UNIVERSIDAD PERUANA UNIÓN

FACULTAD DE INGENIERÍA Y ARQUITECTURA

Escuela Profesional de Ingeniería de Sistemas

Implementación de ambientes de integración y despliegue continuo (CI/CD) para aplicaciones empresariales en el centro de datos de la Universidad Peruana Unión Filial Juliaca

Tesis para obtenertener el Título profesional de Ingeniero de Sistemas

Por:

Bach. Bonnier Nilss, Mamani Larico

Asesor:

Ing. Angel Rosendo, Condori Coaquira

Juliaca, Diciembre del 2020

DECLARACIÓN JURADA DE AUTORÍA DEL INFORME DE TESIS

Angel Rosendo Condori Coaquira, de la Facultad de Ingeniería y Arquitectura, Escuela Profesional de Ingeniería de Sistemas, de la Universidad Peruana Unión.

DECLARO:

Que el presente informe de investigación titulado: "**IMPLEMENTACIÓN DE AMBIENTES DE INTEGRACIÓN Y DESPLIEGUE CONTINUO (CI/CD) PARA APLICACIONES EMPRESARIALES EN EL CENTRO DE DATOS DE LA UNIVERSIDAD PERUANA UNIÓN FILIAL JULIACA**" constituye la memoria que presenta el Bachiller **Bonnier Nilss Mamani Larico** para obtener el título de Profesional de Ingeniero de Sistemas, cuya tesis ha sido realizada en la Universidad Peruana Unión bajo mi dirección.

Las opiniones y declaraciones en este informe son de entera responsabilidad del autor, sin comprometer a la institución.

Y estando de acuerdo, firmo la presente declaración en Juliaca, a los 24 días del mes de agosto del año 2021

Ing. Angel Rosendo Condori Coaquira
Asesor

ACTA DE SUSTENTACIÓN DE TESIS

En Puno, Juliaca, Villa Chullunquiani, a 28 día(s) del mes de diciembre del año 2020 siendo las 7:30 horas, se reunieron en el Salón de Grados y Títulos de la Universidad Peruana Unión, Filial Juliaca, bajo la dirección del Señor Presidente del jurado: Mtro. Lennin Henry Centurión Julca, el secretario: Mg. Esbel Angel Sullon Macalupu y los demás miembros: Ing. Jorge Eddy Otazú Tuque y Ing. Eder Gutiérrez Quispe y el asesor Ing. Angel Rosendo Gondori Coaguira con el propósito de administrar el acto académico de sustentación de la tesis titulada: Implementación de ambientes de integración y despliegue continuo (CI/CD) para aplicaciones empresariales en el centro de datos de la Universidad Peruana Unión Filial Juliaca de el(los)/la(las) bachiller/es: a) Bonnier Nilss Mamani Larico b) conducente a la obtención del título profesional de Ingeniero de Sistemas

(Nombre del Titulo Profesional)

con mención en

El Presidente inició el acto académico de sustentación invitando al (los)/a(la)(las) candidato(a)/s hacer uso del tiempo determinado para su exposición. Concluida la exposición, el Presidente invitó a los demás miembros del jurado a efectuar las preguntas, y aclaraciones pertinentes, las cuales fueron absueltas por el(los)/la(las) candidato(a)/s. Luego, se produjo un receso para las deliberaciones y la emisión del dictamen del jurado.

Posteriormente, el jurado procedió a dejar constancia escrita sobre la evaluación en la presente acta, con el dictamen siguiente:

Candidato (a): Bonnier Nilss Mamani Larico

CALIFICACIÓN	ESCALAS			Mérito
	Vigesimal	Literal	Cualitativa	
Aprobado	15	B-	Bueno	Muy Bueno

Candidato (b):

CALIFICACIÓN	ESCALAS			Mérito
	Vigesimal	Literal	Cualitativa	

(*) Ver parte posterior

Finalmente, el Presidente del jurado invitó al(los)/a(la)(las) candidato(a)/s a ponerse de pie, para recibir la evaluación final y concluir el acto académico de sustentación procediéndose a registrar las firmas respectivas.

Presidente — Secretario

Asesor — Miembro — Miembro

Candidato/a (a) — Candidato/a (b)

DEDICATORIA

El presente trabajo investigativo está dedicado principalmente a Dios, por ser el inspirador y darnos fuerza para continuar en este proceso de obtener uno de los anhelos más deseados. A mis padres, por su amor, trabajo y sacrificio en todos estos años, gracias a ellos hemos logrado llegar hasta aquí y convertirnos en lo que somos. Ha mi esposa y a mi hijo por el apoyo brindado por estar siempre presentes, y por el apoyo moral, que me brindaron a lo largo de esta etapa. A todas las personas que nos han apoyado y han hecho que el trabajo se realice con éxito en especial a aquellos que nos abrieron las puertas y compartieron sus conocimientos.

AGRADECIMIENTOS

Agradecemos a Dios por bendecirnos la vida, por guiarnos a lo largo de nuestra existencia, ser el apoyo y fortaleza en aquellos momentos de dificultad y de debilidad. Gracias a mi padre Raul Mamani Machaca, a mi madre Zoraida Larico Pomari, a mi esposa Vilma Vargas Nina, a mi hijo Bonnier Nilss Rodrigo Mamani Vargas por ser los principales promotores de mis sueños, por confiar y creer en mis expectativas, por los consejos, valores y principios que me han inculcado. Agradecemos a nuestros docentes de la Escuela Profesional de Ingeniería de Sistemas de la Universidad Peruana Unión Campus Juliaca, por haber compartido sus conocimientos a lo largo de la preparación de mi profesión, de manera especial, a mi asesor Angel Rosendo Condori Coaquira quien me ha guiado con su paciencia, y su rectitud como docente.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ANEXOS

SÍMBOLOS USADOS

- UPeU: Universidad Peruana Unión
- DTI: Dirección de Tecnologías de Información
- S.O: Sistema operativo
- FTP: Protocolo de transferencia de archivos
- CI: Integración continua
- CD: Distribución continua
- BD: Base de datos
- IDE: Entorno de desarrollo integrado
- API: Interfaz de programación de aplicaciones
- GUI: Interfaz gráfica de usuario
- CLI: Interfaz de línea de comandos
- TI: Tecnologias de información
- SSL: Capa de conexión segura
- KPI: Key Performance Indicator (Indicador clave de performance)

RESUMEN

En el presente trabajo de investigación se analiza e implementa un modelo orientado a la administración de aplicaciones empresariales en el data center de la UPeU Campus Juliaca, es en esta institución educativa existen numerosas aplicaciones en distintas tecnologías y/o lenguajes que no se encuentran estandarizados y que tienen distintos carriles de instalación, además lo realizan diferentes personas de acuerdo a un conocimiento específico. Esto produce atrasos en puestas en producción, riesgos en las aplicaciones y un alto costo humano al no tener procesos automatizados que puedan ayudar a compilar y realizar las pruebas unitarias. Dicho lo anterior se propone utilizar el método de CI/CD para la optimización de los procesos y el enfoque de ingeniería, entrega continua que trata en crear pipeline o carriles de instalación desde los ambientes de desarrollo hasta los ambientes productivos por cada aplicación, en el cual se definen escenarios y trabajos para realizar procesos automáticos, informando al desarrollador si el componente se encuentra correcto o tiene errores mientras avanza en cada uno de los carriles. Para validar la metodología se utilizó el ciclo Deming de mejora continua junto con un indicador de eficiencia de despliegue con el antes y el después de la solución propuesta y el indicador de calidad para ver la calidad del código al realizar un push. Se implemento el CI/CD se definio un ambiente de integración y despliegue continúo basado en la cultura DevOps, se concluye diciendo que logra ver la mejora realizando una prueba T-Student que muestra una variable T por encima del valor critico en cuanto al despliegue de aplicaciones, también usando el ciclo Deming de mejora continua el cual nos ayudó a resolver problemas y estar siempre en un aprendizaje de forma continua.

Palabras clave: Integración continua, Distribución continua, Deming, DevOps, Docker, Jenkins, Gitlab.

ABSTRACT

In this research work, a model oriented to the administration of business applications in the data center of the UPeU Campus Juliaca is analyzed and implemented, it is in this educational institution there are numerous applications in different technologies and / or languages that are not standardized and that have different installation rails, also performed by different people according to specific knowledge. This results in delays in start-ups, application risks and a high human cost by not having automated processes that can help compile and perform unit tests. Having said the above, it is proposed to use the CI / CD method for the optimization of processes and the engineering approach, continuous delivery that deals with creating pipeline or installation lanes from development environments to production environments for each application, in the which defines scenarios and jobs to carry out automatic processes, informing the developer if the component is correct or has errors as it progresses in each of the lanes. To validate the methodology, the Deming cycle of continuous improvement was used together with a deployment efficiency indicator with the before and after of the proposed solution and the quality indicator to see the quality of the code when performing a push. The CI / CD was implemented, an environment of integration and continuous deployment based on the DevOps culture was defined, it is concluded by saying that it manages to see the improvement by performing a T-Student test that shows a variable T above the critical value in terms of the deployment of applications, also using the Deming cycle of continuous improvement which helped us solve problems and always be in a continuous learning.

Keywords: Continuous Integration, Continuous Distribution, Deming, DevOps, Docker, Jenkins, Gitlab.

CAPÍTULO I. El problema

1.1. Identificación del problema.

Según (Díaz & Muñoz, 2018) el entorno de los centros de datos a evolucionado en todo el mundo donde solo en México se tiene el 20% de centro de datos en todo el mundo. Estos centros de datos tienen necesidades específicas al ser los que albergan los proyectos considerados “Misión Crítica” para las organizaciones.

Uno de los grandes problemas que presenta la industria del software es que a pesar de que hay estándares, metodologías, técnicas, lineamientos y demás herramientas, éstas no se emplean de manera generalizada, haciendo de esta industria una producción de artesanías. Bajo este esquema la industria no puede predecir la culminación de los proyectos, ni afianzar el proceso de producción, tampoco se puede estimar la calidad del producto entregar. Con estos inconvenientes se está impulsando la adopción de estándares y herramientas que ayuden en la producción de software (Zavala Ruiz, 2015).

El despliegue de aplicaciones con máquinas virtuales consume, una gran cantidad de recursos solo para la administración de sistema operativo, esto hace que se sea poco eficiente con el control de recursos. Tener un centro de datos sin un esquema de despliegue definido implica una mala administración de los recursos informáticos tales como procesamiento, ram, tiempo de despliegue y la entrega oportuna del producto al cliente.

El centro de datos de la Universidad Peruana Unión Filial Juliaca cuenta con la infraestructura, tecnología y personal calificado para la construcción de aplicaciones empresariales, a esto se pueden agregar metodologías para el proceso de integración continua y la fase de pruebas que garanticen el mínimo error en el pase a producción. Aparte el centro de datos invierte demasiados recursos en el despliegue de las aplicaciones con el uso de instancias de VMWare que consumen muchos recursos en la administración de sistemas operativos tal como lo podemos ver en el Anexo A y Anexo B. Para esto proponemos Desarrollar un entorno de construcción de aplicaciones que realice y el seguimiento, complimiento de las faces en el ciclo de vida del proyecto de Software.

1.2. Justificación

La presente investigación se realiza para poder mejorar el área de centro de datos de la Universidad Peruana Unión filial Juliaca con la implementación de ambientes de desarrollo de aplicaciones el cual pueda ser administrado en las distintas fases del proceso de desarrollo del software.

La adopción de una cultura de "Desarrollo y Operaciones" (Development and Operations) DevOps contribuye no solo para el desarrollo de aplicaciones de software más rápidas, flexibles y ágiles, sino que mantiene comunicados a los departamentos de las organizaciones. Hablar de DevOps es hacer énfasis en la automatización de procesos y mantener una comunicación directa con el área de operaciones con el cual se cumplira con los objetivos de la empresa con la mayor precisión.

La aplicación de una arquitectura de micro servicios no solo contribuye con estos conceptos, sino que facilita las buenas prácticas. Con el respectivo despliegue en contenedores se reducen los niveles de riesgos en producción, se pueden aislar los fallos y no afectar a todas las aplicaciones. Con este enfoque orientado a contenedores se puede eliminar la mayoría de los problemas que surgen al tener configuraciones de entorno incoherentes y los problemas que se derivan de ellos.

Económicamente la elaboración de esta investigación con el uso de software libre desarrollará un ambiente libre de pagos y licencias que será administrada por el área de centro de datos a diferencia de otras plataformas como TeamCity, Microsoft Azure que cuenten con licencias para el paso a producción y uso de sus herramientas.

A nivel del área de Desarrollo se logrará tener un ambiente de pruebas automatizadas logrando tener un producto con mayor calidad y un mínimo de error en sus entregables.

1.3. Presunción filosófica

A lo largo de nuestra preparación académica, adquirimos diferentes aptitudes en los 5 años en las salas de estudios universitarios, donde simultáneamente obtuvimos varios capacidades y habilidades en el avance de nuestra vocación de Ingeniería de Sistemas. La preparación que obtenemos en nuestra institución universitaria no solo se basa en formación académica, sin embargo, también nos preparamos con estándares y valores cristianos y en

esta exploración construimos un modelo de ambiente de integración y despliegue continúo (CI/CD) para aplicaciones empresariales, mostramos las aptitudes escolares que conseguimos durante toda nuestra preparación universitaria

Salmos 1:1-3 dice: “Bienaventurado el varón que no anduvo en consejo de malos, Ni estuvo en camino de pecadores, Ni en silla de escarnecedores se ha sentado; 2 Sino que en la ley de Jehová está su delicia, Y en su ley medita de día y de noche. 3 será como árbol plantado junto a corrientes de aguas, Que da su fruto en su tiempo, Y su hoja no cae; Y todo lo que hace, prosperará.”

1.4. Objetivos

- **Objetivo general.**

  Desarrollar un ambiente de construcción de aplicaciones empresariales del centro de datos de la Universidad Peruana Unión Filial Juliaca.

- **Objetivos específicos.**
  - ✓ Analizar y diseñar el ambiente de desarrollo.
  - ✓ Seleccionar las herramientas adecuadas para mejorar el ambiente de desarrollo.
  - ✓ Establecer una nueva arquitectura de despliegue de servicios.
  - ✓ Definir un ambiente de integración y despliegue continúo basado en la cultura DevOps.
  - ✓ Monitorear el despliegue de aplicaciones en Developement y Master.

CAPÍTULO II. Revisión de la literatura

2.1. Antecedentes de la investigación.

En la investigación de licenciatura en Ingeniero en Informática realizado por Farias (2017) con el título de estudio, Definición de un ambiente de construcción de aplicaciones empresariales a través de Devops, Microservicios y Contenedores. Desarrolló la automatización de procesos de integración continua de microservicios con el cual obtuvo mejores resultados en la organización al emplear ambientes de integración. Durante el desarrollo de la investigación se utilizó la metodología SCRUM. En esta sección se estudiarán sus principales características y fases de implementación, los resultados muestran la gran efectividad al usar Devops, se logra implementar código con una frecuencia 30 veces mayores.

Díaz, Oswaldo Muñoz, Mirna (Mexico 2018) en su artículo Implementación de un enfoque DevOps más la administración de riesgos muestran las mejores prácticas al memento de desplegar una infraestructura DevOps, así mismo muestran los problemas de seguridad que conllevan los contenedores y como solucionarlos.

Olmedo, Paula Beatriz Noel, Fernanda Moyano, Pucheta (Argentina 2017) en su trabajo el rol de Docker muestra como los contenedores mejoran el performance y consumo de recursos en el despliegue de aplicaciones a si también muestra la facilidad de integración con múltiples lenguajes y servidores.

Clínica americana (Juliaca 2018) a partir del lanzamiento de aplicaciones de gran escala y conforme a su plan de mejora en la administración de servidores, comienza a desarrollar todas sus aplicaciones en contendores Docker, esto con el fin de hacer un mantenimiento de librerías en desuso y desacoplar el mantenimiento del sistema a la del servidor.

GitLab (EEUU 2016) uno de los más grandes repositorios en el mundo realizar su más ambicioso proyecto convirtiendo su plataforma en un administrador de aplicaciones DevOps para la comunidad, realizando a si un salto hacia una nueva forma en la administración de servidores.

2.2. Marco teórico.

2.2.1. Arquitectura de Software.

Una arquitectura de software se entiende o se ve como la estructura de un sistema, que contiene componentes, sus propiedades vistas a otros componentes y las relaciones entre ellos. “La arquitectura comprende los conceptos y atributos esenciales en el ámbito de un sistema, representados en cada una de sus partes, vínculos y en los fundamentos que determinan su diseño y avance.” (Standard, 2011)

La arquitectura de software es de muy importante ya que la manera en que la que se tiene que estructurar un sistema tiene que tener un impacto directo sobre la capacidad de este para satisfacer lo que normalmente conocemos como los atributos de calidad del sistema. Podemos decir que algunos ejemplos de atributos de calidad son el desempeño que esta tiene, el cual tiene que ver con el tiempo de respuesta del sistema a las peticiones que a esta se le hacen, la usabilidad, el cual tiene que ver en qué tan sencillo les resulta a los usuarios realizar operaciones el sistema, o también la modificabilidad, el cual tiene que ver en qué tan simple resulta realizar cambios en el sistema. Estos atributos de calidad son parte fundamental de los requerimientos no funcionales del sistema y son características que deben expresar de forma cuantitativa. (ISO/IEC/IEE 42010, 2011)

El punto de vista arquitectónico es un producto de trabajo el cual nos establece los convenios para la construcción, interpretación y uso de vistas para una arquitectura. Una vista expone la arquitectura de un sistema desde el punto de vista de los requerimientos específicos de la aplicación.

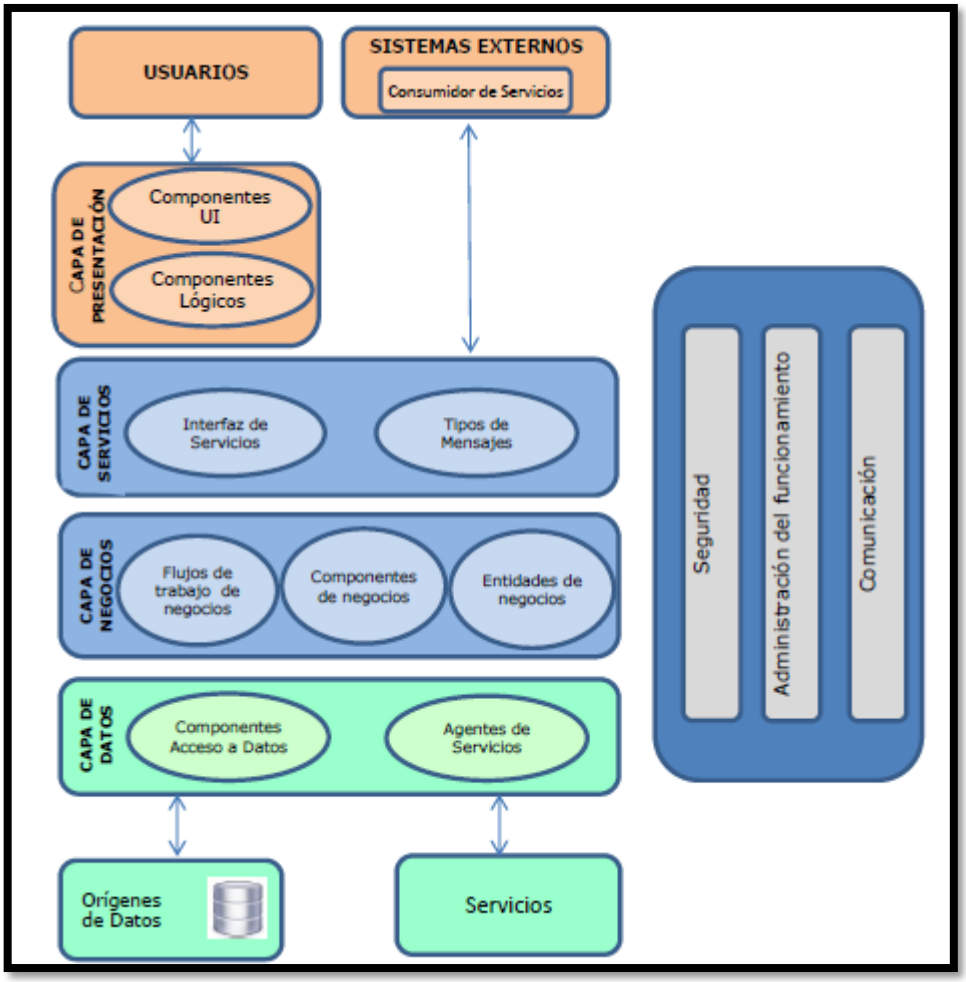

Figura 1. Descripción de Arquitectura de un estilo específico.
Fuente: (Microsoft Corporation, 2009)

2.2.1.1.El ciclo de desarrollo de la arquitectura.

Dentro de un proyecto, e independientemente de la metodología que se vaya a usar, se puede hablar de desarrollo de la arquitectura de software. Este desarrollo, el cual inicia con la construcción del sistema, se divide en las siguientes etapas: requerimientos, diseño, documentación y evaluación.

A continuación, describiremos dichas etapas.

Requerimientos. La etapa de requerimientos se centra en captar, informar y organizar las necesidades que alimentan el diseño, por lo que esta etapa las subraya. No obstante, distintos prerrequisitos son igualmente aplicables a la ingeniería, estas son las necesidades prácticas esenciales y los imperativos que conllevan. (Cervantes, 2017).

Diseño. La etapa de diseño es la etapa focal según la arquitectura y presumiblemente la más importante de hacer. En esta etapa se caracterizan las estructuras que componen la arquitectura. La fabricación de estas estructuras depende de los diseños de configuración, las estrategias de planificación y las decisiones mecánicas. El plan que se hace debe buscar lo más importante para cumplir con los prerrequisitos que impactan en la ingeniería, y no solo consolidar diferentes innovaciones ya que están de moda. (Cervantes, 2017).

Documentación. Una vez realizada la configuración de arquitectura, es importante tener la opción de impartirla a cada uno de los asociados al turno de eventos. La correspondencia eficaz del plan dependerá con frecuencia de que el plan se registre adecuadamente. La documentación de un diseño incluye la representación de algunas de sus estructuras a las que se habla a través de diversas perspectivas. Una vista normalmente contiene un gráfico, solo como datos adicionales, para ayudar a comprender el gráfico. (Cervantes, 2017).

Evaluación. Dado que la arquitectura del producto asume una parte básica en el ciclo de avance, es atractivo que el plan se evalúe siempre que se haya archivado para distinguir los posibles problemas y peligros. El beneficio de evaluar el plan es que es un movimiento que se realiza temprano, incluso antes de la codificación, y que el costo de enmendar las deformidades reconocidas a través de la evaluación es considerablemente menor que el costo que necesitaría para abordar estas imperfecciones una vez que la marco ha sido fabricado (Cervantes, 2017).

2.2.1.2.Estilos de arquitectura.

Cada estilo arquitectónico representa una clasificación del marco que contiene un grupo de segmentos, lo que representa una capacidad que necesita el marco, un grupo de conectores que potencian la correspondencia, coordinación y participación entre los segmentos, limitaciones que caracterizan cómo podemos incorporar los segmentos. Que componen el marco y modelos semánticos que permiten al creador comprender las propiedades mundiales de un marco para desglosar las propiedades conocidas de sus partes constituyentes (EcuRed, 2016).

Cabe resaltar que Gutierrez (2013), señala que cada estilo describe una clase de sistemas que comprende:

- Un conjunto de componentes, como BD y módulos computacionales, los cuales desempeñan una función solicitada por parte del sistema.
- Dichos componentes necesitan un conjunto de conectores, los cuales permitirán que exista comunicación, coherencia y colaboración entre ellos.
- Limitaciones que definen cómo se forman los componentes para crear el sistema.
- Modelos semánticos que permiten al diseñador comprender las características generales de un sistema, mediante el análisis de las características de los elementos que lo componen.

En la Tabla 1 citamos las principales áreas de enfoque y los estilos arquitectónicos.

Tabla 1. Estilo de arquitectura.

Categoría	Estilos de Arquitectura
Comunicación	Arquitectura orientada a servicios (SOA), Bus de mensajes
Despliegue	Cliente / Servidor, N-Tier, 3-Tier
Dominio	Diseño impulsado por dominio
Estructura	Arquitectura basada en componentes, orientada a objetos y en capas

Fuente: (Microsoft Corporation, 2009).

2.2.2. Integración continúa.

La integración continua es una práctica de mejora del producto mediante la cual los desarrolladores unen todas las progresiones que realizan al código, en una bóveda focal de manera consistente, después de lo cual se ejecutan los formularios y las pruebas programadas. La conciliación constante generalmente alude al período de creación o coordinación de la medida de distribución del producto e incluye un segmento de mecanización, por ejemplo, averiguar cómo incorporar con frecuencia. Los objetivos críticos

de la integración continua son descubrir y corregir errores más rápidamente, mejorar la calidad de la programación y reducir el tiempo que lleva aprobar y entregar nuevas actualizaciones de programación. (Olmedo & Moyano Pucheta, 2018).

Tabla 2. Tabla comparativa de herramientas de integración continúa.

	Entrega continua	Alojamiento en la nube	Licencia	Precio versión comercial	Versión gratuita	Particularidades
Jenkins	✔	✔	MIT	-	✔	Numerosos plugins
Travis CI	X	✔	MIT	69-489$/mes	✔	Conexión directa con Github
Banboo	✔	✔	De propietario	10-126500$(Pago único)	✔	
Gitlab CI	✔	✔	MIT/EE	4-99$/mes	✔	Conexión directa con otros productos de Atlassian
Circle CI	✔	✔	De propietario	50-3150$/mes	✔	Fácil de utilizar
Cruise Control	X	X	BSD	-	✔	Completamente gratuita
CodeShip	✔	✔	De propietario	75-1500$/mes	✔	Versión profesional y básica
TeamCity	✔	X	De propietario	299-21999€(Pago único)	✔	Gated Comits

Fuente: (Ionos, 2017)

2.2.3. DevOps.

El término DevOps, que es una mezcla de las expresiones en inglés development (desarrollo) y operations (operaciones), alude a la asociación de individuos, ciclos y tecnología para ofrecer un incentivo a los clientes de manera consistente (Azure, 2017).

Tabla 3. Funcionalidad de las herramientas de integración continúa.

Tecnología/funcionalidad	Github	Gitlab	Bitbucket	TFS	AzureDevOps
Revisión de código	Si	Si	Si	Si	Si
Incidencias	Si	Si	Si	Si	Si
Wiki	Si	Si	Si	Si	Si
Proyectos privados (Colaborativos)	De pago	Si	Si hasta 5 usuarios	Si	Si
Proyectos personales	Si	Si	Si	Si	Si
Integración continua	Con plugins	Si	Con otra herramienta	Si	Si
Monitorización	No	Si	No	No	Si
Open Source	No	Si	No	No	No
On-Premise	De pago	Si	De pago	Si	No
Cloud	Si	Si	Si	No	Si

Fuente: (Barredo, 2019)

2.2.3.1.DevOps para los equipos.

DevOps nos ayuda a que los roles recientemente aislados, se coordinen y trabajen juntos para crear productos mejores y más confiables. Al recibir una cultura DevOps combinada con dispositivos y prácticas DevOps, los grupos obtienen la capacidad de reaccionar más fácilmente a las necesidades de los clientes, aumentar la confianza en las aplicaciones que crean y lograr los objetivos comerciales en menos tiempo, (Azure, 2017).

2.2.3.2.Ventajas de usar DevOps.

Los grupos que comprenden la cultura, las prácticas y los instrumentos de DevOps mejoran la ejecución y producen mejores productos en menos tiempo, lo que aumenta la lealtad del consumidor. Esta mejora en la cooperación y la eficiencia es fundamental para lograr los objetivos comerciales, por ejemplo, estos:

- Reducción del tiempo de la comercialización
- Adaptación al mercado y la competencia
- Mantenimiento de la estabilidad y la confiabilidad del sistema
- Mejora del tiempo medio de la recuperación

2.2.3.3.Ciclo de vida de las aplicaciones y DevOps.

DevOps impacta el ciclo de vida de las aplicaciones a lo largo de las etapas de planeamiento, desarrollo, entrega y uso. Cada etapa depende de las demás y las etapas no son explícitas para un trabajo. En una cultura DevOps genuina, todos los trabajos están comprometidos con alguna ruta en todas las etapas, Figura 2, (Azure, 2017).

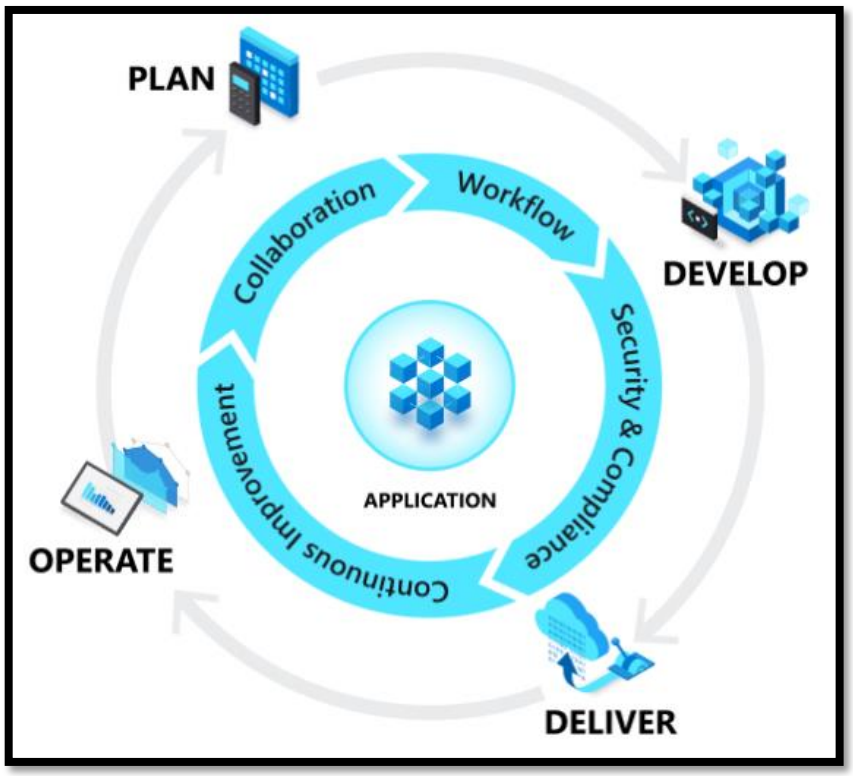

Figura 2. DevOps.
Fuente: (Azure, 2017)

2.2.3.3.1. Planear.

En la fase de planificación, los equipos de DevOps conciben, definen y describen las características y la funcionalidad de las aplicaciones y sistemas que crearán. Realizan un seguimiento del progreso tanto de forma amplia como detallada, desde tareas de un solo producto hasta tareas que abarcan carteras de varios productos. La creación de trabajos pendientes, el seguimiento de errores, la gestión del desarrollo de software ágil con Scrum, el uso de tableros Kanban y la visualización del progreso son algunas de las formas en que los equipos de DevOps planifican con agilidad y visibilidad (Azure, 2017).

2.2.3.3.2. Desarrollar.

La fase de desarrollo incorpora todas las partes de la programación (escritura, pruebas, revisión e integración del código por parte de los miembros del equipo) y compilar ese código para hardware de compilación que pueden enviar a numerosos entornos. Los grupos de DevOps buscan avanzar rápidamente sin renunciar a la calidad, solidez o eficiencia. Para hacer esto, utilizan herramientas sumamente productivas, automatizan los avances diarios y manuales, e iteran el código en pequeños aumentos utilizando pruebas automáticas e integración continua, (Azure, 2017).

2.2.3.3.3. Entregar.

La entrega es la forma de enviar solicitudes en condiciones de ejecución de manera constante y confiable. La etapa de entrega también incorpora el envío y la configuración de la base del centro completamente administrada que conforma esas condiciones, (Azure, 2017).

En la etapa de entrega, los grupos caracterizan una entrega del ciclo de versiones con claras etapas de respaldo manual. Asimismo, construyen entradas programadas que mueven las aplicaciones comenzando con una etapa y luego a la siguiente hasta que sean accesibles para los clientes. La mecanización de estos ciclos los hace controlados, versátiles y repetibles. Por lo tanto, los grupos de DevOps pueden transmitir sin esfuerzo, certeza y serenidad significativa, (Azure, 2017).

2.2.3.3.4. Funcionamiento.

La etapa de funcionamiento incluye mantener y verificar las aplicaciones, e investigar posibles problemas, en las condiciones en curso. Al recibir ensayos de DevOps, los grupos trabajan para garantizar confiabilidad, alta accesibilidad y el objetivo de no tener vacaciones en el marco, mientras implementan seguridad y administración. Los grupos de DevOps intentan reconocer los problemas antes de que influyan en el encuentro con el cliente y los moderan rápidamente a medida que surgen. Mantener esta observación requiere una telemetría extensa, precauciones importantes y una percepción completa de las aplicaciones y el marco fundamental, (Azure, 2017).

2.2.4. Procedimientos con DevOps.

DevOps es una práctica de diseño de productos que espera unir el avance de la programación y la actividad de programación, también podemos decir que es la asociación de personas, ciclos y avances para permitir la transmisión incesante de valor significativo a los clientes. El término DevOps, que comprende el avance y las operaciones de desarrollo, nombra una práctica de mejora de productos que reúne la etapa de avance y las actividades de TI. Implica que habrá coordinación y cooperación entre disciplinas que se desconectaron recientemente. El diseño de calidad y los grupos de seguridad también son importantes para el grupo más grande en el modelo DevOps. (Andreessen & Drucker, 2015.).

DevOps incorpora las prácticas del centro al igual que la organización y verificación, mejora, fabricación y prueba, entrega, observación y tareas. Estas prácticas, junto con los instrumentos y las innovaciones de DevOps, le permiten mecanizar el ciclo de vida de la aplicación. Los ciclos que solían ser manuales y tediosos para los grupos, por ejemplo, actualizar el código o aprovisionar otro clima, ahora deberían ser posibles de manera rápida e incesante cuando se utilizan aparatos y ensayos DevOps. Además, es más sencillo cumplir con pautas de bienestar y calidad inquebrantable, dado que estas contemplaciones se incorporan al ciclo. (Sandobalin-Guamán, 2016).

2.2.4.1.Etapas del Pipeline de DevOps

En el pipeline de DevOps se presentan etapas los cuales van desde el ambiente de desarrollo hasta el ambiente de producción, en la Figura 3, se muestran cada una de las etapas que esta conlleva. Esto puede variar de acuerdo a la organización, pero se trata de presentar un estándar que se puede aplicar y modificar en base a las necesidades de las organizaciones y sus procesos (Banica, Radulesco, Rosca, & Hagiu, 2017).

Administración de la librería de aplicaciones.

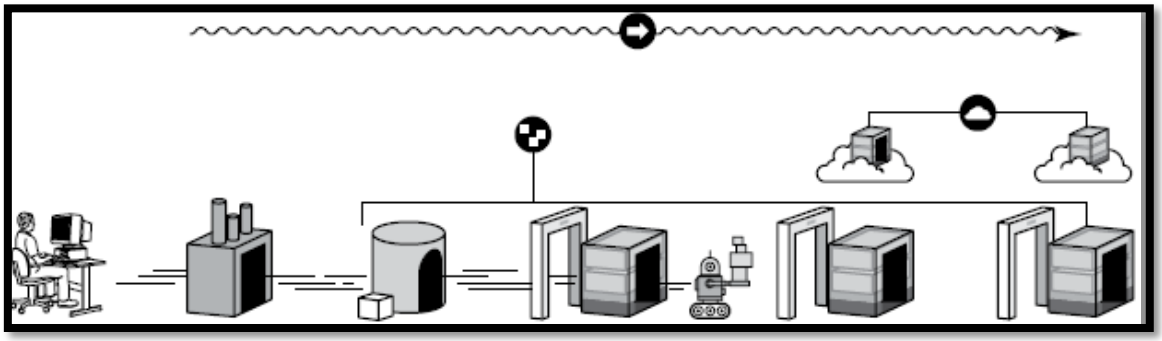

Figura 3. Etapas del Pipeline de DevOps.
Fuente: (Banica, Radulesco, Rosca, & Hagiu, 2017).

Ahora describiremos, cada uno de los ambientes del pipeline:

Desarrollo: Este ambiente provee herramientas a los desarrolladores para escribir y probar el código. Las herramientas para la gestión de control de fuentes, de colaboración, pruebas unitarias y planificación (Escueladedatos, 2019).

Construcción: En esta etapa se compila el código, se crea y se prueba por unidad lo que se va a desplegar. Las herramientas de compilación que se usan aquí varían de acuerdo a la plataforma que tienen las organizaciones. Por lo general utilizan servidores de compilación para facilitar la demanda de compilaciones que requiere una integración continua (Escueladedatos, 2019).

Repositorio de Paquetes: Conocido también como repositorio de archivos o artefactos permite almacenar el código creado en la etapa de Construcción. También almacenan los archivos asociados requeridos para facilitar el despliegue, tales como los archivos de configuración, de infraestructura y secuencias de comandos de implementación (Escueladedatos, 2019).

Pruebas: En esta etapa los grupos de Control de Calidad, usuarios de aceptación hacen las pruebas reales. Las herramientas que se utilizan aquí son las requeridas por el equipo de Control de Calidad tales como: gestión del entorno de prueba, gestión de datos de prueba (Escueladedatos, 2019).

Escenario y Producción: Aquí es donde se despliegan las aplicaciones. Las herramientas de gestión y monitoreo son las que se utilizan en esta etapa que permitan a las organizaciones supervisar los despliegues realizados en producción (Escueladedatos, 2019).

2.2.5. Practicas populares con DevOps.

Una de las prácticas más populares para el tema de la integración continua es Jenkins, el cual es rico en funciones y es ampliamente extensible con sus complementos. Además, Jenkins y sus complementos mejoran rápidamente. Hay una nueva versión menor de Jenkins lanzada semanalmente, principalmente con mejoras, ocasionalmente con errores. La comunidad gestiona la estabilidad central mediante el uso de una versión de soporte a largo plazo de Jenkins, que es madura y menos rica en características en comparación con la última versión. Para un sistema estable en un entorno complejo, debe supervisar, limpiar el almacenamiento, realizar copias de seguridad, mantener el control de sus scripts de Jenkins y limpiar y pulir constantemente, (Mitesh & Berg, 2017).

Jenkins ofrece muy buen soporte para crear canalizaciones de CI y CD.

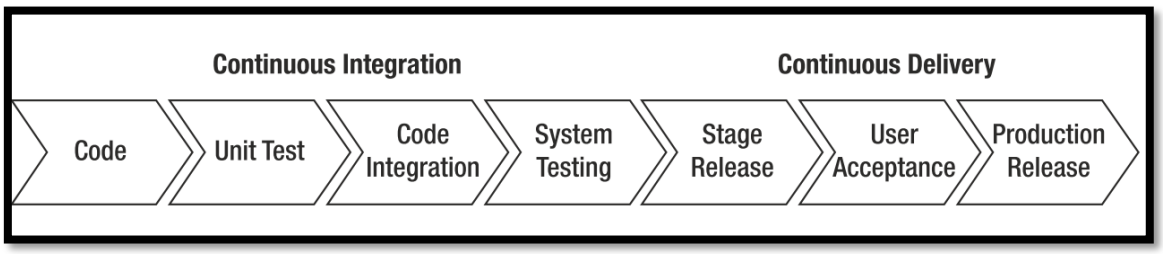

Figura 4. La tubería para la integración y entrega continua.
Fuente: (Mitesh & Berg, 2017).

2.2.5.1.Code.

El primer paso para crear la línea es el código. Cuando el código está listo, y el desarrollador lo ha completado y probado localmente, generalmente se libera en el repositorio central. Debido a que estamos implementando una integración continua, normalmente lanzamos en otra rama. Por lo general, esto está conectado a una política de ramificación. Por ejemplo,

podemos crear una rama con la cantidad de tareas que queremos implementar. El uso de una política de ramificación ayuda a definir las diferentes tareas realmente en desarrollo y a identificar qué parte del código se libera realmente, (Soni & Berg, 2017).

2.2.5.2.UnitTest

El siguiente paso en el desarrollo es la prueba unitaria. Este paso consiste en escribir la prueba para el código y ejecutarla directamente cuando comenzamos el proceso para CI y CD. Jenkins puede automatizar este proceso de una manera fácil. Con Jenkins, es posible conectar un repositorio y activar la prueba unitaria cuando el código se confirma directamente en el repositorio. Jenkins descarga el código del repositorio. Después de la descarga, podemos compilar el software y ejecutar el conjunto de pruebas. Después del edificio, si se pasa la prueba, podemos pasar a los siguientes pasos. Jenkins puede gestionar fácilmente todo este proceso cuando configuramos el proyecto y conectamos nuestro código al repositorio, (Soni & Berg, 2017).

2.2.5.3.Code Integration

Cuando el código ha sido probado, es hora de integrarlo en la rama principal. Esto puede ser automático o, en la mayoría de los casos, después de una política de revisión de código. Use una política de revisión de código para hacer cumplir la calidad del código y ayudar a compartir el conocimiento del sistema. Con esta política, requerimos un mínimo de dos personas para aprobar el código primero, antes de que se fusione con la rama principal. Cuando el código está integrado, podemos iniciar otra batería de pruebas unitarias en la rama principal. Para una política de CD, el código debe integrarse automáticamente, pero esto significa que no hay una revisión ejecutada por un humano. En este caso, podemos usar un análisis de código estático para identificar posibles problemas con el código. Con Jenkins, es posible usar una herramienta para ejecutar el análisis de código estático, a veces llamado análisis de código, si falla la compilación, (Soni & Berg, 2017).

2.2.5.4.System Testing

Cuando el código se integra en la rama principal, es posible comenzar algunas pruebas del sistema. Este tipo de prueba es diferente de la prueba unitaria, porque usamos datos reales, en lugar de simulados. Este tipo de prueba a veces se llama prueba de integración. Con este

tipo de pruebas, integramos y probamos todos los componentes del sistema. El alcance de este tipo de prueba identifica cualquier nuevo error introducido con la nueva característica, y los datos se derivan de un componente real del sistema, en lugar de un simulacro, porque queremos probar todo el sistema, (Soni & Berg, 2017).

2.2.5.5.Stage Release

Cuando la prueba del sistema tiene luz verde, Jenkins puede encargarse del lanzamiento del software. Esto puede ocurrir de diferentes maneras, dependiendo de cómo lancemos el software. Si, por ejemplo, queremos lanzar Docker, Jenkins puede insertar el código directamente en el registro que usamos para el servidor de escenario. Con Jenkins, también podemos ejecutar el script en Bash o Windows, para implementar el software en el servidor. El lanzamiento en un servidor de etapas es crucial para garantizar la calidad del software. Este servidor es utilizado por el equipo de ingeniería de control de calidad y debe ser similar al utilizado en el entorno de producción. De esta manera, es posible emular el entorno real y obtener más errores antes de lanzarlo a producción, (Soni & Berg, 2017).

2.2.5.6.User Acceptance

La aceptación del usuario es una fase importante en la construcción de la línea de liberación continua. Consiste en una batería de pruebas para verificar la funcionalidad, desde el punto de vista del usuario. Esta fase puede ser automática, basada en la prueba escrita por el equipo de ingeniería de QA, pero puede incluir alguna ejecución manual. En relación con la entrega continua, esta prueba se ejecuta primero con Jenkins, y después de que el código se lanza en algún servidor de pruebas, el equipo de control de calidad prueba el software manualmente. Esta prueba se utiliza para desarrollar un acuerdo con el usuario final sobre el software más deseable para lanzar, (Soni & Berg, 2017).

2.2.5.7.Production Relase

La última y más crítica fase del proceso de la línea es el lanzamiento de producción. De acuerdo con la política de entrega continua, esta fase puede ocurrir más de una vez al día. Además, podemos decidir no lanzar directamente a producción, sino optar por una versión de muestra en un servidor de pruebas y usar estos servidores para probar la versión con un número limitado de usuarios. Esto ayuda a identificar cualquier problema y a crear alertas

en el sistema. En CD, automatizamos todos los procesos y lanzamos cualquier cambio en el código directamente a producción. Esto puede ser un simple cambio de imagen o una etiqueta o una solución para un error. Con una política de entrega continua pura, esencialmente lanzamos una pequeña parte del software más veces al día. Jenkins puede ayudar a automatizar todos los aspectos de esta fase y crear una tubería completa para CD, (Soni & Berg, 2017).

2.2.5.8.Estrategia de Ramificación

Diseñar una buena estrategia de ramificación es esencial para tener una buena integración y entrega continua. La estrategia más popular para bifurcar es crear una bifurcación para cada característica, Figura 5. Cuando lanzamos el software en la sucursal, ejecutamos la prueba de la unidad y, si tiene luz verde, podemos fusionar el código. Para esto, el software GitFlow es muy útil, (Soni & Berg, 2017).

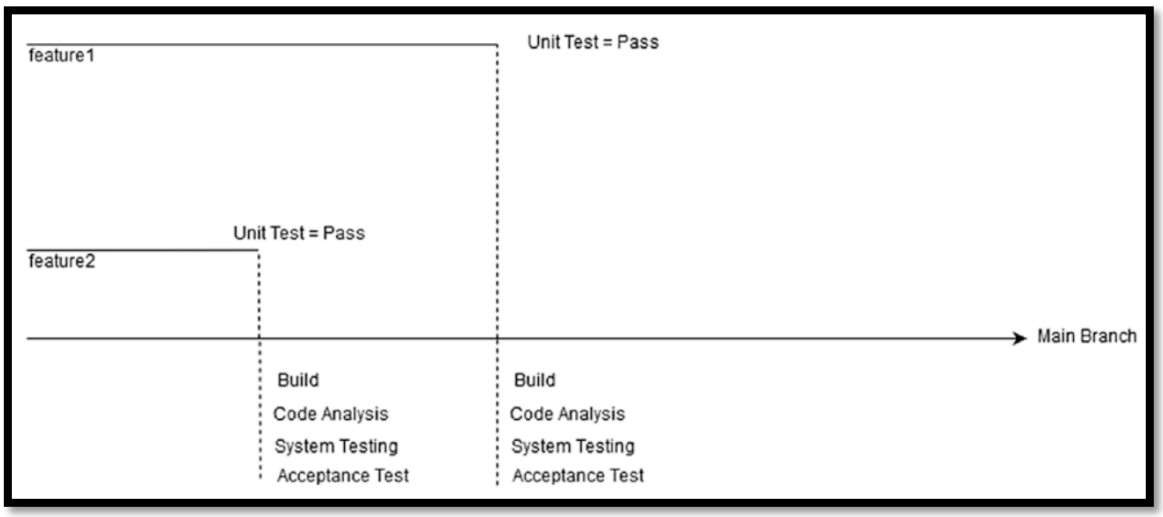

Figura 5. Características de estrategias de ramificación.
Fuente: (Soni & Berg, 2017).

Esta estrategia es la más simple y es completamente automática. Cada vez que fusionamos el código en la rama principal, Jenkins construye el código cuando la compilación está lista. Jenkins ejecuta algunos otros pasos para verificar la calidad del software que lanzamos.

El primer paso es el análisis de código o análisis de código estático. Esto se realiza mediante un complemento. Si el análisis del código es positivo, podemos pasar al siguiente paso, que es la prueba del sistema o la prueba de integración. Esto normalmente está diseñado para

probar el sistema con datos reales, en lugar de simulados. El objetivo es utilizar el componente real para la prueba, (amazon, 2018).

2.2.6. Integración Continua/Distribución Continua (CI/CD).

CI / CD es un método para distribuir aplicaciones a los clientes habitualmente mediante la utilización de la automatización en las fases de desarrollo de una aplicación. Las ideas principales acreditadas a CI / DC son la Integración continua, Distribución continua y la Implementación continua. CI / CD es una solución para los problemas que genera el código nuevo de equipos de desarrollo.

En particular, CI / CD consolida la automatización persistente y el control progresivo durante todo el ciclo de vida de la aplicación, desde las etapas de integración y prueba hasta las de distribución e implementación. Esta disposición de prácticas se conoce como canales CI/CD y es respaldada por DevOps, Figura 6, (RedHat, 2016).

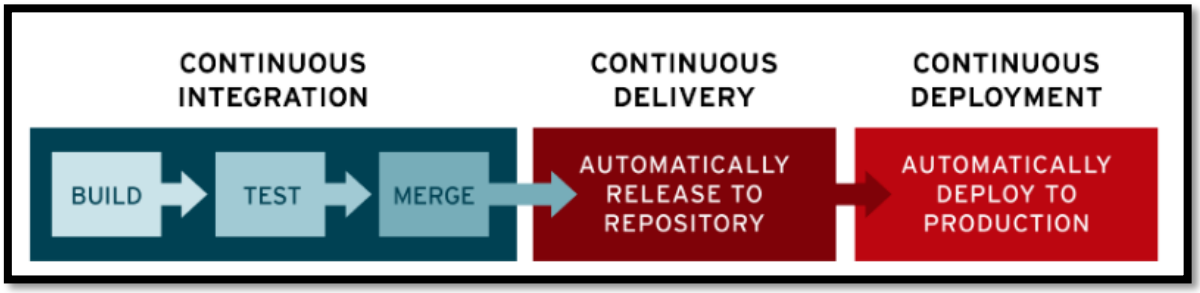

Figura 6. CI/CD.
Fuente: (RedHat, 2016)

El CI / DC puede determinar los actos conectados de integración y distribución continuas, o los tres actos conectados de integración continua, distribución continua e implementación continua Para confundir un poco más las cosas, la expresión "distribución continua" también incorpora los ciclos de implementación continua.

2.2.6.1.Integración continúa.

El objetivo de la mejora de la aplicación actual es que varios ingenieros trabajen al mismo tiempo en varios elementos de una aplicación similar. Sea como fuere, si una organización combina todo el código fuente diversificado en un solo día conocido como "Merge day", las asignaciones posteriores pueden ser aburridas y tediosas, y pueden llevar bastante tiempo. Esto sucede a la luz del hecho de que, cuando un diseñador que trabaja en segregación

actualiza un ajuste en una aplicación, existe la probabilidad de que este cambio tenga dificultades con esas progresiones al mismo tiempo ejecutadas por diferentes ingenieros. Este problema también puede verse afectado si cada diseñador modifica su propio clima de avance incorporado (IDE) cercano, en lugar de tener el consentimiento del grupo para un IDE basado en la nube, (RedHat, 2016).

La integración continua (CI) ayuda a los diseñadores a consolidar los cambios que realizan para codificar en una división mutua o "rama" con la mayor frecuencia posible, incluso día a día. Cuando las progresiones actualizadas por un ingeniero convergen en una aplicación, se aprueban con la mejora de la aplicación programada y la ejecución de varios grados de pruebas automatizadas típicamente pruebas unitarias y de coordinación para confirmar que las progresiones no han adulterado la aplicación. Esto implica probar todo, desde clases y actividades hasta los diferentes módulos que componen toda la aplicación. En caso de que una prueba individual identifique una disputa entre el código nuevo y el actual, CI simplifica la determinación de esos errores con frecuencia y rapidez, (RedHat, 2016).

2.2.6.2.Distribución continúa.

Tras la automatización de los diseños y las pruebas unitarias y la integración del CI, la distribución continua automatiza la llegada de ese código aprobado a un archivo. En consecuencia, para que el ciclo de distribución continua tenga éxito, es importante que CI se haya consolidado a partir de ahora en su canal de avance. El objetivo de la distribución continua es tener una base de código que se pueda transmitir en un entorno de creación en cualquier momento, (RedHat, 2016).

En una distribución continua, cada etapa, desde la combinación de cambios de código hasta la dispersión de diseños preparados para la creación, incluye pruebas de computación y descarga de código. Hacia el final de este ciclo, el grupo de actividades puede enviar una aplicación para llegar a la creación rápidamente y sin ningún problema, (RedHat, 2016).

2.2.6.3.Implementación continúa.

La última fase de la canalización consolidada de integración y distribución continuas es la implementación continua, que automatiza la llegada de una aplicación a la creación, ya que, en una distribución continua, que automatiza la llegada de una creación preparada en forma

a una bóveda de código. Dado que no hay una contribución manual a la etapa de la implementación continua, la disposición progresiva depende profundamente del plan correcto de la automatización de pruebas, (RedHat, 2016).

En términos prácticos, la implementación continua implica que un cambio ejecutado por un ingeniero en una aplicación puede activarse en un par de momentos de componerla esperando que haya pasado las pruebas automatizadas. Esto hace que el ciclo de recibir y unirse incesantemente a las críticas de los clientes sea mucho más simple. Juntos, estos ensayos de CI / CD conectados hacen que la organización de una aplicación sea más segura, ya que es más sencillo entregar cambios a las aplicaciones en partes pequeñas, en lugar de hacerlo al mismo tiempo. Sea como fuere, también hay muchas iniciativas iniciales por hacer, ya que las pruebas automatizadas deben estar destinadas a obligar a las diferentes fases de prueba y entrega en el canal CI / CD, (RedHat, 2016).

2.2.7. Porque CI/CD con GitLab.

De hecho, la explicación principal se basa en que es sencilla, podemos fabricar un pipeline total con un solo instrumento. Diferentes razones se basarían en que es rápido y es de código abierto. Con GitLab CI / CD en un lugar similar, podemos crear tickets, unir demandas, redactar código y desarrollar una coordinación y transmisión constantes sin depender de otra aplicación. Es fundamentalmente una excursión constante, GitLab CI/CD ejecuta sus expansiones en una cosa llamada GitLab Runners. Estos runners son máquinas virtuales aisladas que ejecutan pasos predefinidos a través de la API, la API de GitLab CI para ser exactos. Este instrumento independiente permite que las actividades pasen por las construcciones de los pipelines más rápido, comparada a cuando se corren en una sola instancia.

2.2.8. Centos 8.

CentOS es una distribución de Linux que se lanzó en marzo de 2004. El proyecto de código abierto, desarrollado y apoyado por una gran comunidad, se basa en los paquetes fuente de Red Hat Enterprise Linux (RHEL), una distribución comercial de pago que solo se puede utilizar en combinación con contratos de soporte. Red Hat, el proveedor de RHEL, está obligado a publicar como recurso open source el código fuente de los componentes de software integrados en las diversas licencias libres. Esto permite a los desarrolladores

apoyarse sin ningún coste en el código fuente de RHEL durante la programación como parte del proyecto CentOS, (DigitalGuide, 2015).

2.2.9. VSPhere Client.

VMware vSphere Client es una aplicación basada en web que se conecta a vCenter Server para que los administradores de TI puedan administrar instalaciones y manejar objetos de inventario en una implementación de vSphere, (TechTarget, 2015).

VSphere Client presenta una interfaz gráfica de usuario GUI con un navegador de objetos, el espacio de trabajo principal y el panel de tareas y alarmas. A través de esta GUI, los administradores de vSphere pueden administrar y supervisar los objetos listados en un centro de datos virtualizado, (TechTarget, 2015).

2.2.10. Metodología de la investigación.

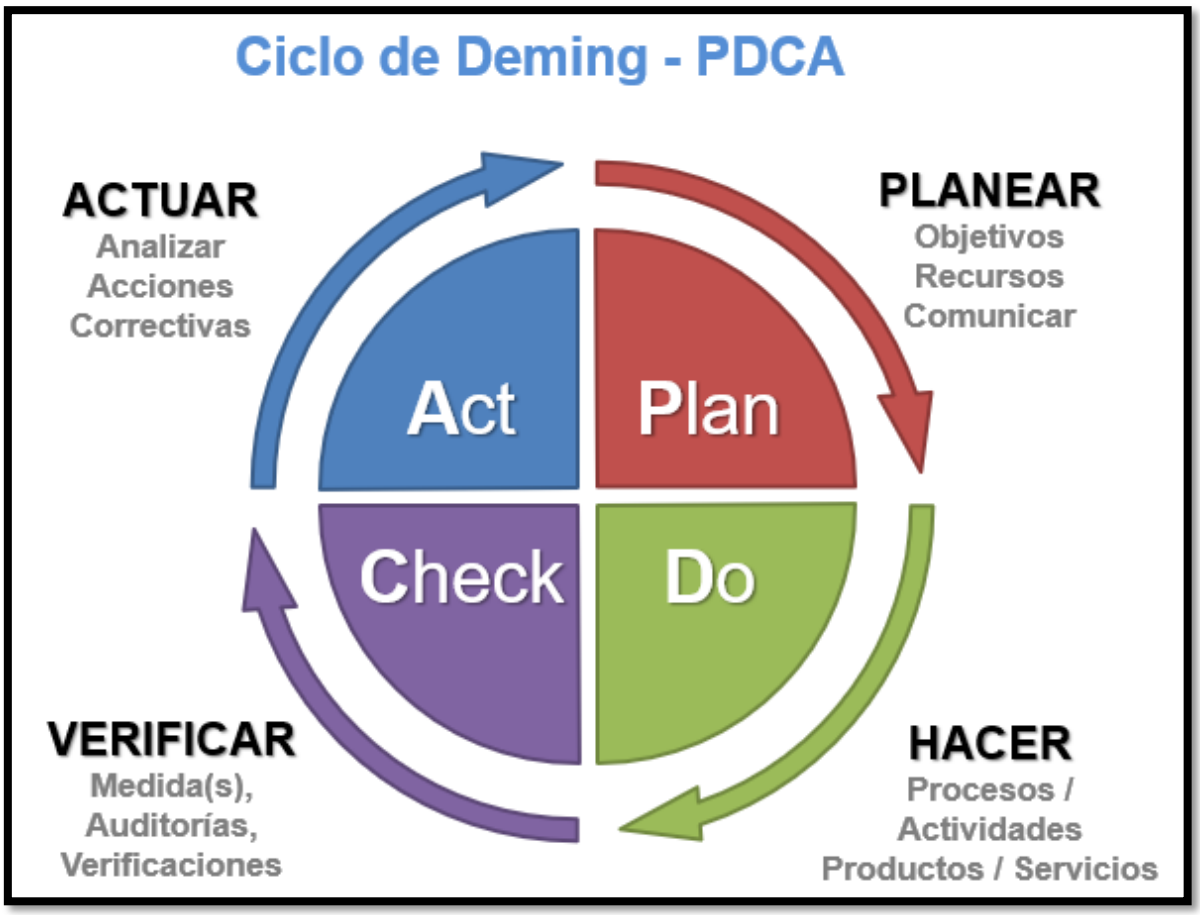

Figura 7. Metodología de la investigación.
Fuente: (Elaboración propia, 2020)

2.2.10.1. Descripción de la metodología ciclo deming.

Fase1. Planear.

Comienza con una etapa de planificación: Para hacer esto, primero se decide la circunstancia actual. El tema se describe a continuación con el objetivo de que pueda decidir con precisión cómo se debe lograr el objetivo. Esta etapa también incorpora la sólida disposición de los activos vitales. Aquí también decide tanto el estado actual como los activos adicionales que se necesitarán, (Garcia, 2016).

Por último, el grupo también debe coincidir en los factores de logro. ¿Qué debe ocurrir para que el cambio se considere efectivo? Solo si los objetivos se han caracterizado explícitamente, se puede estimar además si se ha logrado un resultado adecuado. Esto también incorpora objetivos de selección que sean razonables y factibles. No se ve bien caracterizar los triunfos idealistas que no se pueden lograr en un tiempo sensato y con un esfuerzo legítimo, (Garcia, 2016).

Fase 2. Hacer.

Después de la planificación, comienza la etapa de implementación. El grupo o individuo necesita actualmente configurar lo que ha dispuesto en el escenario principal. Lo mejor aquí es seguir con pequeños avances y cuestionar el uso una y otra vez. En este sentido, podemos garantizar que no perdemos el control durante la implementación y que mantenemos el rumbo. Poco a poco, se ha demostrado adicionalmente con el progreso que el ciclo de cambio debe intentarse desde una perspectiva menor, por ejemplo, primero en un elemento y no en toda la gama de elementos, o en una oficina en lugar de en toda la organización.

Eventualmente, este avance posterior también puede considerarse como una etapa de prueba. El tiempo invertido en el uso de esta etapa también se utiliza para aprender: dado que algo se ha organizado con precisión, no significa que deba funcionar en la práctica. La experiencia recogida en la etapa do conduce directamente a la tercera etapa, (Garcia, 2016).

Fase 3. Verificar.

Durante la revisión o verificación, se contrastan los resultados obtenidos y se caracterizan las metas. En esta etapa, echa un vistazo a lo que funcionó admirablemente y lo que no

cumplió con su forma. Es fundamental analizar imparcialmente el acuerdo y su ejecución. Los problemas de la etapa de uso no deben verse como decepciones, sino como ocasiones para sacar provecho de ellos, ya que esta etapa está destinada a eso.

La etapa de verificación resume, pero además disecciona: ¿por qué no salió todo como se esperaba? Cuando descubra cómo surgieron los problemas, será todo menos difícil cambiar el arreglo y lograr mejores resultados la próxima vez, (Garcia, 2016).

Fase 4. Actuar.

Cuando se conocen los problemas y se han reconocido las causas, la disposición se puede reorganizar y actualizar, por último. Si bien la etapa hacer se completó como una prueba y se realizó desde una perspectiva menor, el cuarto paso envuelve una imagen más amplia. Dependiendo de la estructura en la que utilice el ciclo PDCA, puede aumentar su aplicación.

Una vez finalizada la etapa Actuar, el nuevo estado de envío se considera estándar. Este grado de valor no debería abandonarse actualmente. Posteriormente, igualmente conviene introducir un tipo de control. Por lo general, puede analizar su uso y asegurarse de que no se repitan los viejos diseños de errores. Además, otra persona un entrenador, un jefe, etc. puede asumir el control de este trabajo de control. Es importante que el avance futuro no vuelva al estado anterior a la actualización del ciclo, (Garcia, 2016).

2.2.10.2. Ventajas y desventajas del ciclo PDCA.

El círculo de Deming es una valiosa herramienta para realizar mejoras de una manera manejable e inteligente. En lugar de cambiar precipitadamente los métodos estándar, continuamos con pequeños avances y constantemente bajo una percepción exigente. Sin embargo, esta es también una de las cargas increíbles del ciclo de Deming: es necesario diseñar suficientes oportunidades para ejecutar los nuevos ciclos. El ciclo PDCA no tiene en cuenta la investigación rápida.

Tabla 4. Ventajas y desventajas del PDCA.

Ventajas	Desventajas
Puede ayudar en todo tipo de situaciones.	Una definición poco específica puede llevar a un uso incorrecto del método.
La configuración es sencilla y requiere poca orientación.	Los cambios deben planificarse para periodos de tiempo largos.
La idea cíclica invita a la mejora constante.	No permite una resolución rápida de problemas que requieran de mucha urgencia.
El enfoque de revisión permite controlar y analizar la implementación.	

Fuente: (Ionos, 2019)

2.2.11. Indicadores de gestión.

Un indicador de gestión podemos decir que es una expresión cuantitativa del comportamiento y desempeño de un proceso, el cual su magnitud, al ser comparada con algún nivel de referencia, puede estar señalando una desviación sobre el cual se toman acciones correctivas o preventivas dependiendo del caso el caso, (Minero, 2019)

2.2.11.1. Tipos de indicadores de gestión.

- Indicadores de eficiencia
- Indicadores de eficacia
- Indicadores de cumplimiento
- Indicadores de evaluación
- Indicadores de Capacidad
- Indicadores de Productividad
- Indicadores de Calidad
- Indicadores de Rentabilidad
- Indicadores de Competitividad
- Indicadores de Valor

2.2.12. Prueba t de Student.

El t de student se utiliza para determinar si puede haber una diferencia bastante significativa entre la media de dos grupos, es un tipo de estadística deductiva con lo mencionado asumimos que las variables dependientes tienen un tipo de distribución normal, en ella se especifica el nivel de probabilidad o nivel de significación que se esta dispuesto a aceptar, (EstadisticaInvestigación, 2017).

La formula general del t de student es:

$$t = \frac{X - \mu}{s/\sqrt{n}}$$

Lo detallamos de la siguiente manera el numerador es la diferencia a probar y el denominador la desviacion estándar de la diferencia también llamado el error estándar, t representa el valor estadístico que se busca, x es el promedio de la variable de la muestra analizada, el miu es el promedio poblacional de la variable que se estudiara, en el denominador se tiene a s como representante de la muestra de la desviación estándar y n el tamaño de esta, (EstadisticaInvestigación, 2017).

2.2.12.1. Pasos para resolver con el t de student.

- Paso 1. "Plantear la hipótesis nula (H0) y la hipótesis alternativa (H1). La hipótesis alternativa plantea matemáticamente lo que queremos demostrar, en tanto que la hipótesis nula plantea exactamente lo contrario," (EstadisticaInvestigación, 2017).
- Paso 2. "Determinar el nivel de significancia (rango de aceptación de la hipótesis alternativa), a. Se considera un nivel alfa de: 0.05 para proyectos de investigación; 0.01 para aseguramiento de la calidad; y 0.10 para estudios o encuestas de mercadotecnia," (EstadisticaInvestigación, 2017).
- Paso 3. "Evidencia muestral, se calcula la media y la desviación estándar a partir de la muestra," (EstadisticaInvestigación, 2017).
- Paso 4. "Se aplica la distribución T de Student para calcular la probabilidad de error por medio de la fórmula general presentada al principio y se contrasta con el valor T obtenido de la tabla correspondiente" (EstadisticaInvestigación, 2017).

- Paso 5. “En base a la evidencia disponible se acepta o se rechaza la hipótesis alternativa. Si la probabilidad de error (p) es mayor que el nivel de significancia se rechaza la hipótesis alternativa. Si la probabilidad de error (p) es menor que el nivel de significancia se acepta la hipótesis alternativa,” (EstadisticaInvestigación, 2017).

Sim embargo al final lo que se tiene que contrastar es el valor de T que se haya obtenido en el problema contra el valor T crítico que se obtiene de la tabla de T de Student.

Aquí podemos encontrar la tabla de tabla de T de Student. Tabla

2.3. Métodos y procedimientos.

2.3.1. Imágenes Docker.

Al igual que las máquinas virtuales, los soportes de Docker se basan en imágenes, que se examinan solo con diseños con todas las pautas que requiere el motor de Docker para hacer un compartimento. Como duplicado versátil de un titular, una imagen de Docker se representa como un documento de contenido (Dockerfile). Antes de comenzar un compartimento en un marco, se apila un paquete con la imagen relacionada si la imagen ahora no se conserva localmente. La imagen apilada prepara todos los marcos de registro con los límites esenciales para la ejecución. Un compartimento puede considerarse como un ciclo de ejecución de una imagen, (Ionos, 2020).

2.3.1.1.DockerFile.

Es un archivo de configuración que se utiliza para crear imágenes. En ese archivo indicamos qué es lo que queremos que tenga la imagen, y los distintos comandos para instalar las herramientas. Esto sería un ejemplo de DockerFile para tener una imagen de Ubuntu con Git instalado, (Garzas, 2015).

2.3.1.2.Containers.

Son instancias en ejecución de una imagen. Son los que ejecutan cosas, los que ejecutarán nuestra aplicación. El concepto de contenedor es como si restauráramos una máquina virtual a partir de un snapshot, (Garzas, 2015).

A partir de una única imagen, podemos ejecutar varios contenedores, Figura 8.

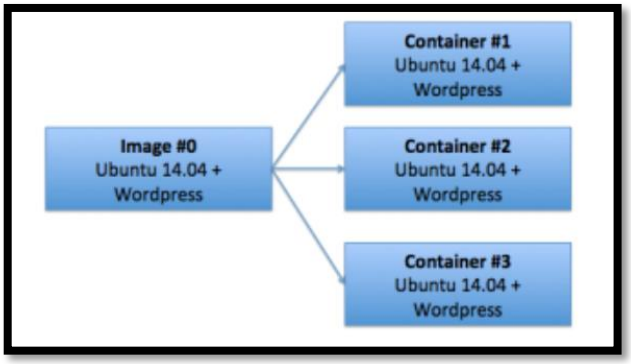

Figura 8. Ejemplo de varios contenedores.
Fuente: (Garzas, 2015)

2.3.1.3.Volumes.

No es una buena práctica guardar los datos persistentes dentro de un contenedor de Docker. Para eso están los volúmenes, fuera de los contenedores. Así podremos crear y borrar contenedores sin preocuparnos por que se borren los datos. Además, los volúmenes se utilizan para compartir datos entre contenedores, (Garzas, 2015).

2.3.1.4.Links.

Sirven para enlazar contenedores entre sí, que están dentro de una misma máquina, sin exponer a los contenedores cuáles son los datos de la máquina que los contiene, (Garzas, 2015).

2.3.2. Docker Hub.

El Docker Hub es un registro para repositorios de software basado en la nube, es decir, una especie de biblioteca para las imágenes Docker. Este servicio online está formado por repositorios públicos y privados. En los primeros se ofrece a los usuarios la posibilidad de subir sus propias imágenes y compartirlas con la comunidad. Aquí se dispone de un gran número de imágenes Docker oficiales realizadas por el equipo de desarrolladores de la plataforma, así como de proyectos de código abierto consolidados. Por el contrario, en los repositorios privados del registro no todo el mundo tiene acceso a las imágenes que se cargan, aunque estas sí pueden ser compartidas dentro de una misma empresa o en un determinado círculo, (Garzas, 2015).

2.3.3. Motor de Docker o (Docker engine).

El corazón de cualquier proyecto Docker es el motor de Docker, esto es, una aplicación cliente-servidor de código abierto disponible para todos los usuarios en la versión actual en todas las plataformas establecidas, (Ionos, 2020).

Los componentes que conforman la arquitectura básica de este motor son: un daemon con funciones de servidor, una interfaz de programación (API) basada en REST (Representational State Transfer) y la terminal del sistema operativo (Command-Line Interface, CLI) como interfaz de usuario client, (Ionos, 2020).

- Daemon de Docker: en el motor de Docker se utiliza un proceso daemon como servidor que funciona en un segundo plano del sistema host y permite el control central del motor de Docker. Además, se encarga de crear y administrar todas las imágenes, contenedores o redes, (Ionos, 2020).
- La APIREST: especifica una serie de interfaces que permite a otros programas interactuar con el daemon y darle instrucciones. Uno de estos programas es la terminal del sistema operativo, (Ionos, 2020).
- La terminal: Docker utiliza la terminal del sistema operativo como programa cliente, el cual interacciona con el daemon a través de la API REST y permite a los usuarios controlarlo a través de scripts o comandos, (Ionos, 2020).

Docker permite ejecutar, parar o gestionar los contenedores de software directamente desde la terminal. Con el comando docker e instrucciones como build (crear), pull (descargar) o run (ejecutar) es posible comunicarse con el daemon, lo que posibilita que tanto cliente como servidor se encuentren en el mismo sistema. Además, es posible dirigirse al daemon en otro sistema diferente. En función del tipo de conexión que se deba establecer, la comunicación entre cliente y servidor se produce bien a través de la API REST, de sockets de UNIX o de una interfaz de red, (Ionos, 2020).

2.3.4. Pipelines.

Un pipeline Jenkins es un conjunto de plugins que soporta la implementación e integración de pipelines tuberías de despliegue continuo en Jenkins. Un pipeline es un conjunto de instrucciones del proceso que siga una aplicación desde el repositorio de control de versiones

hasta que llega a los usuarios. Cada cambio en tu software, lleva a un complejo proceso hasta que es desplegado. Este proceso incluye desde construir software de forma fiable y repetible conocido como “build”, hasta realizar todos los pasos de testing y despliegues necesarios. La definición de un pipeline Jenkins, se escribe en un fichero de texto llamado Jenkinsfile que se puede subir al repositorio junto con el resto del proyecto de software. (Moreno, 2019).

CAPÍTULO III. Materiales y métodos

3.1. Lugar de ejecución.

La investigación y puesta en marcha del ambiente (CI/CD) será en el área de TI de la Universidad Peruana Unión de la Filial Juliaca, ya que cuenta con un equipo profesional para la adopción de nuevas metodologías.

3.2. Herramientas.

3.2.1. Docker.

Es una tecnología de creación de contenedores que permite la creación y el uso de contenedores de Linux. Docker es sin duda la tecnología más popular en estos días en el mundo de la Tecnología de la Información (TI). Principalmente, hay dos tendencias principales en el panorama de Docker. Primero la plataforma Docker de código abierto está continuamente equipada con características y funcionalidades más correctas y relevantes para convertirla en la plataforma de TI más poderosa y pionera, no solo para desarrolladores de software sino también para equipos operativos de TI locales y fuera de las instalaciones. La segunda tendencia es la adopción sin precedentes de la tecnología de contenedores inspirada en Docker por parte de varios proveedores de servicios y soluciones de TI en todo el mundo con el fin de ofrecer una gama creciente de ofertas premium para sus consumidores y clientes. La simplicidad mejorada en el desarrollo de nuevas aplicaciones de software, el despliegue automatizado y acelerado de contenedores Docker y la extrema maniobrabilidad de los contenedores Docker se están promocionando ampliamente como los diferenciadores clave del éxito sin precedentes de este paradigma único (Chelladhurai, Vinod, & Pethuru, 2017).

3.2.2. Jenkins.

Jenkins es un servidor de automatización de código abierto el cual es ampliamente utilizado por muchas organizaciones con el fin de implementar prácticas populares de DevOps, como la integración continúa y la entrega continúa. Jenkins es rico en funciones y es ampliamente extensible con sus complementos. Además, Jenkins y sus complementos mejoran rápidamente. Hay una nueva versión menor de Jenkins lanzada semanalmente, principalmente con mejoras, ocasionalmente con errores. La comunidad gestiona la

estabilidad central mediante el uso de una versión de soporte a largo plazo de Jenkins, que es madura y menos rica en características en comparación con la última versión. Para un sistema estable en un entorno complejo, debe supervisar, limpiar el almacenamiento, realizar copias de seguridad, mantener el control de sus scripts de Jenkins y limpiar y pulir constantemente (Mitesh & Berg, 2017).

3.2.3. GitLab.

GitLab es una suite de desarrollo todo en uno. Proporciona repositorios Git, seguimiento de problemas, CI y un registro Docker. La estrecha integración entre Git, el seguimiento de problemas y los registros de Docker en GitLab lo convierten en una solución ideal para el desarrollo de aplicaciones. GitLab es de código abierto y se puede instalar localmente (Jiménez, 2018).

3.3. Metodología de la investigación.

Esta investigación es de tipo aplicada ya que nos basaremos en los hallazgos tecnológicos para de esa manera generar el conocimiento con la aplicación directa a los problemas de la institución. Investigación Aplicativo Según (Chavarriaga, Arboleda, & Lidis, 2004) es la etapa en donde se busca madurar la tecnología definida en la etapa inicial, trabajando en el desarrollo de la tecnología y en su aplicación en situaciones reales. A medida que se tienen experiencias de aplicación de la tecnología por parte del grupo de investigación y empresas de la industria, se encuentran las posibles fallas en el modelo de solución propuesto y se definen las adaptaciones que puedan ser requeridas para su aplicación en empresas.

3.3.1. Arquitectura de solución.

La arquitectura se explica de la siguiente manera cada vez que un programador realice una solicitud de subida de código (push) en el repositorio git remoto y suba código a la rama de desarrollo (Develop), Jenkins automáticamente detectara esta petición bajo una llamada WEBHOOK que realiza el lanzamiento de las tareas como la compilación de la aplicación, llame a Sonar para ejecutar los test y realizar el análisis de la calidad del código y, si todo va bien, se vuelve a realice un lanzamiento de realice el cual realiza una lanzamiento de

aplicación a la rama master de git, con lo que se realiza el despliegue al servidor de producción.

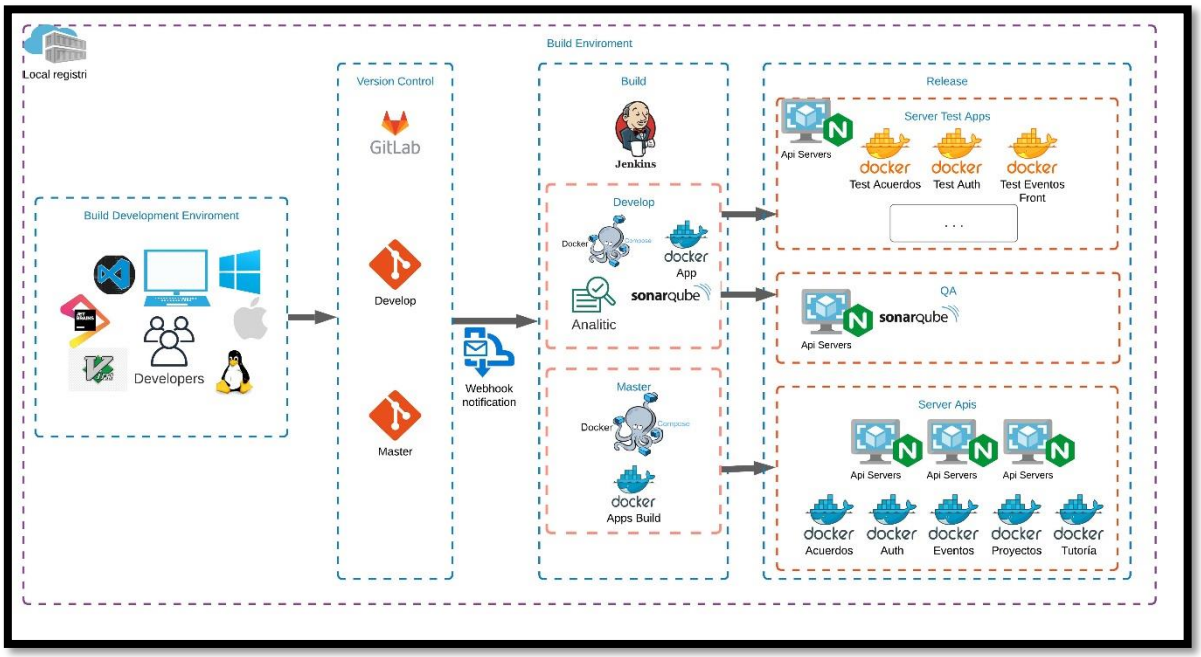

Figura 9. Arquitectura de solución.
Fuente: (Elaboración propia, 2020)

3.4. Desarrollo de la investigación

El enfoque basado en las etapas para la integración continua, en este proyecto se reflejó en 4 etapas, y en la estructura se vera de la siguiente manera, Planear, Hacer, Verificar y Actuar.

3.4.1. Elementos del proceso.

3.4.1.1. Planear.

Para esta etapa se analizó y diseño el ambiente de desarrollo, en la Figura 10 podemos ver cómo era el proceso del ambiente de desarrollo, como se muestra en la imagen la interacción es en su mayoría con personas, viendo este detalle es que se optimizo el ambiente de desarrollo. También se seleccionaron las herramientas adecuadas para mejorar el ambiente de desarrollo.

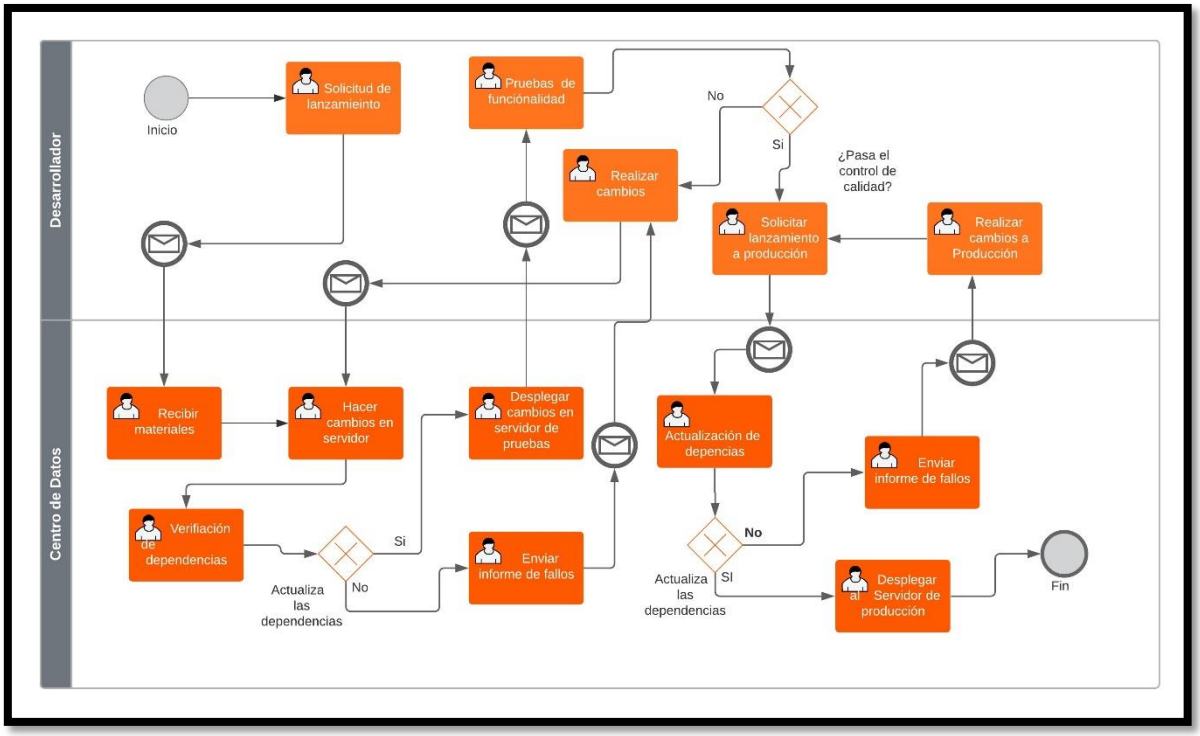

Figura 10. Ambiente de desarrollo anterior.
Fuente: (Elaboración propia, 2020)

Para poder empezar con el diseño del ambiente primero se analizó recopilando los proyectos los cuales estarán en nuestra integración continua y todo lo que conlleva, Figura 11.

Figura 11. Proyectos UPeU.
Fuente: Elaboración propia (2020).

De la misma manera recopilamos todas las herramientas que se usaran para la integración continua, las cuales fueron:

- S.O. para el servidor Centos 8
- Jenkins que usa java como fuente para el despliegue
- Nginx

- Certificado SSL
- Pipeline
- BlueOcean
- Docker
- Sonar
- Git

3.4.1.2. Hacer.

Para esta etapa se define un ambiente de integración y despliegue continúo basado en la cultura DevOps realizando las configuraciones y haciendo uso de las herramientas. Para los ambientes de prueba se inició con la configuración de los servidores, creando un usuario con privilegios de acceso remoto, Figura 12.

Figura 12. Usuario con privilegios para acceso remoto.
Fuente: (Elaboración propia, 2020)

Se asignó las llaves de acceso a los servidores de producción y pruebas, Figura 13.

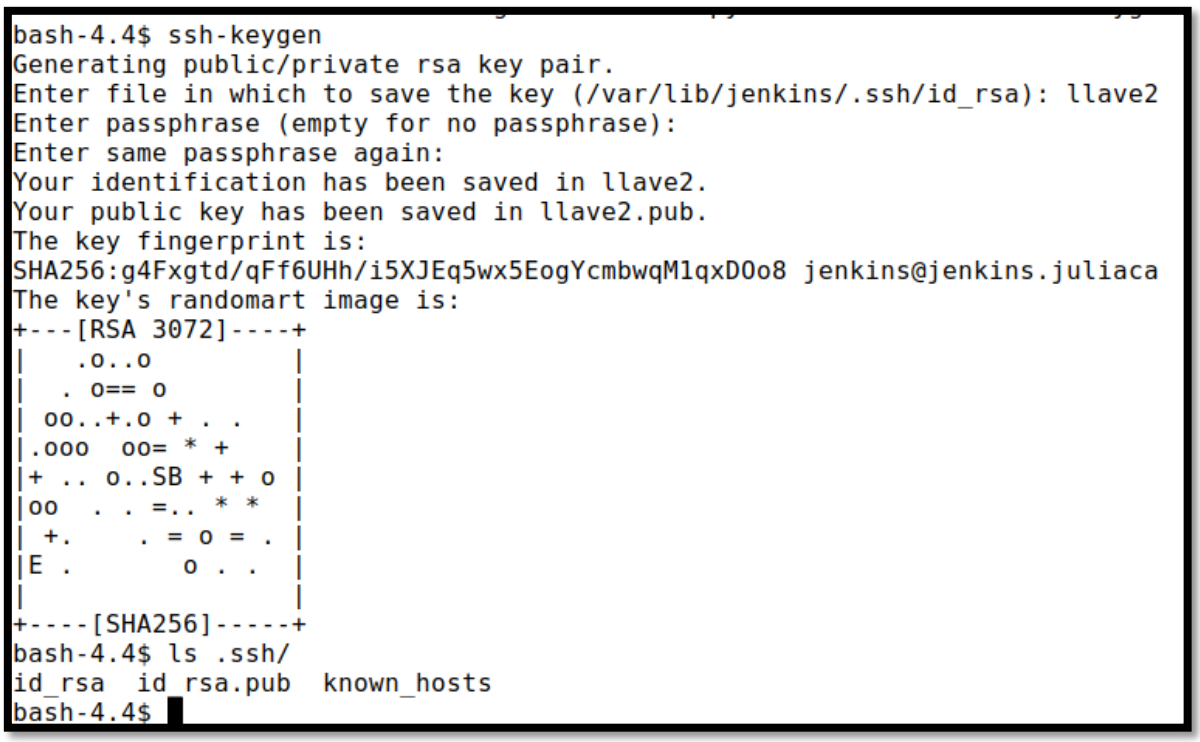

Figura 13. Asignando llaves de acceso.
Fuente: (Elaboración propia, 2020)

Para el despliegue en los servidores de producción y pruebas se instaló Docker y Docker Compose, Figura 14 y Figura 15.

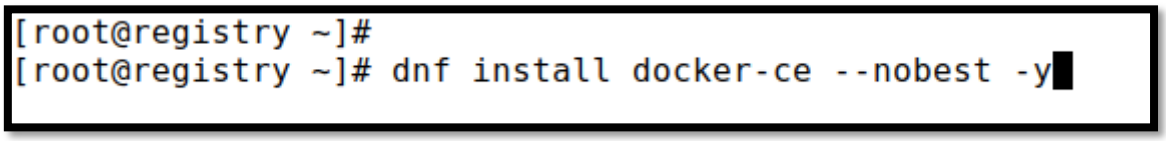

Figura 14. Instalando Docker.
Fuente: (Elaboración propia, 2020)

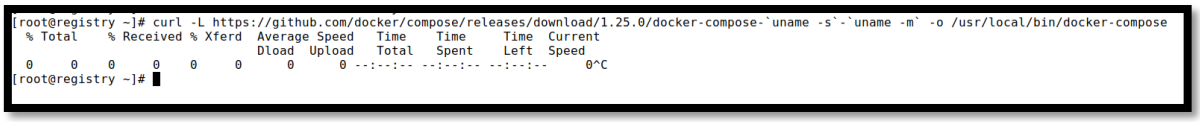

Figura 15. Instalando Docker Compose.
Fuente: (Elaboración propia, 2020)

De la misma manera para tener un buen funcionamiento se incrementó la capacidad de memoria para los servidores, Figura 16.

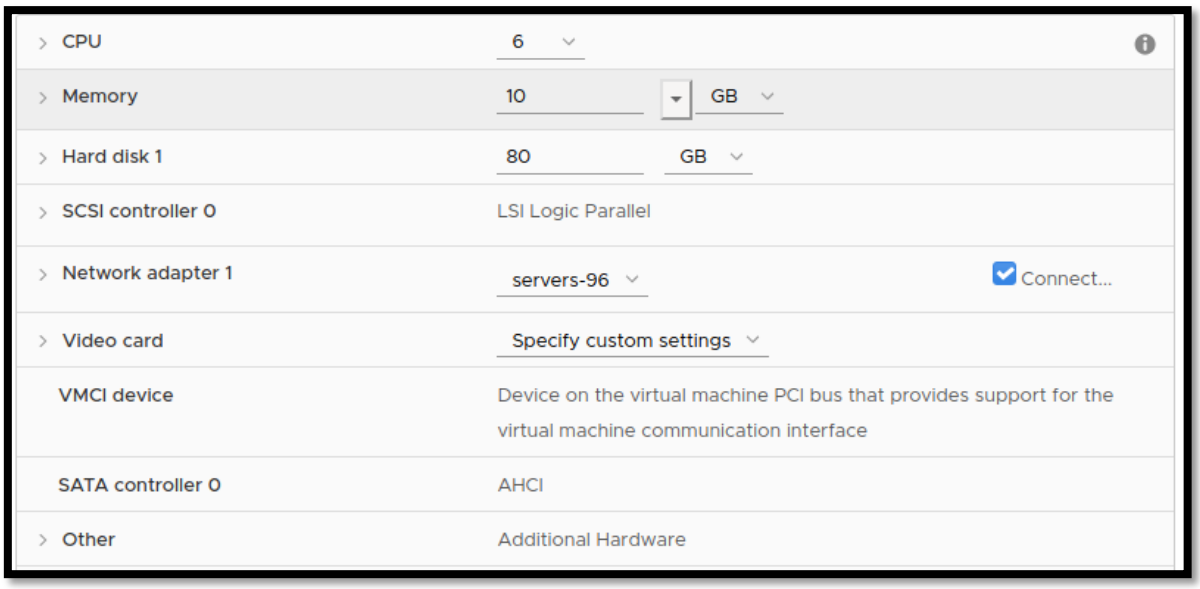

Figura 16. Aumento de recursos para los servidores.
Fuente: (Elaboración propia, 2020)

También se agregó un servicio Proxy para el balanceo de carga en Nginx, Figura 17.

```
server {
        listen 80;
        server_name auth.upeu.pe;
        return 301 https://auth.upeu.pe;
}


upstream loadauth{

        server authserv1.upeu.pe;
        server authserv2.upeu.pe;
        server authserv3.upeu.pe;

}


server {
  listen 443;
  server_name auth.upeu.pe;
  ssl     on;
  ssl_certificate    /etc/nginx/ssl/upeu-pe.crt;
  ssl_certificate_key    /etc/nginx/ssl/upeu-pe.key;

  location / {
    proxy_pass       https://loadauth;
    proxy_set_header Host proyectos.upeu.pe;
    proxy_set_header X-Forwarded-For $remote_addr;
  }
}
```

Figura 17. Balanceo de carga en Nginx.
Fuente: (Elaboración propia, 2020)

Se cambió los parámetros de Selinux a permissive para registrar y alertar al administrador de que se ha violado alguna regla, Figura 18.

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=permissive
# SELINUXTYPE= can take one of these three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

Figura 18. Cambiando parámetro de Selinux a Permissive.
Fuente: (Elaboración propia, 2020)

De la misma manera también se asignó las variables de seguridad y tamaño de memoria a utilizar, Figura 19.

```
[root@sonar ~]# cat /etc/sysctl.d/99-sonarcube.conf
vm.max_map_count=262144
fs.file-max=65536

[root@sonar ~]# cat  /etc/security/limits.d/99-sonarqube.conf
sonarqube - nofile 65536
sonarqube - nproc 4096

[root@sonar ~]#
```

Figura 19. Variables de seguridad y tamaño de memoria.
Fuente: (Elaboración propia, 2020)

También se instaló Java 11 OpenJdk, Figura 20.

```
root@sonar ~]# dnf install -y java-11-openjdk-headless.x86_64
ast metadata expiration check: 2:27:10 ago on Tue 27 Oct 2020 08:30:52 PM -05.
ackage java-11-openjdk-headless-1:11.0.8.10-0.el8_2.x86_64 is already installed.
ependencies resolved.
othing to do.
omplete!
root@sonar ~]#
```

Figura 20. Instalando Java 11.
Fuente: (Elaboración propia, 2020)

Se realizó la instalación de postgres para las bases de datos con Sonar, Figura 21, Figura 22, Figura 23 y Figura 24 también se hizo las configuraciones de seguridad de postgres, Figura 25.

```
[root@sonar ~]# dnf install postgresql-server
Last metadata expiration check: 2:28:52 ago on Tue 27 Oct 2020 08:30:52 PM -05.
Package postgresql-server-10.14-1.module_el8.2.0+487+53cc39ce.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[root@sonar ~]#
```

Figura 21. Instalación de Postgres.
Fuente: (Elaboración propia, 2020)

```
[root@sonar ~]# systemctl start postgresql.service
```

Figura 22. Instalación de postgres 2.
Fuente: (Elaboración propia, 2020)

Figura 23. Instalación de postgres 3.
Fuente: (Elaboración propia, 2020)

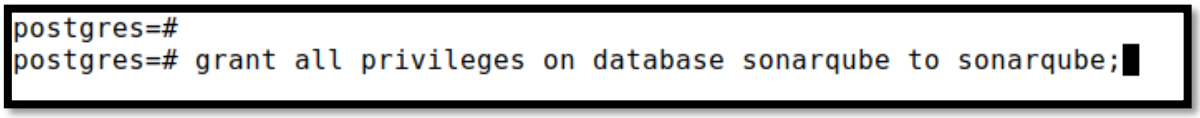

Figura 24. Privilegios de postgres a sonar.
Fuente: (Elaboración propia, 2020)

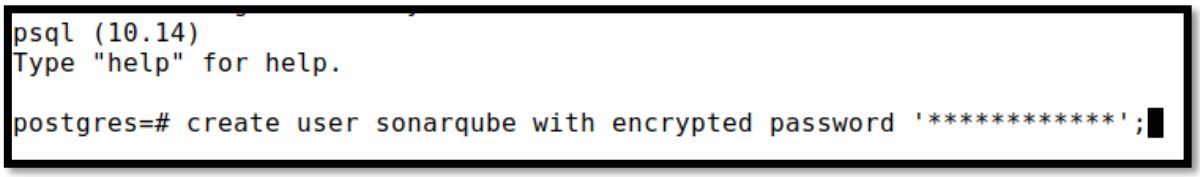

Figura 25. Seguridad en Postgres.
Fuente: (Elaboración propia, 2020)

Se creó los usuarios del sistema para el despliegue de sonar, Figura 26.

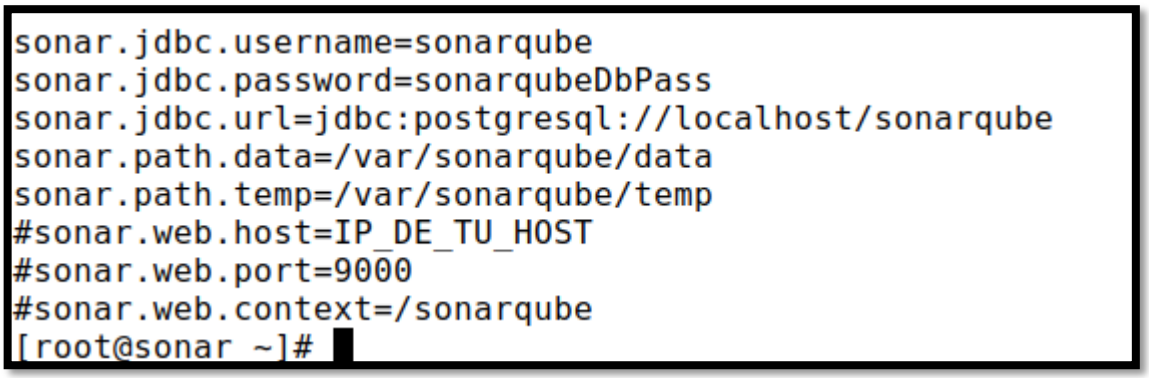

Figura 26. Usuario para el despliegue de sonar.
Fuente: (Elaboración propia, 2020)

Se instaló SonarQube, Figura 27, Figura 28, Figura *29*, Figura 30 y Figura 31 también se habilito los servicios de systemd para el inicio de sonar, Figura 32.

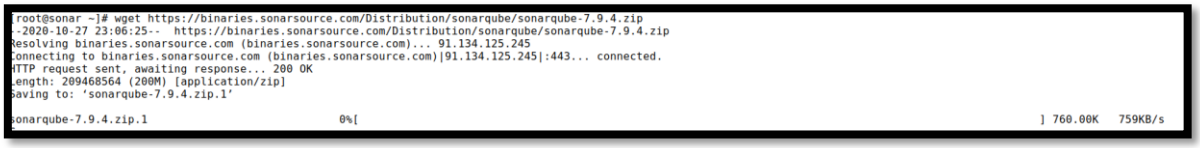

Figura 27. Instalando Sonarqube.
Fuente: (Elaboración propia, 2020)

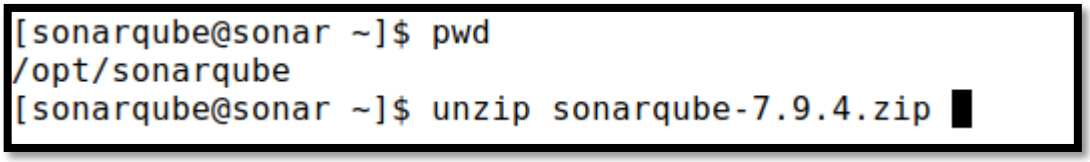

Figura 28. Instalando Sonarqube2.
Fuente: (Elaboración propia, 2020)

Figura 29. Instalando Sonarqube3.
Fuente: (Elaboración propia, 2020)

Figura 30. Instalando Sonarqube4.
Fuente: (Elaboración propia, 2020)

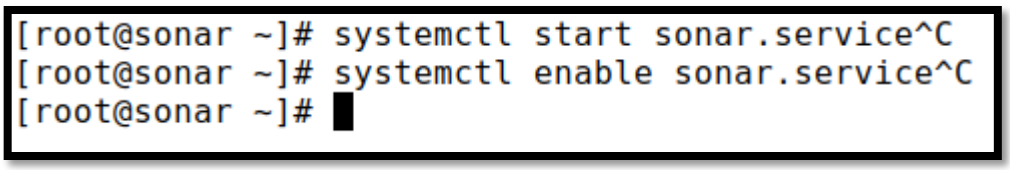

Figura 31. Instalando Sonarqube5.
Fuente: (Elaboración propia, 2020)

```
[root@sonar ~]# cat /etc/systemd/system/sonar.service

[Unit]
Description=Servicio que gestiona SonarQube
After=postgresql.service

[Service]
Type=forking
User=sonarqube
Group=sonarqube
ExecStart=/opt/sonarqube/bin/linux-x86-64/sonar.sh start
ExecStop=/opt/sonarqube/bin/linux-x86-64/sonar.sh stop
ExecReload=/opt/sonarqube/bin/linux-x86-64/sonar.sh restart
PIDFile=/opt/sonarqube/bin/linux-x86-64/SonarQube.pid
LimitNOFILE=65536
LimitNPROC=4096
Restart=always

[Install]
WantedBy=multi-user.target
[root@sonar ~]#
```

Figura 32. Servicios de systemd habilitados.
Fuente: (Elaboración propia, 2020)

También se instaló Nginx: Servicio para proxear el puerto 9000 al 443 y asignar un dominio estable sonar.upeu.pe, Figura 33.

```
[root@sonar ~]# dnf install nginx
Last metadata expiration check: 2:42:07 ago on Tue 27 Oct 2020 08:30:52 PM -05.
Package nginx-1:1.14.1-9.module_el8.0.0+184+e34fea82.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[root@sonar ~]# cat /etc/nginx/sites-available/sonar.upeu.pe.conf
server {
        listen 80;
        server_name sonar.upeu.pe;
        return 301 https://sonar.upeu.pe;
}

server {
    listen 443;
    server_name sonar.upeu.pe;

    ssl    on;
  ssl_certificate    /etc/pki/tls/upeu-ssl/upeu-pe.crt;
  ssl_certificate_key    /etc/pki/tls/upeu-ssl/upeu-pe.key;


  client_max_body_size 200M;


    location / {
      proxy_set_header        Host $host:$server_port;
      proxy_set_header        X-Real-IP $remote_addr;
      proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header        X-Forwarded-Proto $scheme;

      # Fix the "It appears that your reverse proxy set up is broken" error.
      proxy_pass          http://127.0.0.1:9000;
      proxy_read_timeout  90;

      proxy_redirect      http://127.0.0.1:9000 https://sonar.upeu.pe;

      # Required for new HTTP-based CLI
      proxy_http_version 1.1;
      proxy_request_buffering off;
      # workaround for https://issues.jenkins-ci.org/browse/JENKINS-45651
      add_header 'X-SSH-Endpoint' 'sonar.upeu.pe' always;
    }
  }
```

Figura 33. Instalando Nginx para proxeo de puertos.
Fuente: (Elaboración propia, 2020)

También se habilitaron los puertos 80 y 443, Figura 34.

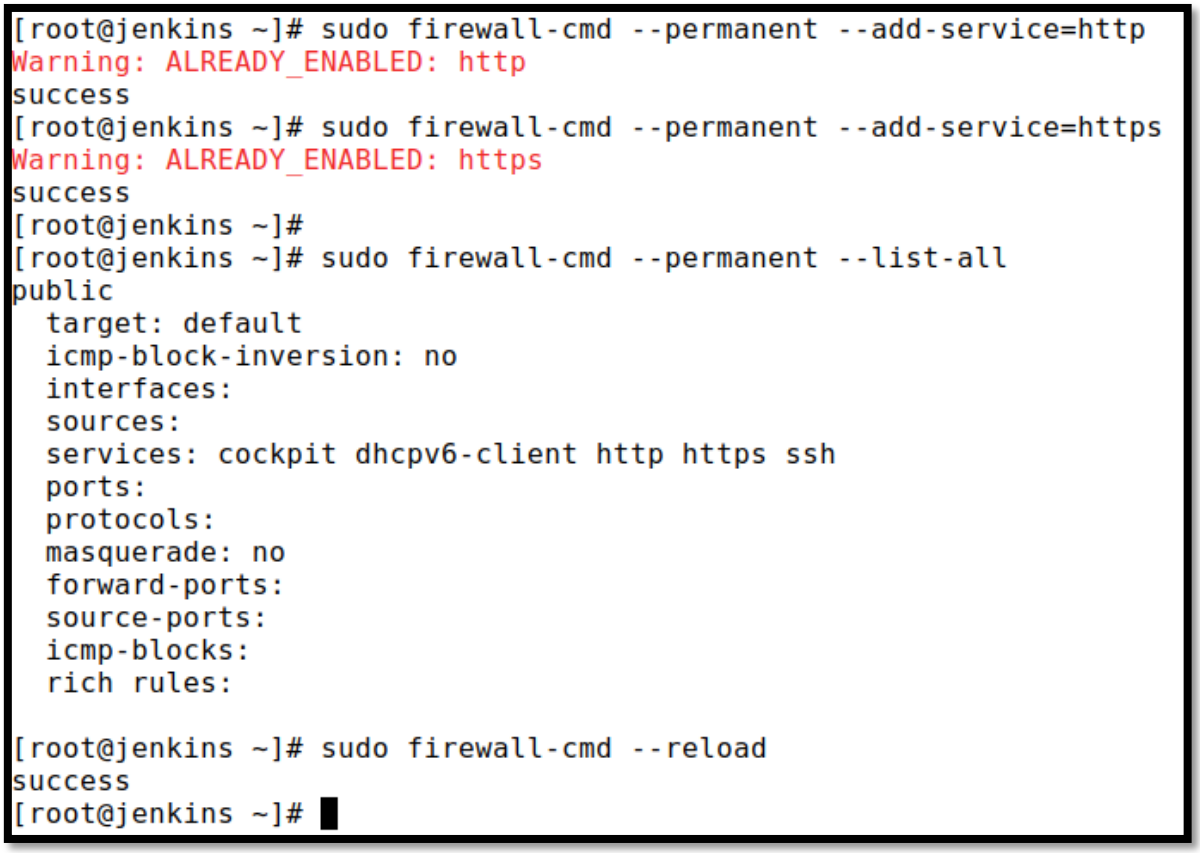

Figura 34. Habilitando puertos 80 y 443.
Fuente: (Elaboración propia, 2020)

Como sabemos que un ambiente se refiere a hardware y software en donde se ejecutara la aplicación, entonces para esta etapa se realizó la instancia de la siguiente manera:

Se inició instalando en un servidor el S.O. Centos 8 con las siguientes características, Figura 35 y Figura 36.

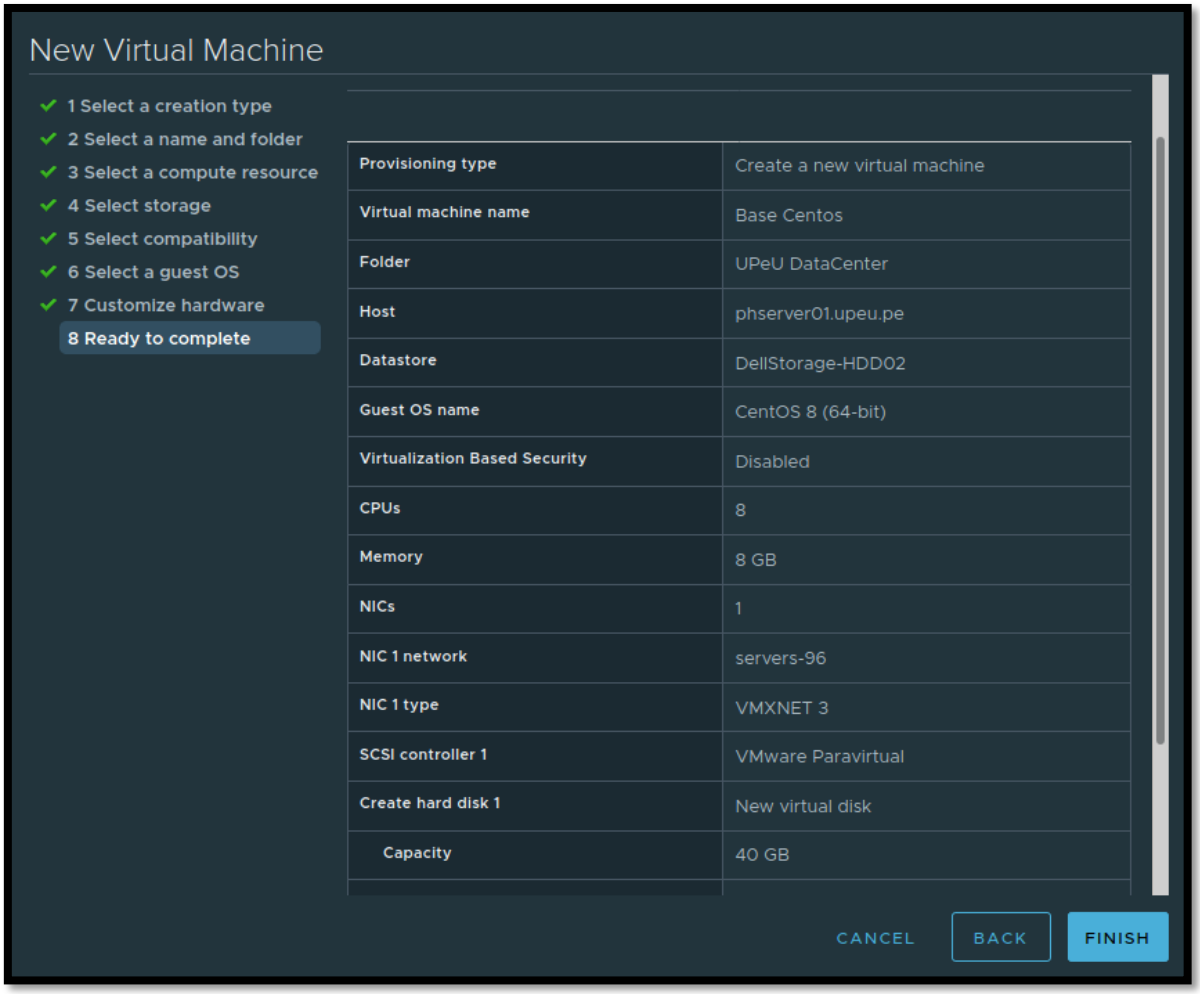

Figura 35. Instalación Centos 8.
Fuente: (Elaboración propia, 2020)

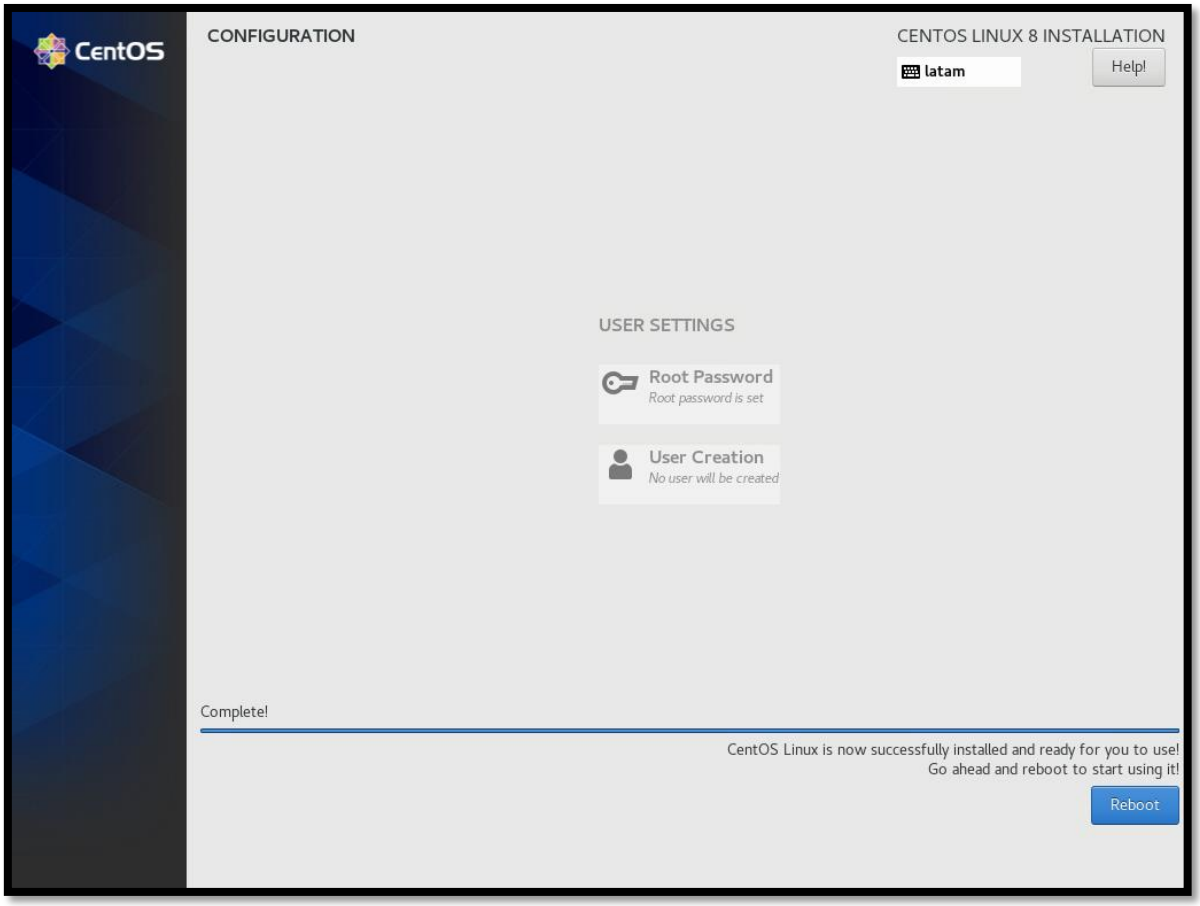

Figura 36. Centos 8 instalado.
Fuente: (Elaboración propia, 2020)

Para luego instalar Java y usar Jenkins como fuente para el despliegue, de la misma manera se seteo las variables de entorno de Java_Home, Figura 37 y Figura 38.

```
[root@jenkins ~]# sudo wget -O /etc/yum.repos.d/jenkins.repo \
>         https://pkg.jenkins.io/redhat/jenkins.repo^C
[root@jenkins ~]# sudo rpm --import https://pkg.jenkins.io/redhat/jenkins.io.key^C
[root@jenkins ~]# sudo yum upgrade^C
[root@jenkins ~]# sudo yum install jenkins java-1.8.0-openjdk-devel^C
[root@jenkins ~]# sudo systemctl daemon-reload^C
[root@jenkins ~]#
```

Figura 37. Instalando Jenkins.
Fuente: (Elaboración propia, 2020)

```
[root@jenkins ~]# cat /etc/profile.d/java.sh
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.x86_64
export PATH=$PATH:$JAVA_HOME/binexport
export JRE_HOME=/usr/lib/jvm/jre
export CLASSPATH=.:$JAVA_HOME/jre/lib:$JAVA_HOME/lib:$JAVA_HOME/lib/tools.jar
[root@jenkins ~]#
```

Figura 38. Seteando variables de Java_Home.
Fuente: (Elaboración propia, 2020)

Posteriormente se generó las credenciales ssh-key para poder acceder a los servidores de despliegue y acceso al git, Figura 39.

Figura 39. Git llaves SSH.
Fuente: (Elaboración propia, 2020)

Luego se instaló el servicio web Nginx para realizar el proxeo del puerto 8080 al 443 también se le asignó un dominio con el nombre de Jenkins.upeu.pe, Figura 40 y Figura 41.

```
[root@jenkins ~]# dnf install nginx
Last metadata expiration check: 0:47:18 ago on Tue 27 Oct 2020 01:31:42 PM -05.
Package nginx-1:1.14.1-9.module_el8.0.0+184+e34fea82.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[root@jenkins ~]# cat /etc/nginx/
```

Figura 40. Instalando Nginx.
Fuente: (Elaboración propia, 2020)

```
[root@registry sites-available]# ls
registry.upeu.conf
[root@registry sites-available]# cat registry.upeu.conf
server {
        listen 80;
        server_name registry.upeu.pe;
        return 301 https://registry.upeu.pe;
}

server {
    listen 443;
    server_name sonar.upeu.pe;

    ssl    on;
  ssl_certificate    /etc/pki/tls/upeu-ssl/upeu-pe.crt;
  ssl_certificate_key    /etc/pki/tls/upeu-ssl/upeu-pe.key;


  client_max_body_size 1024M;


    location / {

             # Do not allow connections from docker 1.5 and earlier
    # docker pre-1.6.0 did not properly set the user agent on ping, catch "Go *" user agents
    if ($http_user_agent ~ "^(docker\/1\.(3|4|5(?!\.[0-9]-dev))|Go ).*$" ) {
      return 404;
    }

    proxy_pass                          http://127.0.0.1:5000;
    proxy_set_header  Host              $http_host;   # required for docker client's sake
    proxy_set_header  X-Real-IP         $remote_addr; # pass on real client's IP
    proxy_set_header  X-Forwarded-For   $proxy_add_x_forwarded_for;
    proxy_set_header  X-Forwarded-Proto $scheme;
    proxy_read_timeout                  900;
    }
  }

[root@registry sites-available]#
```

Figura 41. Proxeo del puerto 8080 al 443.
Fuente: (Elaboración propia, 2020)

Luego se instaló el certificado SSL para también habilitar el puerto 80 y 443, Figura 42.

```
[root@jenkins ~]# cat /etc/nginx/sites-available/jenkins.upeu.pe-ssl.conf
server {
        listen 80;
        server_name jenkins.upeu.pe;
        return 301 https://jenkins.upeu.pe;
}

server {
    listen 443;
    server_name jenkins.upeu.pe;

    ssl    on;
  ssl_certificate    /etc/pki/tls/upeu-ssl/upeu-pe.crt;
  ssl_certificate_key    /etc/pki/tls/upeu-ssl/upeu-pe.key;

    location / {
      proxy_set_header        Host $host:$server_port;
      proxy_set_header        X-Real-IP $remote_addr;
      proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header        X-Forwarded-Proto $scheme;

      # Fix the "It appears that your reverse proxy set up is broken" error.
      proxy_pass          http://127.0.0.1:8080;
      proxy_read_timeout  90;

      proxy_redirect      http://127.0.0.1:8080 https://jenkins.upeu.pe;

      # Required for new HTTP-based CLI
      proxy_http_version 1.1;
      proxy_request_buffering off;
      # workaround for https://issues.jenkins-ci.org/browse/JENKINS-45651
      add_header 'X-SSH-Endpoint' 'jenkins.upeu.pe' always;
    }
  }

[root@jenkins ~]#
```

Figura 42. Instalando certificado SSL y habilitando puerto 80 y 443.
Fuente: (Elaboración propia, 2020)

Para tener una mejor manera de instanciar los ambientes de prueba se instalaron los plugins como Pipeline, BlueOcen, Docker, Git, Sonar, entre otros, Figura 43, Figura 44, Figura 45, Figura 46 y Figura 47.

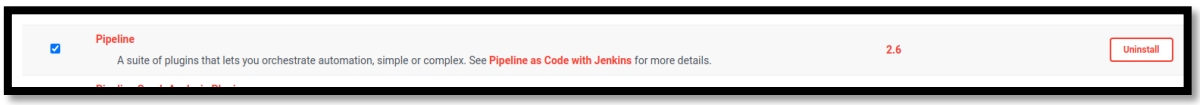

Figura 43. Instalando plugin Pipeline.
Fuente: (Elaboración propia, 2020)

Figura 44. Instalando plugin Blue Ocean.
Fuente: (Elaboración propia, 2020)

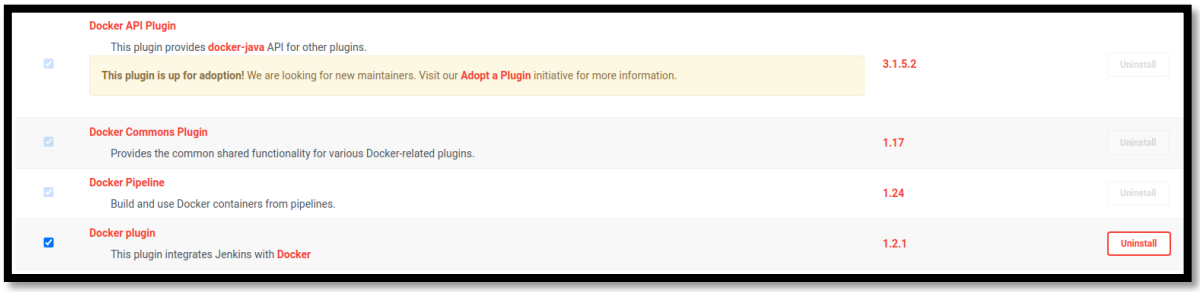

Figura 45. Instalando plugin docker.
Fuente: (Elaboración propia, 2020)

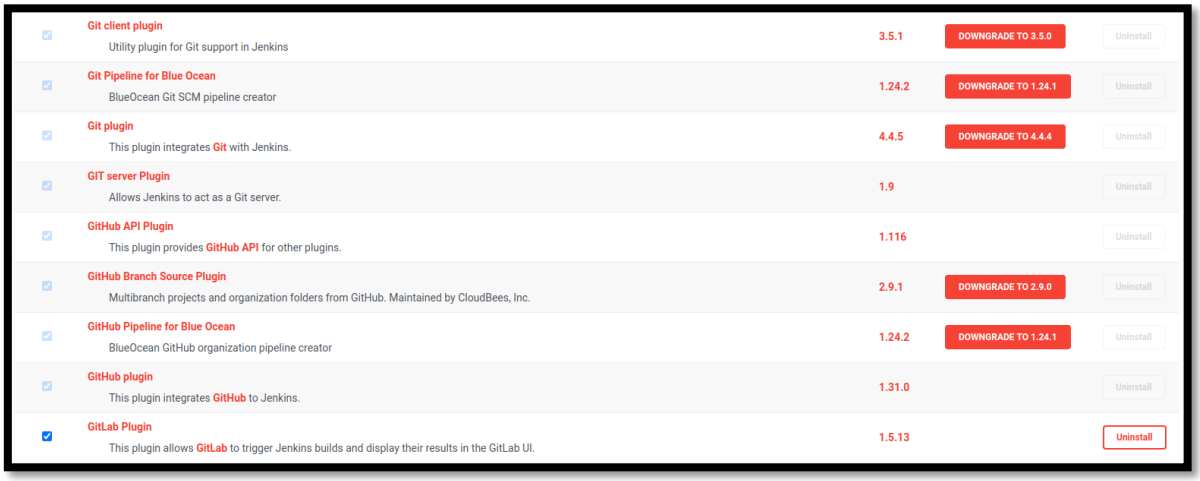

Figura 46. Instalando plugin Git.
Fuente: (Elaboración propia, 2020)

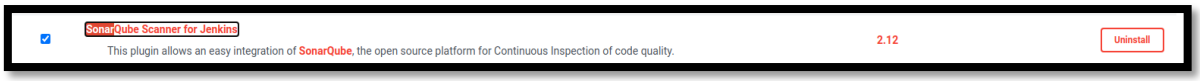

Figura 47. Instalando plugin Sonar.
Fuente: (Elaboración propia, 2020)

Se crearon las imágenes con las llaves y librerías del sistema necesario para el despliegue de las aplicaciones, Figura 48.

```
FROM ubuntu
RUN apt update -y && apt upgrade -y &&  apt install build-essential unzip python-dev libaio-dev python3 python3-pip -y
COPY ./lib_external_system/instantclient_11_2 /opt/instantclient_11_2/
RUN ln -s /opt/instantclient_11_2/libclntsh.so.11.1 /opt/instantclient_11_2/libclntsh.so
ENV ORACLE_HOME=/opt/instantclient_11_2/
ENV LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME
RUN echo "export ORACLE_HOME=/opt/instantclient_11_2/" >  /etc/profile.d/oracle.sh
RUN echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/instantclient_11_2/" >>  /etc/profile.d/oracle.sh
RUN pip3 install cx_oracle==5.3
```

Figura 48. Creando imágenes y llaves para el despliegue del sistema.
Fuente: (Elaboración propia, 2020)

También se cargó las imágenes al docker registry, Figura 49.

Figura 49. Docker Registry.
Fuente: (Elaboración propia, 2020)

Se crearon los docker files que llamaran a las imágenes y ejecutaran el código de los desarrolladores, Figura 50 y Figura 51.

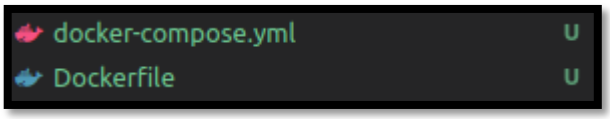

Figura 50. Docker file 1.
Fuente: (Elaboración propia, 2020)

```
Dockerfile > ...
1   FROM registry.upeu.pe:django-oracle
2   LABEL maintainer="Pacifi"
3
4   ENV PYTHONUNBUFFERED=1
5   RUN mkdir /code
6   WORKDIR /code
7   COPY requirements.txt /code/
8   RUN pip install -r requirements.txt
9   COPY . /code/
10  EXPOSE 8000
11  CMD exec gunicorn config.wsgi:application --bind 0.0.0.0:8000 --workers 3
```

Figura 51. Docker file 2.
Fuente: (Elaboración propia, 2020)

Se crearon los jenkinsfiles archivos que ejecutan las tareas, Figura 52 y Figura 53.

Figura 52. Jenkinsfile 1.
Fuente: (Elaboración propia, 2020)

```
pipeline {
 agent any
 stages {


       stage('Checkout-git develop'){
          when {
               branch 'develop'
           }
              steps{
                   git branch: 'develop', url: 'git@git.upeu.pe:DEV_2019/acuerdos_backend.git'
               }
       }

       stage('Checkout-git Master'){
          when {
               branch 'master'
           }
              steps{
                   git branch: 'master', url: 'git@git.upeu.pe:DEV_2019/acuerdos_backend.git'
               }
       }


      stage('Despliegue Develop') {
          when {
              branch 'develop'
          }
          steps {
              echo 'preparando despliegue en el entorno desarollo'
              sh 'ssh desarrollo@192.168.96.84 "cd /home/desarrollo/deploy/acuerdos_back/proyecto && git pull origin develop"'
              sh 'ssh desarrollo@192.168.96.84 "source /home/desarrollo/deploy/acuerdos_back/bin/activate && pip install -r /home/desarrollo/deploy/acuerdos_ba
              sh 'ssh devops@192.168.96.84 sudo supervisorctl restart all'
              sh 'ssh devops@192.168.96.84 sudo systemctl restart nginx'
          }
       }

       stage('Sonar Testing') {
          when {
              branch 'develop's
          }
          steps {
              echo 'Sonar Cuality'
              sh 'docker run --rm --net host  -e SONAR_HOST_URL=https://sonar.upeu.pe  -v $(pwd):/usr/src sonarsource/sonar-scanner-cli sonar-scanner -Dsonar.
          }
       }
```

Figura 53. Jenkinsfile 2.
Fuente: (Elaboración propia, 2020)

Se configuraron los proyectos en jenkins.upeu.pe para ver las pasarelas, Figura 54.

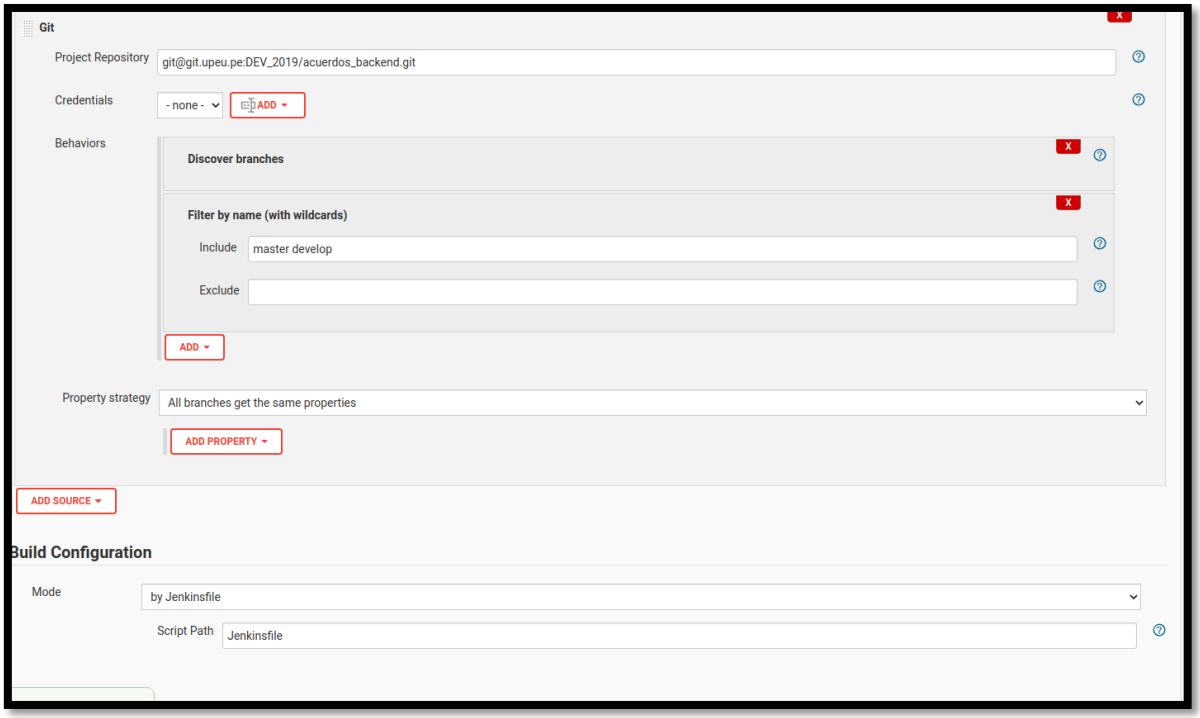

Figura 54. Configuración de proyectos en Jenkins.
Fuente: (Elaboración propia, 2020)

Se agregó la configuración de build stat a los repositorios Readme, Figura 55, Figura 56, Figura 57.

Figura 55. Configuración de Buildstat 1.
Fuente: (Elaboración propia, 2020)

```
.. |flat-develop| image:: https://jenkins.upeu.pe/buildStatus/icon?job=Acuerdos%2FAcuerdos+Backend%2Fdevelop
   :alt: build status
   :scale: 100%
   :target: https://jenkins.upeu.pe/blue/organizations/jenkins/Acuerdos%2FAcuerdos%20Backend/activity

.. |flat-master| image:: https://jenkins.upeu.pe/buildStatus/icon?job=Acuerdos%2FAcuerdos+Backend%2Fmaster
   :alt: build status
   :scale: 100%
   :target: https://jenkins.upeu.pe/blue/organizations/jenkins/Acuerdos%2FAcuerdos%20Backend/activity
```

Figura 56. Configuración de Buildstat 2.
Fuente: (Elaboración propia, 2020)

```
Estado de Tubería
~~~~~~~~~~~~~~~~~

+---------------+---------------------+
| Branch        | Status              |
+===============+=====================+
| Develop       | |flat-develop|      |
+---------------+---------------------+
| Master        | |flat-master|       |
+---------------+---------------------+
```

Figura 57. Configuración de Buildstat 3.
Fuente: (Elaboración propia, 2020)

Se configuro las llaves de acceso y tipo de proyectos en sonar.upeu.pe, Figura 58.

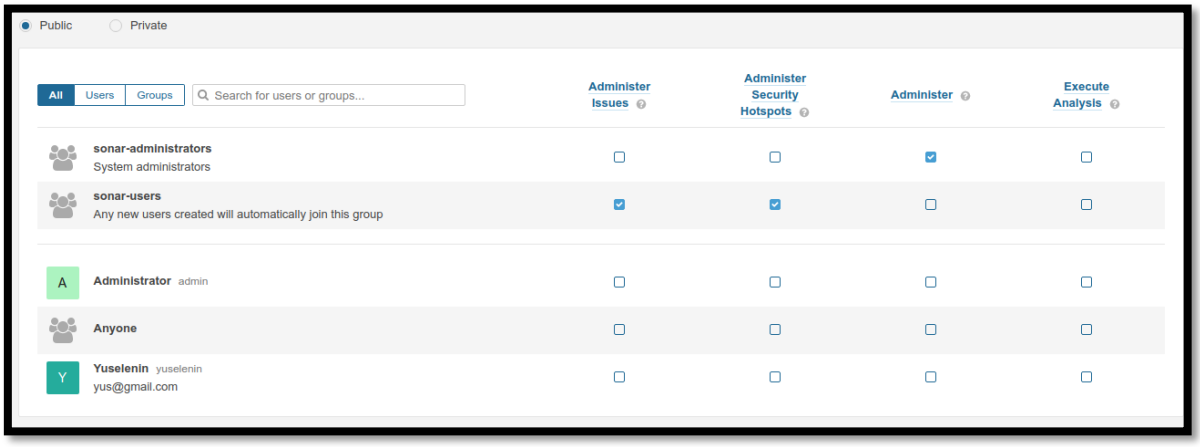

Figura 58. Configurando llaves de acceso.
Fuente: (Elaboración propia, 2020)

Se agregó la tarea de ejecución en docker para el análisis de código, Figura 59.

Figura 59. Tarea de ejecución en docker.
Fuente: (Elaboración propia, 2020)

3.4.1.3. Verificar.

3.4.1.3.1. Indicador de usabilidad.

Para poder medir la usabilidad con el nuevo diagrama de despliegue de aplicaciones que propusimos y según el estándar ISO/IEC 25010 que nos permitió medir la usabilidad mediante el aprendizaje, protección frente a errores de usuario y accesibilidad. Para el caso de la implementación de un ambiente de integración continua podemos ver en la Figura 60, de cómo se manejaba anteriormente, vimos que si un equipo quiere empezar con un proyecto este tendría que elegir el sistema operativo y configurar su escritorio de acuerdo a como a él le parezca mejor y pasara lo mismo con todos los miembros del equipo, esto conlleva al estar en una etapa avanzada del proyecto tener algunos conflictos quizás por versiones de sistema operativo u otra configuración, esto se solucionó al implementar un cambien de integración CI/CD, como podemos ver en la Figura 67, se mejoró en este aspecto ya que ahora se trabaja con contenedores y algunos requerimientos no funcionales ya están definidos el cual ayuda al grupo a mejorar al momento de desplegar las aplicaciones.

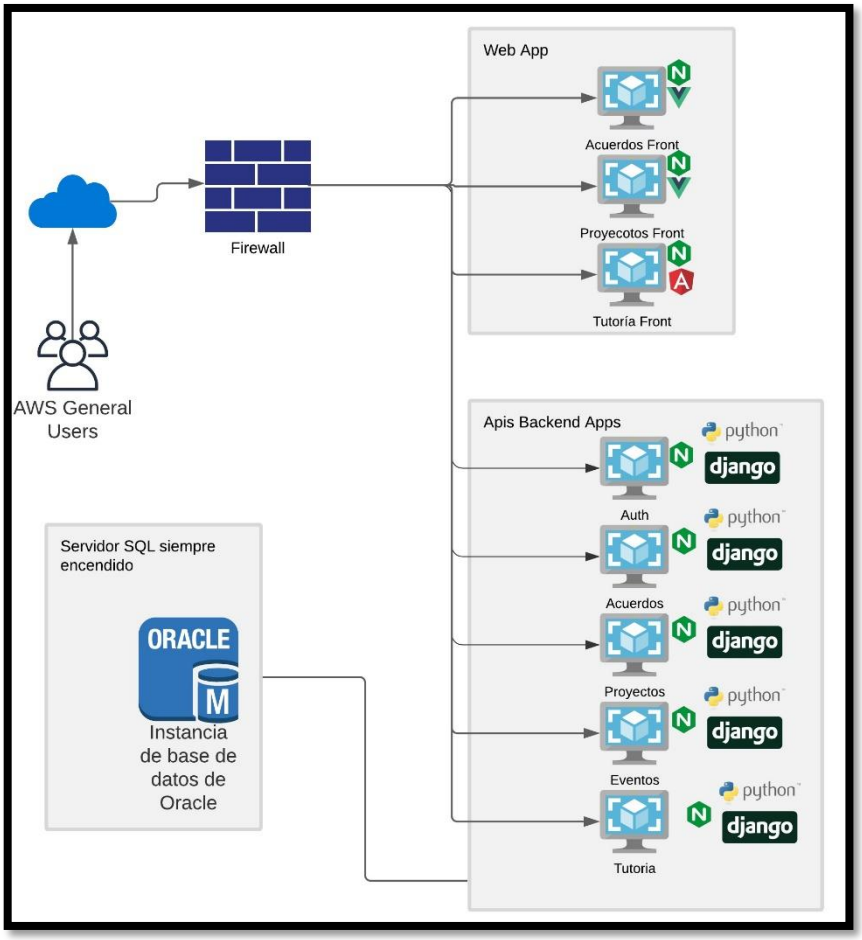

Figura 60. Diagrama de despliegue analizado.
Fuente: (Elaboración propia, 2020)

Este diagrama la infraestructura usada para el lanzamiento de aplicaciones usando Django, el cual consta de una aplicación por servidor y una conexión directa a base de datos.

3.4.1.3.2. Indicador de calidad.

Para el indicador de calidad se mide la calidad de software con Sonar en la Figura 61, se ve el reporte que Sonar nos provee con respecto a la calidad de software, se ve que en fiabilidad no se encontró ningún fallo, en la parte de seguridad vemos que se encontró que los host son seguros y que no se hay vulnerabilidad y para mantenibilidad se ve que hay código que aún se puede mejorar, también vemos en la Figura 62, que tenemos tres apartados de Errores de código, vulnerabilidades y código bien desarrollado, se puede observar que no hay errores de código que no hay vulnerabilidades pero vemos que se puede mejorar en cuanto al desarrollo del código, la misma explicación para la Figura 63.

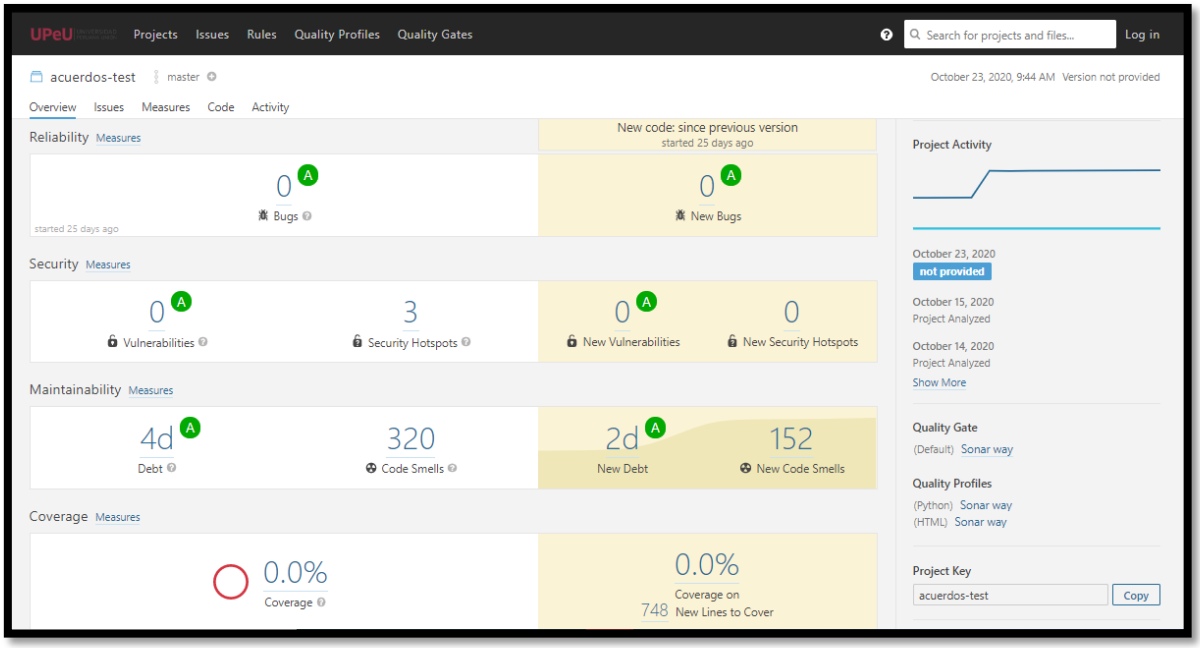

Figura 61. Reporte de la calidad del software.
Fuente: (Elaboración propia, 2020)

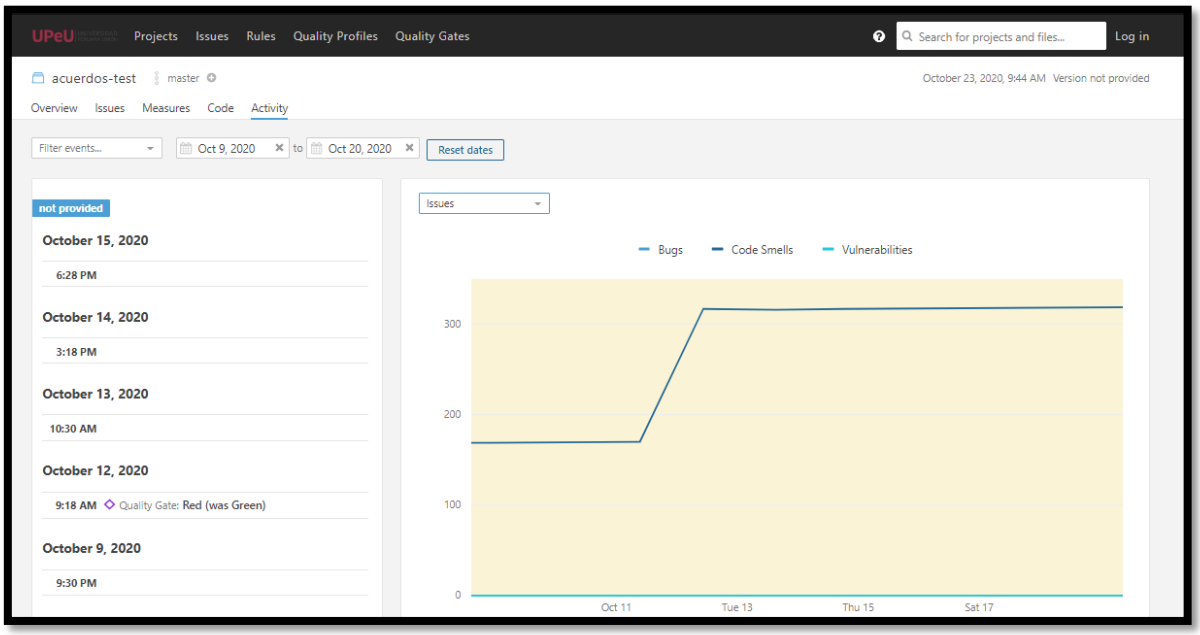

Figura 62. Actividad del software subido.
Fuente: (Elaboración propia, 2020)

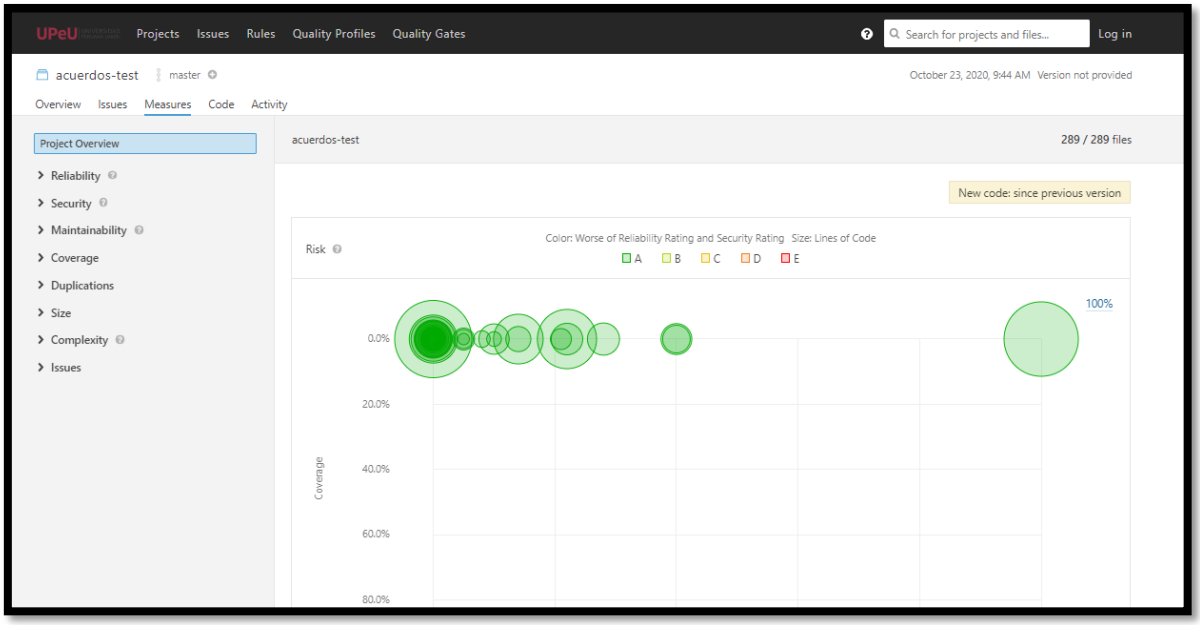

Figura 63. Reportes de las medidas del software subido.
Fuente: (Elaboración propia, 2020)

3.4.1.3.3. Indicador de eficiencia de desempeño.

Para el caso práctico desarrollado y poder medir su eficiencia mediante el comportamiento de tiempos y recursos según la ISO/IEC 25010, una vez implementado el sistema solo se consideró los tiempos de respuesta, en el ambiente de desarrollo, para lo cual, se tomó 2 conjuntos de datos Desarrollador y Centro de Datos, cada uno representan el proceso de despliegue de una aplicación. El primer conjunto mediante el proceso con el sistema antiguo de despliegue y el segundo con la utilización del nuevo proceso de despliegue con CI/CD. En la Figura 10, se detalla la interacción del proceso anterior. En la Figura 64, podemos visualizar el nuevo proceso propuesto, en la Tabla 5 podemos ver la representación del comportamiento en cuanto a tiempos y recursos del antes y después de la integración del CI/CD.

Tabla 5. Comportamiento de tiempos y recursos.

	Proceso de despliegue sin CI/CD	Tiempo estimado	Proceso de despliegue con CI/CD	Tiempo estimado
Desarrollador	• Solicitud de lanzamiento. • Pruebas de funcionalidad. • Solicitar lanzamiento a producción.	120 min	• Solicitar cambios en el servidor • Pruebas de funcionalidad • Realizar cambios de producción	30 min

Centro de datos	• Recepción de requerimientos. • Cambios en el servidor. • Desplegar cambios en el servidor de pruebas. • Verificación de dependencias. • Enviar informe de fallos. • Desplegar servidor de producción.	180 min	• Jenkins • Sonar • Git	3 min

Fuente: (Elaboración propia, 2020)

Tabla 6. Tiempos totales para desplegar aplicaciones según petición de cambios

	Tiempo Manual	Tiempo Post Ambiente
Preuba de despliegue	300	33
Lanzamiento a producción	150	20
Lanzamiento de Servidor pruebas	200	10
Lanzamiento a producción Frontend	120	10
Revisión de Errores	120	15
Pase a producción y empaquetado de librerías	180	20
Resolución de conflictos	65	10
Solución de problemas por falta de documentación de librerías	30	5

Fuente: (Elaboración propia, 2020)

Tabla 7. Prueba T para medias de 2 muestras emparejadas

	Tiempo Manual	*Tiempo Post Ambiente*
Media	145.625	15.375
Varianza	7024.55357	78.26785714
Observaciones	8	8
Grados de libertad	7	
Estadístico t	4.81702902	
P Value	0.00096402	
Valor crítico de t	1.89457861	

Fuente: (Elaboración propia, 2020)

Como es un conjunto de datos menor a 30 y provienen de una distribución normal, se aplica la prueba T-Student para muestras relacionadas, considerando el tamaño de la muestra (n): 8, grados de liberta (n-1): 7 y el nivel de la significancia del 5%, $\propto = 0.05$ se obtuvo un p-value igual a 0.00096402. Donde los resultados obtenidos se tiene un intervalo de confianza de 1.89457861 que determina la aceptación o rechazo de una hipótesis nula (H0), dado a que el valor cae en la zona de rechazo se concluye que la implementación del sistema se obtiene una diferencia de mejora significativa en cuanto a la eficiencia en los tiempos en el proceso de lanzar, monitorear, resolver, entregar aplicaciones. Adema, con un T calculado al 4.81 siendo mayor al intervalo de confianza y obteniendo una probabilidad de p-valor = 0.00096402, el cual es menor al nivel de significancia ($\propto = 0.05$).

3.4.1.4. Actuar.

Como los resultados fueron satisfactorios se implanto los ambientes de integración y despliegue continuo para las aplicaciones. Una vez finalizada e implantada la mejora, las actividades en el área de desarrollo de la Universidad Peruana Unión Campus Juliaca, funcionaron más eficientemente. No obstante, periódicamente se volverá a buscar nuevas mejoras y volver a aplicar el círculo de Demming. En la Figura 64, podemos ver como quedo el nuevo proceso para la mejora en la integración continua de las aplicaciones desarrolladas.

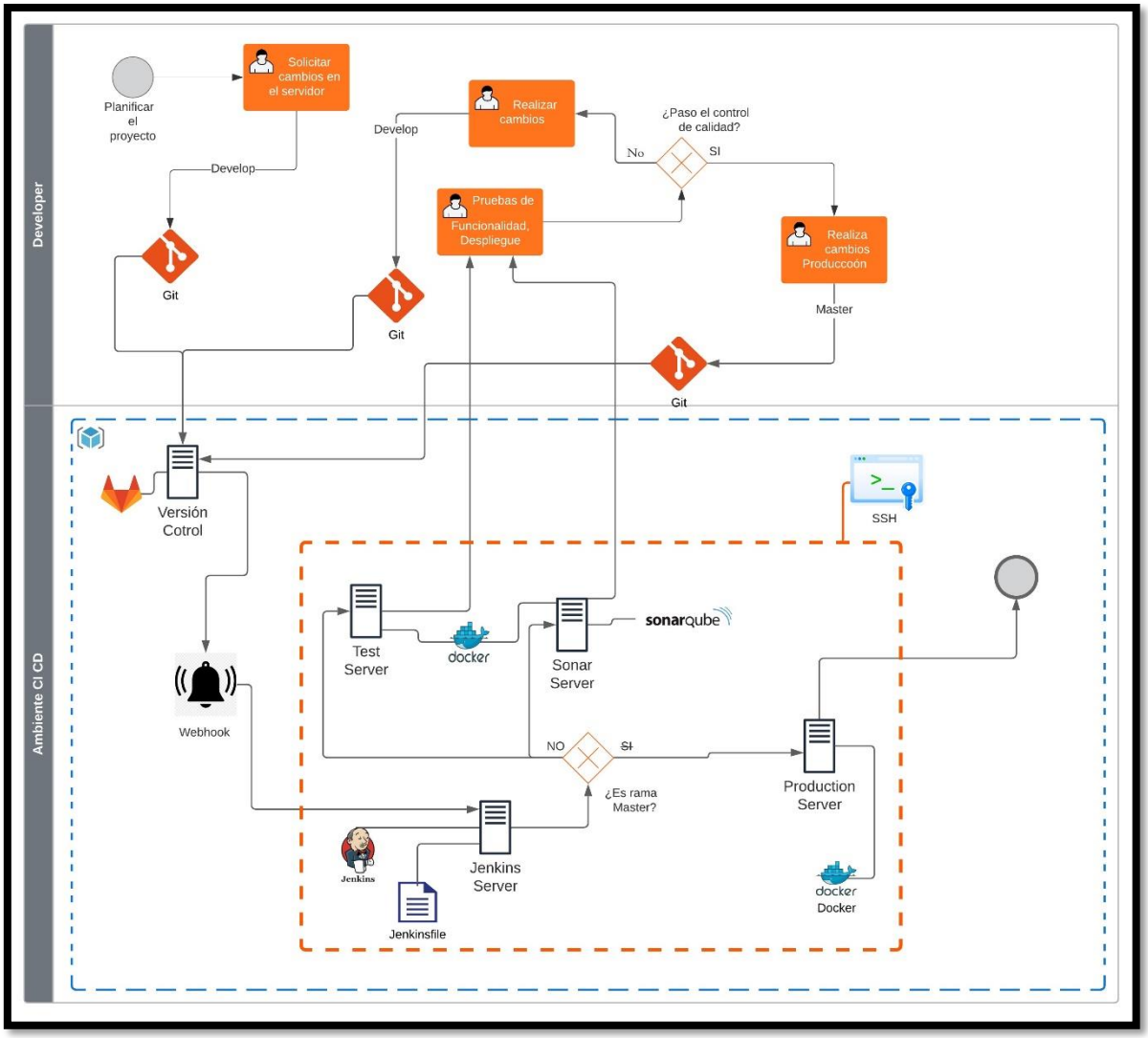

Figura 64. Proceso de Despliegue propuesto.
Fuente: (Elaboración propia, 2020)

El nuevo proceso muestra como gran parte de los tareas que se realizaban por parte del equipo del centro de datos de forma manual pasan a ser tareas automatizadas dentro del proceso de lanzamiento de aplicaciones.

CAPÍTULO IV. Resultados y discusión

4.1. Resultado general - Desarrollar un ambiente de construcción de aplicaciones empresariales del centro de datos de la Universidad Peruana Unión Filial Juliaca.

Se logra realizar la implementación de los ambientes para la construcción de aplicaciones empresariales bajo una cultura de devops y una infraestructura basada en herramientas que eliminen pasos manuales para la reducción de errores y a la ves un realizar escalamiento de equipos pequeños y aislados, el ambiente está basado como un hito inicial de seguir la cultura de devops la cual realiza el despliegue continuo con Jenkins y a la vez realizar el minitoreo de dicho despliegue bajo una infraestructura unificada.

4.2. Resultado 1 - Analizar y diseñar el ambiente de desarrollo.

Para la etapa de planificación se analizó y diseñó los ambientes de desarrollo ya mencionados, en la Figura 65, podemos ver el diagrama antes de la propuesta. Vemos que el ambiente de desarrollo no es muy óptimo ya que para cada proyecto se necesita realizar nuevamente los requerimientos no funcionales, así como la configuración, instalación de dependencias y variedad de servicios por cada desarrollador. En la Figura 66, podemos ver la propuesta de la mejora para el ambiente de desarrollo utilizando contenedores para establecer algunas configuraciones generales y de esa manera el grupo pueda avanzar con el desarrollo de la aplicación con mayor eficiencia.

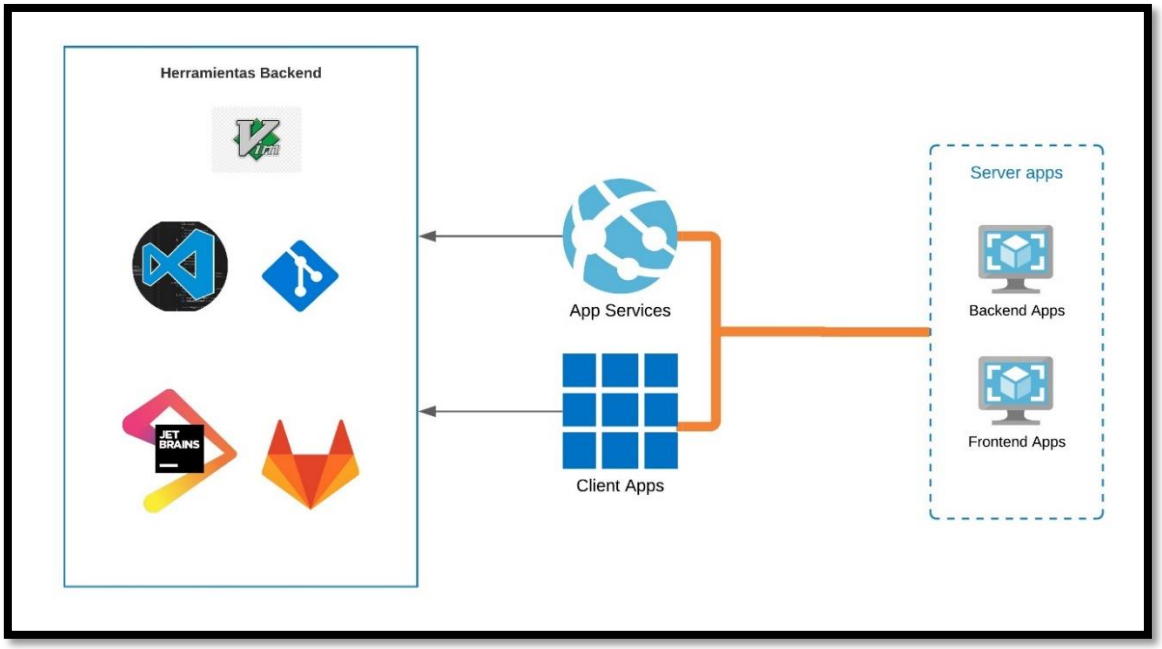

Figura 65. Diagrama anterior del ambiente de desarrollo.
Fuente: (Elaboración propia, 2020)

4.3. Resultado 2 - Seleccionar las herramientas adecuadas para mejorar el ambiente de desarrollo.

Para la seleccionar las herramientas adecuadas para esta investigación se tomo en cuenta la investigación de (Alejandro, 2017, pág. 106) en donde muestra el porque usar dichas herramientas open source y porque son buenas de la misma manera estando en la misma etapa de planificación se seleccionó las herramientas que se usaran para la mejora del despliegue de aplicaciones en la Figura 66 podemos ver que las herramientas seleccionadas fueron las siguientes Docker que trabaja como contenedor, Jenkins para automatizar las tareas, docker-registry para el almacenamiento de imágenes construidas por los desarrolladores.

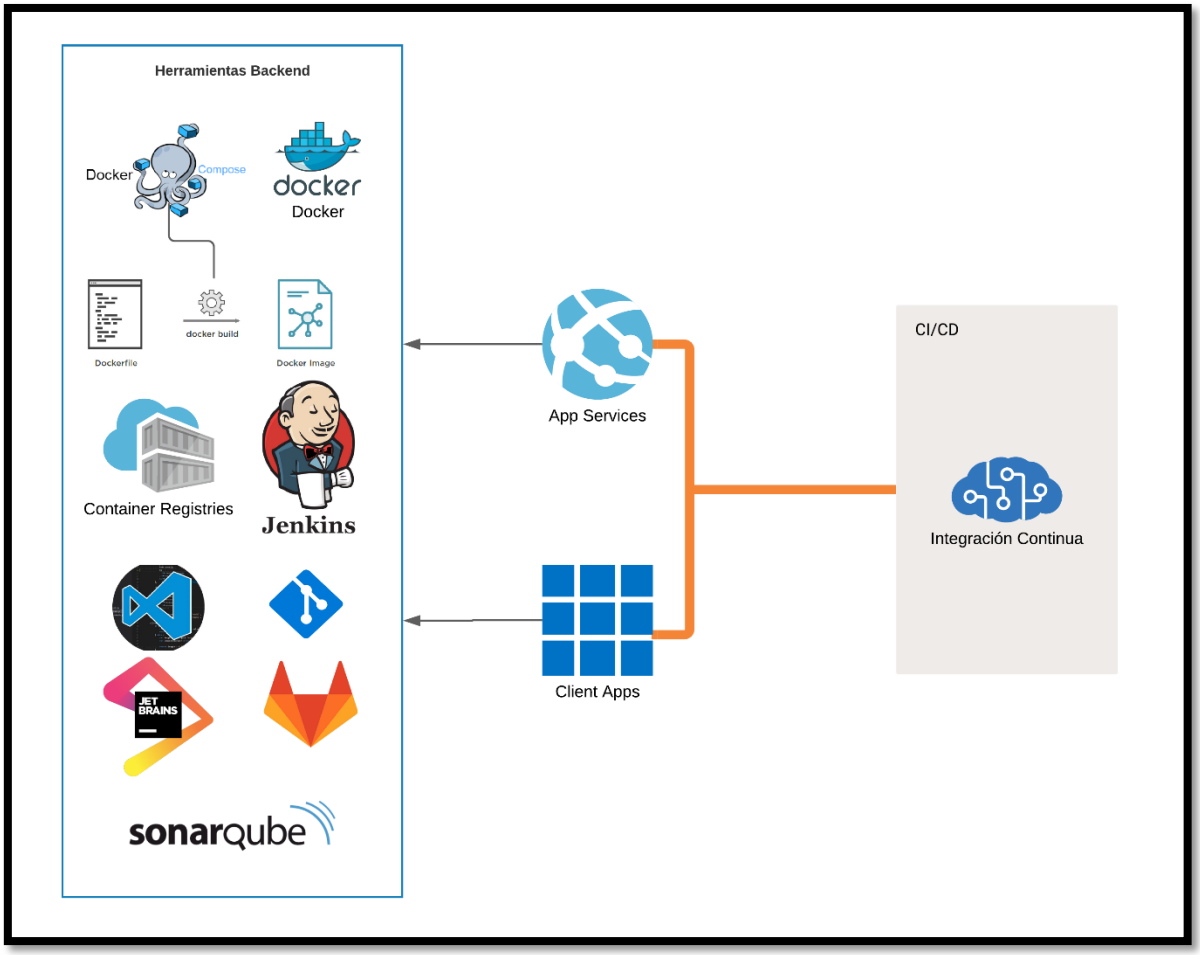

Figura 66. Herramientas para la mejora del ambiente de desarrollo.
Fuente: (Elaboración propia, 2020)

4.4. Resultado 3 - Establecer una nueva arquitectura de despliegue de servicios.

Para la etapa de implementación de acuerdo a nuestro tercer objetivo se estableció una nueva arquitectura de despliegue de servicios en el cual se instaló y un sistema de balenceo de carga y se modifico el sistema de despliegue de servidores a contenedorescomo, vimos y explicamos en la Figura 60, mostrando como era la arquitectura de despliegue antes de la implementación y las deficiencias que esta tenia, en la Figura 67 podemos ver la propuesta que trabaja con contenedores los cuales ya están definidos según los requerimientos del grupo de desarrollo mejorando el despliegue ya que cada miembro del grupo trabaja bajo un mismo entorno siendo este una replica del entorno de producción.

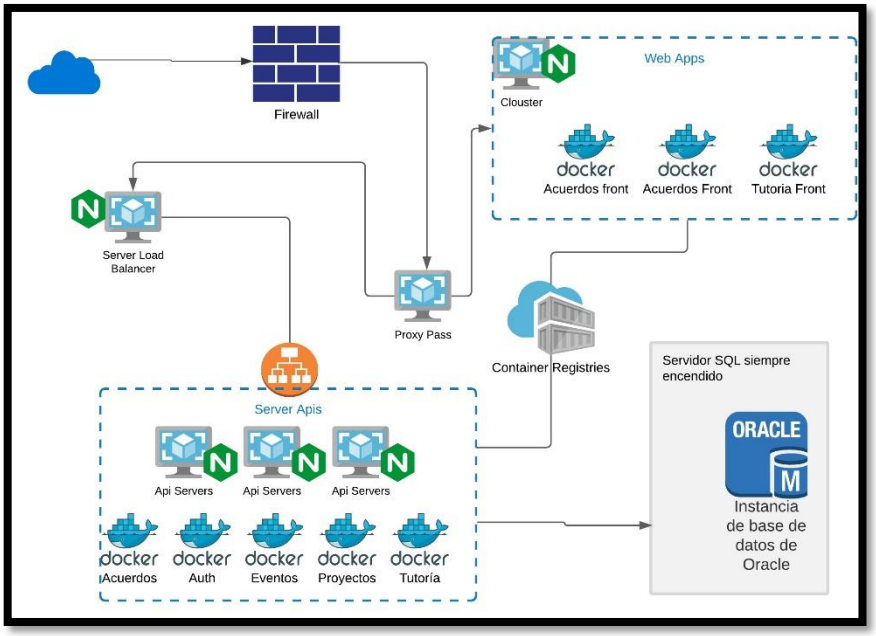

Figura 67. Diagrama de despliegue propuesto.
Fuente: (Elaboración propia, 2020)

La Figura 67 muestra un diagrama de despliegue basado en contenedores y llamadas a través de imágenes definidas en el servicio de docker-registry, así mismo muestra un sistema de balanceo de carga entre diferentes servidores las cuales mejoran el performance de solicitudes por parte del cliente.

4.5. Resultado 4 - Definir un ambiente de integración y despliegue continúo basado en la cultura DevOps.

Para nuestra etapa de verificación y de acuerdo a nuestro cuarto objetivo se definió un ambiente de integración y despliegue continúo basado en la cultura DevOps, en la Figura 68 vemos que cada vez que un programador haga realice una solicitud de push en el repositorio git remoto y suba código a la rama de desarrollo (develop), Jenkins lo detecte mediante un triger webhook, logrando así que0 compile la aplicación, llame a Sonar para ejecutar los test y realizar el análisis de la calidad del código y, si todo ha ido bien, se vuelve a realizar el push pero en la rama master el cual figurara el servidor de producción.

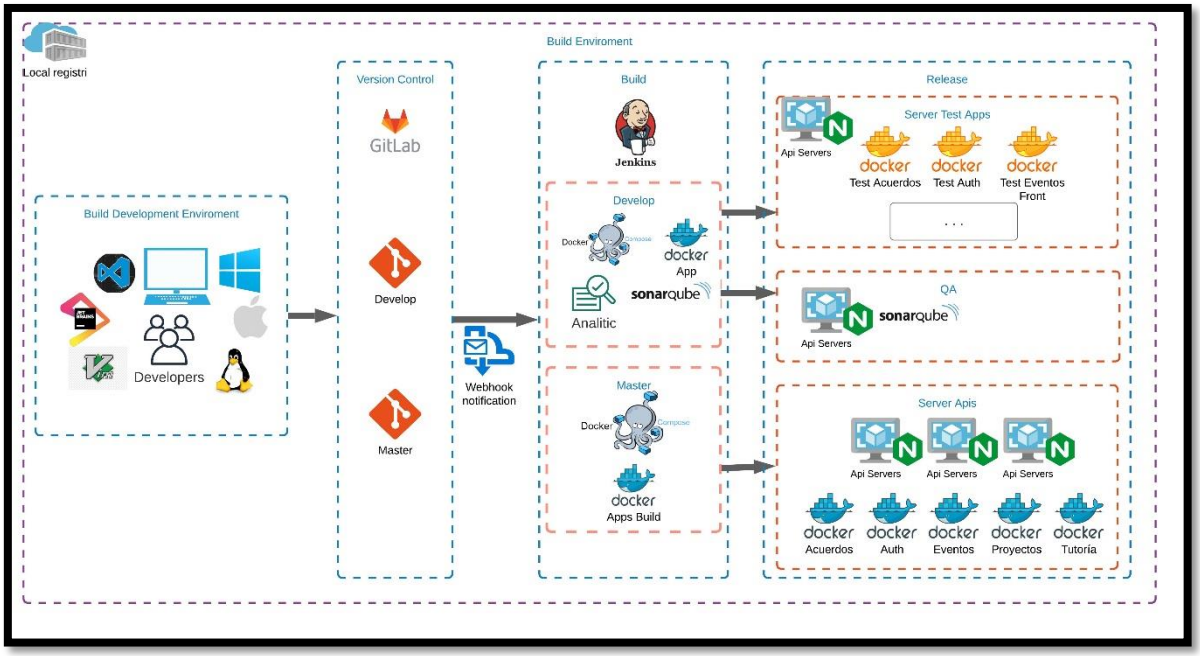

Figura 68. Arquitectura de CI/CD.
Fuente: (Elaboración propia, 2020)

4.6. Resultado 5 - Monitorear el despliegue de aplicaciones en Developement y Master.

Para la etapa de actuar y según a nuestro quinto objetivo podemos ver mediante el plugin BlueOcean como nos muestra el reporte de los test y análisis de calidad de código, en la Figura 69 podemos ver el reporte del test en develop, si este llega a estar correctamente los siguiente seria realizar un push pero en la rama master de la misma manera en la Figura 70 podemos ver el reporte del test en la rama master.

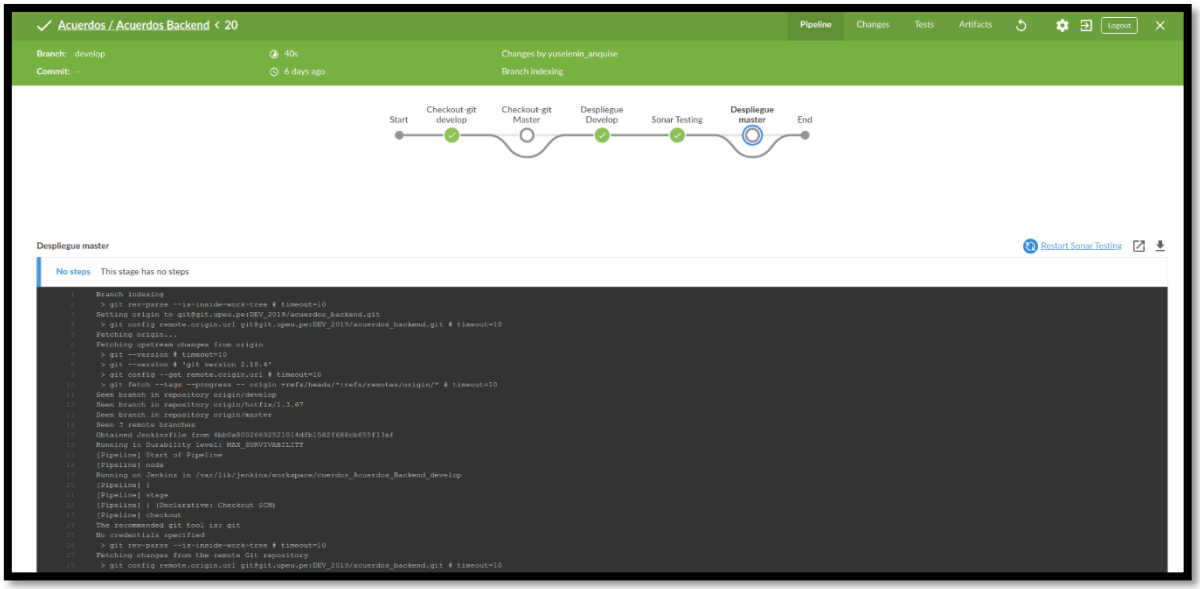

Figura 69. Reporte de despliegue de aplicaciones en Develop.
Fuente: (Elaboración propia, 2020)

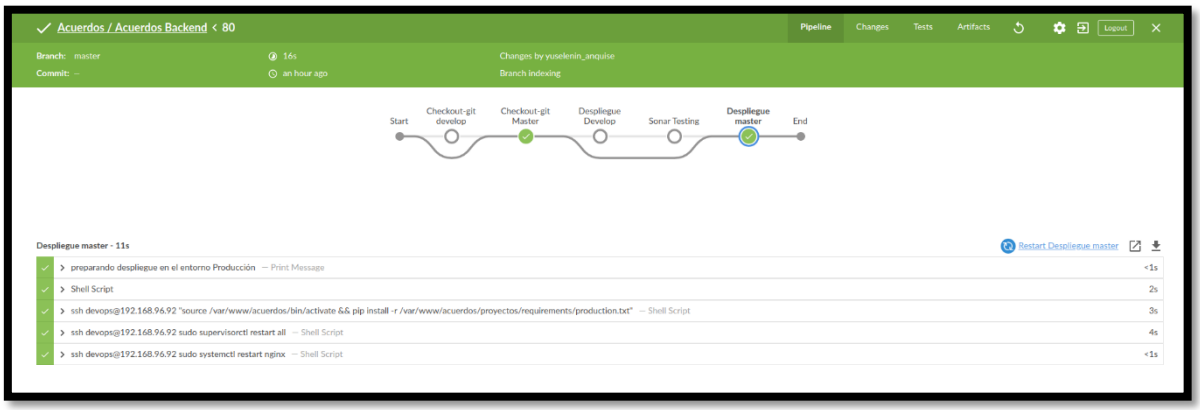

Figura 70. Reporte de despliegue de aplicaciones en Master.
Fuente: (Elaboración propia, 2020)

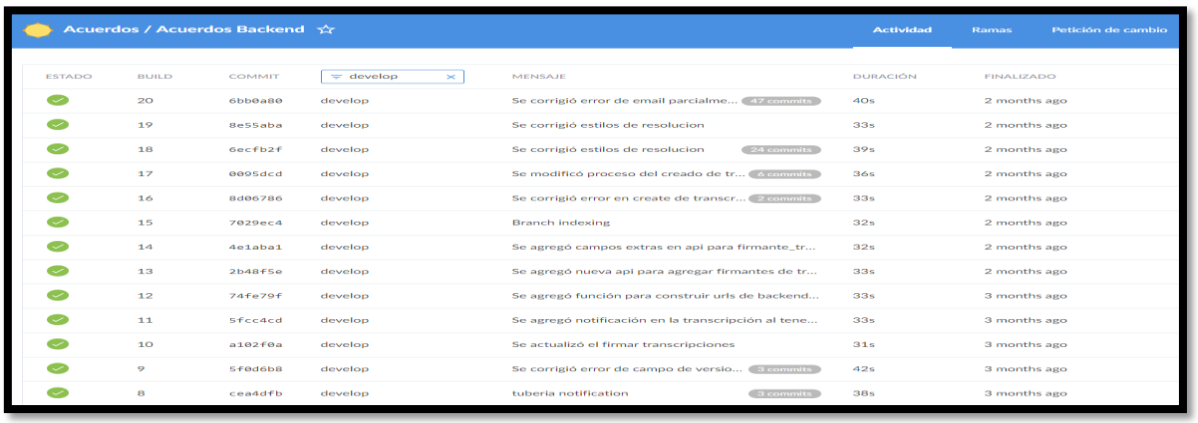

Figura 71 Reporte de despliegue por request.
Fuente: (Elaboración propia, 202*0)*

CAPÍTULO V. Conclusiones y recomendaciones.

5.1. Conclusiones.

En cuanto al objetivo general con la implementación del ambiente de construcción de aplicaciones se logra ver una mejora sustancial en el proceso de desarrollo realizando una prueba T-Student el cual nos muestra un valor t de 4.81702902 que está por encima del valor critico de 1.89487861 validando así la implementación en cuanto a la mejora del proceso de despligue.

En cuanto al primer objetivo específico se logró analizar el ambiente de desarrollo y el ciclo de vida del producto encontrando un divorcio entre el área desarrollo y el centro de datos los cuales limitan el tiempo de respuesta para un lanzamiento oportuno de aplicaciones.

Con respecto a nuestro segundo objetivo específico se logró seleccionar las herramientas adecuadas para la mejora del ciclo de vida de las aplicaciones las cuales ayudan a una implementación exitosa del ambiente de desarrollo.

En cuanto a nuestro tercer objetivo específico, una vez que se analizó y diseñó el ambiente de desarrollo se logra establecer una nueva arquitectura de despliegue de servicios que ayuda a realizar un mejor escalamiento en cuanto al lanzamiento de aplicaciones y performance.

Con respecto a nuestro cuarto objetivo específico logramos definir el ambiente de integración y despliegue continuo con la cultura DevOps, al ya tener seleccionado las herramientas adecuadas logramos implementar dichas herramientas para mejorar el despliegue de servicios, así como el monitoreo de las mismas.

En cuanto a nuestro quinto objetivo específico se logró monitorear el despliegue de aplicaciones tanto en Develop como en Master haciendo uso del plugin BlueOcean se logra ver de una manera entendible como va el despliegue de aplicaciones.

5.2. Recomendaciones.

Se recomienda como plan para utilizar los softwares de Integración Continua en los sistemas que administra el departamento de Dirección y Tecnologías de Información específicamente en el área de Redes y Comunicaciones que se realicen capacitaciones sobre su uso.

Se recomienda realizar una vista completa de todo lo comprende una cultura Devops para realizar un análisis completo y ver los siguientes procesos a mejorar con el ciclo de Deming.

Se recomienda la instalación de más plugins para la aseguración del software como, por ejemplo

- Selenium que nos permite la probar el correcto funcionamiento de las ventanas.
- Jmeter que nos permite ver la cantidad de carga soportada por los servicios desplegados
- Test unitarios. Jenkins nos permite reconocer los test unitarios y ejecutarlos por nosotros asegurando la calidad y performance de nuestras funciones.

Se recomienda la implementación de herramientas de monitoreo basado en LOGS y METRICAS para el seguimiento de aplicaciones y a si continuar con la implementación de una cultura DevOps en los grupos de trabajo.

Por último, se recomienda ver la automatización completa de aprovisionamiento de servicios basados en Nube.

REFERENCIAS.

Azure. (21 de 08 de 2017). *Azure*. Obtenido de Azure: https://docs.microsoft.com/es-es/azure/architecture/microservices/ci-cd

Garcia, E. (10 de 11 de 2016). *Equipo altran*. Obtenido de Equipo altran: https://equipo.altran.es/el-ciclo-de-deming-la-gestion-y-mejora-de-procesos/

Garzas, J. (31 de 07 de 2015). *javiergarzas.com*. Obtenido de javiergarzas.com: https://www.javiergarzas.com/2015/07/entendiendo-docker.html

Ionos. (15 de 05 de 2017). *Digital Ionos*. Obtenido de Digital Ionos: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/herramientas-de-integracion-continua/

Ionos. (30 de 07 de 2019). *Startup Guide Ionos*. Obtenido de Startup Guide Ionos: https://www.ionos.es/startupguide/productividad/circulo-de-deming/

Moreno, O. (29 de 10 de 2019). *oscarmoreno.com*. Obtenido de oscarmoreno.com: http://oscarmoreno.com/pipeline-jenkins/

RedHat. (12 de 07 de 2016). *RedHat*. Obtenido de RedHat: https://www.redhat.com/es/topics/devops/what-is-ci-cd

Sánchez, J. F. (02 de 07 de 2018). *https://sdos.es*. Obtenido de https://sdos.es: https://sdos.es/blog/pruebas-de-rendimiento-con-jmeter-ejemplos-basicos

Andreessen, M., & Drucker, P. (n.d.). DevOps Metrics. https://doi.org/10.1145/3159169

Banica, L., Radulesco, M., Rosca, D., & Hagiu, A. (2017). Is DevOps another Project Management Methodology? Informatica Economica, 21(3/2017), 39–51. https://doi.org/10.12948/issn14531305/21.3.2017.04

Chelladhurai, J. S., Vinod, S., & Pethuru, R. (2017). Learning Docker. In Materials Research Bulletin (Second Edi, Vol. 31). https://doi.org/10.1016/0025-5408(96)80018-3

Díaz, O., & Muñoz, M. (2018). Implementación de un enfoque DevSecOps + Risk

Management en un Centro de Datos de una organización Mexicana. RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação, 26, 43–53. https://doi.org/10.17013/risti.26.43-53

Guillermo Jiménez Marco, Yanuar, A., Suhartanto, H., Sahu, P. K., Boettiger, C.,

Eddelbuettel, D., … Stäubert, S. (2018). DevOps, la nueva tendencia en el desarrollo de sistemas TI, un caso práctico en el análisis de incidencias de software. Journal of Coastal Research, 82(1), 102. https://doi.org/10.2112/si82-013.1

Gundecha, U. (2014). Learning Selenium Testing Tool with Python. MUMBAI: Packt

Publishing Ltd.

Mitesh, S., & Berg, A. M. (2017). Jenkins 2.x Continuous Integration Cookbook – Third

Edition (Third Edit). Mumbai: Packt Publishing Ltd.

Olmedo, P. B., & Moyano Pucheta, F. N. (2018). El rol de Docker para ejecutar pruebas

Automatizadas como parte de la Integración Continua. XLVII Jornadas Argentinas de Informática e Investigación Operativa (47 JAIIO), 167–173. Retrieved from http://47jaiio.sadio.org.ar/sites/default/files/EST-15.pdf

Sandobalin-Guamán, J., Zuñiga-Prieto, M., Insfran, E., Abrahão, S., & Cano, C. (2016). Una

Aproximación DevOps para el Desarrollo Dirigido por Modelos de Servicios Cloud. XII Jornadas de Ciencia e Ingeniería de Servicios (JCIS), (July), 83–92. Retrieved from http://biblioteca.sistedes.es/articulo/una-aproximacion-devops-para-el-desarrollo-dirigido-por-modelos-de-servicios-cloud/

Sobrevilla, G., Hernández, J., Velasco, P., & Soriano, S. (2017). Aplicando Scrum y

Prácticas de Ingeniería de Software para la Mejora Continua del Desarrollo de un Sistema Ciber-Físico. ReCIBE, Revista Electrónica de Computación, Informática, Biomédica y Electrónica, 6(1), 1–15. Retrieved from http://recibe.cucei.udg.mx/ojs/index.php/ReCIBE/article/view/60/58

Standard, I. (2011). International Standard ISO / IEC / IEEE Systems and software engineering — Life cycle processes — Project management (First Edit).

Zavala Ruiz, J. M. (2015). ¿Por Qué Fracasan los Proyectos de Software ? Un Enfoque Organizacional. Congreso Nacional de Software Libre, (2), 20–42. https://doi.org/10.13140/RG.2.1.4741.3206

ANEXOS

Anexo A. Instancias del servidor phserver03.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Name	State	Status	Managed By	Host	Provisioned Spa	Used Space	Host CPU	Host Mem	Guest Mem - %	Guest OS	Compatibility	Memory Size	Reservation	CPUs
2	Academico Prue	Powered Off	Normal		phserver03.upe	19.21 GB	16 GB	0 Hz	0 B	0	CentOS 7 (64-bi	ESXi 6.5 and lat	3 GB	0 B	4
3	API BI Contable	Powered On	Normal		phserver03.upe	42.11 GB	7.51 GB	0 Hz	679 MB	1	CentOS 7 (64-bi	ESXi 6.5 and lat	2 GB	0 B	2
4	ContaWeb New	Powered Off	Normal		phserver03.upe	74.21 GB	70 GB	0 Hz	0 B	0	CentOS 7 (64-bi	ESXi 6.5 and lat	4 GB	0 B	4
5	Cpe_Win_factur	Powered On	Normal		phserver03.upe	372.13 GB	346.18 GB	263 MHz	6.06 GB	23	Microsoft Windo	ESXi 5.0 and lat	6 GB	0 B	6
6	Db Huellas 2019	Powered Off	Normal		phserver03.upe	208.32 GB	200 GB	0 Hz	0 B	0	Microsoft Windo	ESXi 5.0 and lat	8 GB	0 B	12
7	Db Huellas Prod	Powered On	Normal		phserver03.upe	208.11 GB	208.11 GB	2.66 GHz	8.1 GB	40	Microsoft Windo	ESXi 5.0 and lat	8 GB	0 B	12
8	Elvis Contable B	Powered Off	Normal		phserver03.upe	58.21 GB	50 GB	0 Hz	0 B	0	Microsoft Windo	ESXi 6.5 and lat	8 GB	0 B	6
9	GLPI 2021	Powered On	Normal		phserver03.upe	44.11 GB	44.11 GB	0 Hz	4.04 GB	11	Ubuntu Linux (6-	ESXi 6.5 and lat	4 GB	0 B	4
10	Lnx Abiatar Proc	Powered On	Normal		phserver03.upe	1,015.11 GB	600.16 GB	95 MHz	14.88 GB	4	Oracle Linux 4/5	ESXi 5.0 and lat	15 GB	0 B	15
11	Lnx_Academicol	Powered On	Normal		phserver03.upe	12.12 GB	12.12 GB	71 MHz	700 MB	1	Other 2.6.x Linu	ESX/ESXi 4.0 ar	4 GB	0 B	4
12	Lnx_ContaWeb	Powered On	Normal		phserver03.upe	74.11 GB	74.11 GB	71 MHz	1.67 GB	4	CentOS 4/5 or la	ESX/ESXi 4.0 ar	4 GB	0 B	4
13	Oradatos Produ	Powered On	Normal		phserver03.upe	730.11 GB	621.73 GB	0 Hz	1.62 GB	0	Oracle Linux 6 (6	ESXi 6.5 and lat	20 GB	0 B	20
14	PostgresDB	Powered On	Normal		phserver03.upe	55.11 GB	10.53 GB	0 Hz	1.16 GB	1	Ubuntu Linux (6-	ESXi 6.5 and lat	5 GB	0 B	4
15	ProxyPass	Powered On	Normal		phserver03.upe	33.11 GB	6.77 GB	0 Hz	554 MB	2	CentOS 6 (64-bi	ESXi 6.5 and lat	3 GB	0 B	4
16	Radius Base	Powered Off	Normal		phserver03.upe	64.21 GB	60 GB	0 Hz	0 B	0	CentOS 7 (64-bi	ESXi 6.5 and lat	4 GB	0 B	2
17	Radius Producti	Powered On	Normal		phserver03.upe	64.11 GB	64.11 GB	0 Hz	939 MB	0	CentOS 7 (64-bi	ESXi 6.5 and lat	4 GB	0 B	2
18	Residencias	Powered Off	Normal		phserver03.upe	56.21 GB	50 GB	0 Hz	0 B	0	CentOS 7 (64-bi	ESXi 6.5 and lat	6 GB	0 B	3
19	Sim Backend	Powered On	Normal		phserver03.upe	19.11 GB	19.11 GB	0 Hz	882 MB	0	CentOS 7 (64-bi	ESXi 6.5 and lat	4 GB	0 B	4
20	Sim New	Powered On	Normal		phserver03.upe	86.11 GB	86.11 GB	23 MHz	4.04 GB	33	Microsoft Windo	ESX/ESXi 4.0 ar	4 GB	0 B	2
21	SysProject Back	Powered Off	Normal		phserver03.upe	24.19 GB	20 GB	0 Hz	0 B	0	Ubuntu Linux (6-	ESXi 6.5 and lat	4 GB	0 B	4
22	Virtual Smart Zo	Powered On	Normal		phserver03.upe	113.11 GB	113.11 GB	1.68 GHz	13.05 GB	47	Other (32-bit)	ESXi 5.0 and lat	13 GB	0 B	4
23	Win_sim_ctr_clo	Powered Off	Normal		phserver03.upe	99.86 GB	97.67 GB	0 Hz	0 B	0	Microsoft Windo	ESX/ESXi 4.0 ar	2 GB	0 B	2
24															

Anexo B. Instancias del servidor phserver02.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Name	State	Status	Managed By	Host	Provisioned Spa	Used Space	Host CPU	Host Mem	Guest Mem - %	Guest OS	Compatibility	Memory Size	Reservation	CPUs
2	Academico Prod	Powered Off	Normal		phserver02.upe	44.21 GB	40 GB	0 Hz	0 B	0	CentOS 7 (64-bi	ESXi 6.5 and lat	4 GB	0 B	4
3	acuerdos_backe	Powered Off	Normal		phserver02.upe	18.21 GB	15 GB	0 Hz	0 B	0	CentOS 7 (64-bi	ESXi 6.5 and lat	3 GB	0 B	4
4	AcuerdosTest	Powered Off	Normal		phserver02.upe	22.2 GB	20 GB	0 Hz	0 B	0	Ubuntu Linux (6-	ESXi 6.5 and lat	2 GB	0 B	3
5	Admision	Powered On	Normal		phserver02.upe	82.08 GB	11.22 GB	0 Hz	2.03 GB	2	Ubuntu Linux (6-	ESXi 6.5 and lat	2 GB	0 B	3
6	Alfresco of	Powered Off	Normal		phserver02.upe	84.2 GB	15.03 GB	0 Hz	0 B	0	Ubuntu Linux (6-	ESXi 6.5 and lat	4 GB	0 B	6
7	alfresco_back	Powered Off	Normal		phserver02.upe	85.2 GB	76.82 GB	0 Hz	0 B	0	Ubuntu Linux (6-	ESXi 6.5 and lat	5 GB	0 B	4
8	Auth	Powered On	Normal		phserver02.upe	42.09 GB	42.09 GB	0 Hz	1.23 GB	1	Ubuntu Linux (6-	ESXi 6.5 and lat	2 GB	0 B	4
9	Capellania	Powered Off	Normal		phserver02.upe	19.21 GB	16 GB	0 Hz	0 B	0	CentOS 7 (64-bi	ESXi 6.5 and lat	3 GB	0 B	2
10	Capellania 2	Powered Off	Normal		phserver02.upe	74.2 GB	70 GB	0 Hz	0 B	0	CentOS 4/5 or la	ESX/ESXi 4.0 ar	4 GB	0 B	4
11	Congreso Binaci	Powered On	Normal		phserver02.upe	22.08 GB	22.08 GB	0 Hz	1.16 GB	2	CentOS 7 (64-bi	ESXi 6.5 and lat	2 GB	0 B	2
12	Control de Huell	Powered On	Normal		phserver02.upe	85.09 GB	85.09 GB	25 MHz	5.05 GB	4	Microsoft Windo	ESXi 5.0 and lat	5 GB	0 B	4
13	Control Huellas I	Powered Off	Normal		phserver02.upe	85.2 GB	80 GB	0 Hz	0 B	0	Microsoft Windo	ESXi 5.0 and lat	5 GB	0 B	4
14	Desarrollo Test	Powered Off	Normal		phserver02.upe	88.21 GB	12.43 GB	0 Hz	0 B	0	Ubuntu Linux (6-	ESXi 6.5 and lat	8 GB	0 B	12
15	Docker Registry	Powered Off	Normal		phserver02.upe	82.21 GB	80 GB	0 Hz	0 B	0	CentOS 8 (64-bi	ESXi 6.7 and lat	2 GB	0 B	2
16	Fronteds Angula	Powered Off	Normal		phserver02.upe	32.2 GB	10.35 GB	0 Hz	0 B	0	Ubuntu Linux (6-	ESXi 6.5 and lat	2 GB	0 B	3
17	GitLab Clone	Powered Off	Normal		phserver02.upe	124.21 GB	120 GB	0 Hz	0 B	0	CentOS 7 (64-bi	ESXi 6.5 and lat	4 GB	0 B	4
18	GitLab Producio	Powered On	Normal		phserver02.upe	124.09 GB	124.09 GB	467 MHz	3.93 GB	26	CentOS 7 (64-bi	ESXi 6.5 and lat	4 GB	0 B	4
19	GLPI Server	Powered On	Normal		phserver02.upe	106.09 GB	106.09 GB	25 MHz	5.93 GB	2	CentOS 7 (64-bi	ESXi 6.5 and lat	6 GB	0 B	5
20	Infraestructura D	Powered On	Normal		phserver02.upe	903.08 GB	903.08 GB	0 Hz	2.92 GB	1	CentOS 7 (64-bi	ESXi 6.5 and lat	3 GB	0 B	2
21	jenkins Back	Powered Off	Normal		phserver02.upe	66.21 GB	60 GB	0 Hz	0 B	0	CentOS 8 (64-bi	ESXi 6.7 and lat	6 GB	0 B	4
22	Lnx_Abiatar_Pru	Powered On	Normal		phserver02.upe	724.1 GB	385.9 GB	77 MHz	7.18 GB	2	Oracle Linux 4/5	ESXi 5.0 and lat	12 GB	0 B	8
23	Lnx_APIREST-P	Powered On	Normal		phserver02.upe	21.08 GB	21.08 GB	0 Hz	1.01 GB	1	Ubuntu Linux (6-	ESXi 5.0 and lat	1 GB	0 B	2
24	Lnx_Cat Wordpr	Powered On	Normal		phserver02.upe	41.08 GB	13.49 GB	0 Hz	1.03 GB	4	Ubuntu Linux (6-	ESXi 6.5 and lat	1 GB	0 B	3
25	Lnx_DNS_Prima	Powered On	Normal		phserver02.upe	22.08 GB	22.08 GB	0 Hz	1.19 GB	12	CentOS 4/5 or la	ESXi 5.0 and lat	2 GB	0 B	2
26	Lnx_Evento_Fro	Powered Off	Normal		phserver02.upe	12.17 GB	10 GB	0 Hz	0 B	0	Ubuntu Linux (6-	ESXi 6.5 and lat	1.98 GB	0 B	2
27	Lnx_Fronteo_Ba	Powered Off	Normal		phserver02.upe	82.2 GB	80 GB	0 Hz	0 B	0	Ubuntu Linux (6-	ESXi 6.5 and lat	2 GB	0 B	5