadores(IT)*. ¿Qué es Gitlab y para qué sirve?: https://formadoresit.es/que-es-gitlab-y-para-que-sirve/

Manjaly, S. (24 de marzo de 2023). *invgate*. Qué es Ansible: la herramienta DevOps para automatizar tareas de IT: https://blog.invgate.com/es/ansible

MAX, R. (2024). *atlassian*. ¿En qué consiste la integración continua?: https://www.atlassian.com/es/continuous-delivery/continuous-integration

Primo, D. (2022). *webreactiva*. ¿Qué es una Pull Request?: https://www.webreactiva.com/podcast/haz-la-primera-pull-request-de-tu-vida

REHKOPF, M. (202). *atlassian*. ¿En qué consiste la integración continua?

SENTRIO. (15 de 12 de 2021). Qué es SonarQube: Verifica y analiza la calidad de tu código: https://sentrio.io/blog/que-es-sonarqube/

sydle. (09 de 06 de 2022). Integración de sistemas: conoce su importancia, sus tipos y sus retos: https://www.sydle.com/es/blog/integracion-de-sistemas-6140d39a84679b13bf127a93

unir.net. (25 de 06 de 2021). Ventajas del uso de Docker en el diseño y gestión de aplicaciones: https://www.unir.net/ingenieria/revista/funciones-docker/

Walsh, D. (14 de agosto de 2023). *blog.hubspot*. GitHub vs. GitLab: diferencias, similitudes y usos: https://blog.hubspot.es/website/github-gitlab#:~:text=GitHub%20destaca%20por%20su%20popularidad,para%20la%20gesti%C3%B3n%20de%20proyectos.

Walther. (4 de septiembre de 2023). *dongee*. Qué es Pipeline: Una Guía Completa para Principiantes: https://www.dongee.com/tutoriales/que-es-pipeline-una-guia-completa-para-principiantes/

Zúñiga, F. G. (10 de 05 de 2017). *arsys*. Artifactory, un repositorio universal para facilitar el trabajo de los equipos de desarrollo: https://www.arsys.es/blog/artifactory-cloud