PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR SEDE ESMERALDAS

ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

TESIS DE GRADO

TÍTULO:
"ENTORNO DE INTEGRACIÓN, ENTREGA Y DESPLIEGUE CONTÍNUO DE SOFTWARE EN LA UNIVERSIDAD CATÓLICA SEDE ESMERALDAS"

LINEA DE INVESTIGACIÓN:
SOFTWARE Y TÉCNOLOGÍA

AUTOR:
VINUEZA CELI OSCAR STALIN

ASESOR:
Mgt. MARC GROB

TRIBUNAL DE GRADUACIÓN

Título: Entorno de integración, entrega y despliegue continuo de software en la Universidad Católica Sede Esmeraldas.

Autor(a): Oscar Stalin Vinueza Celi

Mgt. Marc Grob **f.** _____
Asesor

Mgt. Gustavo Chango Sailema **f.** _____
Lector #1

Mgt. Jaime Sayago **f.** _____
Lector #2

Mgt. Xavier Quiñonez Kú **f.** _____
Coordinador de carrera

DEDICATORIA

Esta investigación se la dedico a Dios por nunca soltar mi mano y guiarme por el camino correcto, ser mi fortaleza y sabiduría.

A mi madre Betty Celi la estrella que guía mi camino, por el inmenso sacrificio para hacer de mí el hombre que soy hoy, su fuerza, carácter y dirección lo son todo en mi vida.

A mi padre Hugo Vinueza por encomendarme la tarea de ser el orgullo que honrara su memoria, por impulsarme con su recuerdo en los momentos más difíciles.

A mis hermanos: Belén, Juan y Vanessa quienes jamás me han abandonado e incondicionalmente han cuidado de mi con cariño y una parte de cada uno de ellos hace parte de mi ahora.

Al amor de mi vida, Elizabeth quien estuvo desde antes del inicio de mi carrera a mi lado y me ha dado todo el apoyo y el amor incondicional que he necesitado.

Oscar Vinueza.

AGRADECIMIENTO

Agradezco a Dios la voluntad, la fuerza y dedicación que me permitió llegar a la culminación de mi pregrado.

A mi familia, por ser parte de este proceso con paciencia y cariño, ser mi apoyo y refugio.

Agradezco a mis amigos, quienes hicieron de mi vida universitaria un camino lleno de alegrías y bellas experiencias.

A mis profesores por sus enseñanzas, quienes con paciencia, profesionalismo y sabiduría nos guiaron en nuestro camino a lo largo de la carrera.

Oscar Vinueza.

RESUMEN

La presente investigación fue realizada con el objetivo principal de modernizar los ambientes de desarrollo y producción a través de la implementación de un entorno de integración, entrega y despliegue continuo de software (CI/CD) por sus siglas en inglés, en el departamento de TICS de la Universidad Católica Sede Esmeraldas.

Se realizó una investigación teórica sobre los conceptos de cada una de las herramientas y tecnologías que conforman un entorno de un CI/CD, se hizo una identificación de las herramientas que utiliza el departamento de TICS y se aplicó la ISO 25010 para realizar una evaluación y comparación de herramientas de servidores de automatización y de pruebas automatizadas de código para seleccionar las herramientas que completarán el entorno de CI/CD propuesto y que posteriormente se implementó para formar el entorno de desarrollo y de producción del sistema médico de la PUCESE.

Para ello se evaluaron aspectos como la completitud funcional, la capacidad de aprendizaje y la reusabilidad de cada una de las herramientas para determinar la calidad de las mismas y poder realizar una elección.

Con la aplicación de la ISO se pudo determinar que de los servidores de automatización Jenkins fue la herramienta seleccionada e implementada, en conjunto con Cypress.io como herramienta de pruebas automatizadas de código, dado que cumplía con las necesidades y especificaciones necesarias en cada uno de los aspectos evaluados que tenía el departamento de TICS de la PUCESE.

Palabras claves: integración, entrega, despliegue, software.

ABSTRACT

This research was carried out with the main objective of modernizing development and production environments through the implementation of a CI / CD environment in the ICT department of the Esmeraldas Headquarters Catholic University.

A theoretical investigation was carried out on the concepts of each of the tools and technologies that make up a CI / CD environment, an identification of the tools used by the TICS department was made, and ISO 25010 was applied to carry out an evaluation and Comparison of automation server tools and automated code testing to select the tools that will complete the proposed CI / CD environment and which was subsequently implemented to form the development and production environment of the PUCESE medical system.

For this, aspects such as functional completeness, learning capacity and reusability of each of the tools were evaluated to determine their quality and to be able to make a choice.

With the application of the ISO it was possible to determine that of the Jenkins automation servers it was the tool selected and implemented, together with Cypress.io as an automated code testing tool, given that it met the needs and specifications necessary in each of the the evaluated aspects that the ICT department of PUCESE had.

Key words: integration, delivery, deployment, software.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

INTRODUCCIÓN

PRESENTACIÓN DE LA INVESTIGACIÓN

El presente trabajo de investigación titulado “Entorno de integración, entrega y despliegue continuo de software en la Pontificia Universidad Católica Sede Esmeraldas” se centró en la implementación un entorno de desarrollo y producción de aplicaciones automatizado, asegurando la calidad de las herramientas que conformen el nuevo entorno mediante la cuidadosa selección de las mismas.

Para la evaluación de las herramientas se aplicó la norma ISO/IEC 25010 que define parámetros para la evaluación de la calidad de software. Cabe resaltar que no todas las herramientas que conforman el nuevo entorno serán evaluadas debido a que algunas de las herramientas son definidas por el departamento de TICS de la universidad.

Este proyecto de investigación consta de tres capítulos, descritos a continuación:

El capítulo I consiste en el marco teórico, donde se detallan los conceptos teóricos elementales y necesarios de la investigación, considerando los antecedentes como base de estudios realizados, y sus diferentes definiciones elementales como definición de herramientas y conceptos.

El capítulo II presenta las metodologías empleadas para el desarrollo de la investigación, detallando aspectos importantes y los instrumentos metodológicos utilizados.

El capítulo III describirá los resultados obtenidos al realizar la parte experimental de la investigación poniendo en práctica las bases teóricas que se detallan en el capítulo I y II.

PLANTEAMIENTO DEL PROBLEMA

En la actualidad el departamento de tecnologías de la información y comunicación (TICs) de la Pontificia Universidad Católica Sede Esmeraldas no cuenta con un entorno de integración, entrega y despliegue automatizado de software, el grupo de ingenieros de sistemas que se encarga del desarrollo y mantenimiento de sistemas del departamento utiliza la metodología SCRUM como procedimiento de desarrollo ágil que le permita el desarrollo y mantenimiento de software de manera rápida y eficiente. Sin embargo, a pesar de utilizar repositorios para llevar un control de versiones y usar un servidor local para el despliegue de las aplicaciones, los procesos involucrados en el despliegue de software no tienen un elemento de automatización ni se realizan pruebas automatizadas al código antes de ser desplegado, las pruebas son manuales a su vez. Es por eso que se evidencia la necesidad de crear un entorno de integración y despliegue continuo que reduzca los tiempos y costos de despliegue a la vez que se asegura la calidad y funcionamiento del software que se desarrolla, este entorno debe conformarse por varios tipos de herramientas que permitan controlar las versiones, automatizar procesos, medir la calidad del código, realizar pruebas automatizadas y servir la aplicación para fines de desarrollo o producción [1].

Un entorno de CI/CD permite ser aplicado no solamente en un proyecto, sino que es extensible para poder ser implementado en todos los proyectos existentes y futuros que se lleven a cabo en el departamento de TICS de la PUCESE, adoptando la metodología ágil DEVOPS en todas las operaciones de desarrollo de software y logrando optimizar el proceso de mantenimiento, desarrollo y actualización de aplicaciones [2].

JUSTIFICACIÓN

A través de esta investigación se buscó conocer herramientas y tecnologías que acorten los tiempos de despliegue e integración del software y que también permitan un control automatizado de la calidad del código desarrollado. Evitando caer en la pérdida de confiabilidad en el software, pérdida de confianza y escalabilidad del cliente como resultado de la falta de control de calidad, que en muchos casos no se hace precisamente por no tener herramientas que se integren en un entorno de integración y despliegue continuo que tenga un servidor que permita automatizar este control de calidad, haciendo que se haga de manera manual, agregando costos al proyecto sobre todo en cuanto a tiempos de desarrollo [3].

Otro de los aportes de esta investigación es el análisis y evaluación de la integración en el entorno de entrega y despliegue continúo de herramientas que permitió realizar el testeo del código de manera automatizada en el proceso de entrega e integración, lo cual reducen los errores en el código a través de la aplicación de estas herramientas en el tiempo de ejecución del despliegue del software, normalmente estas pruebas se realizan manualmente luego de cada avance en la codificación, sin embargo no siempre se realiza antes del despliegue, un error humano puede hacer que el testeo no se haga y en el despliegue se incluyan errores [4].

OBJETIVOS

GENERAL

Implementar un modelo de integración y despliegue automatizado de software a través de la determinación de herramientas que conformen el entorno y la evaluación de la calidad de las mismas.

ESPECÍFICOS

- Identificar herramientas utilizadas para despliegue e integración continua.
- Seleccionar las herramientas necesarias para la creación del entorno.
- Comparar las herramientas de integración y despliegue continuo de software en base a la norma ISO 25010.

CAPÍTULO I: MARCO TEÓRICO

1.1 ANTECEDENTES

En la investigación realizada por Gallaba, el principal objetivo es estudiar y mejorar la robustez que tienen los entornos de integración y despliegue automatizado de software, debido a que si bien estos entornos proveen de una manera ágil de convertir los aportes individuales de los desarrolladores en entregables capaces de ser desplegados en un entorno de producción, si los mismos no se encuentran bien operados o administrados pueden ocasionar problemas con el flujo de trabajo y ocasionar contratiempos. La metodología utilizada en este artículo científico se basa en un análisis bibliográfico de la configuración de los servicios modernos de CI/CD, la síntesis de una hipótesis sobre los problemas actuales de este tipo de entornos y por último un plan para mejorar la robustez de los mismos. Los resultados de esta investigación sugieren que los lenguajes de configuración de los pipelines de los servidores de automatización en ocasiones pueden ser mal utilizados, con lo cual se genera una mala configuración de los entornos de CI/CD desencadenando una serie de problemas en el desarrollo y producción de software, como una solución a esta problemática se recomienda la utilización de una herramienta que permita evitar estos errores de mala configuración de los nodos que conforman los pipelines. Como conclusión se halló que existen diversas problemáticas en la administración de entornos de CI/CD y que la investigación en cuestión pudo recomendar acciones para evitar estas a partir del abordaje de puntos clave en el desarrollo de estos entornos y de esta manera convertirlos en sistemas robustos [5].

En el artículo científico [4] el objetivo principal fue generar un marco de trabajo eficiente a través del cual se lleve a cabo una correcta aplicación de integración, pruebas y entrega continua para automatizar la compilación del código fuente, análisis de código, ejecución de prueba, empaquetado, infraestructura aprovisionamiento, implementación y reportes del entorno. Para llevar a cabo este objetivo se realizó un análisis a fondo sobre los requisitos que tienen las compañías que implementan CI/CD para conocer sus necesidades y en base a las mismas generar un marco de trabajo capaz de satisfacer dichas necesidades. Los resultados de esta investigación se evidencian a través de la generación de un marco de trabajo que describe en cada una de las fases que componen los entornos de integración, entrega y despliegue continuo de software en donde se detalla el paso a paso de cómo se recomienda realizar la implementación de estos entornos en compañías con necesidades similares a las descritas mediante la utilización de pipelines, pruebas unitarias automatizadas, uso de máquinas virtuales, uso de repositorios (Git, SVN, StarStream), el uso de Jenkins como servidor de automatización, Junit como herramienta para la automatización de pruebas unitarias y el uso de Sonar y FindBug como analizadores de código estático para las pruebas de calidad de código fuente. Las conclusiones de esta investigación resaltan la importancia del uso de herramientas que aseguren la calidad de código fuente dentro de los entornos de CI/CD y el uso de ambientes de desarrollo y producción basados en la nube para la infraestructura de los entornos, así como el enfoque en el monitoreo y seguridad de los entornos.

En la investigación [6] se tiene como objetivo principal la determinación de la capacidad de la entrega continua de software, así como servir de guía para desarrolladores informando sobre los beneficios y desafíos a la hora de la implementación de esta práctica. Para lograr estos objetivos se desarrolló una recopilación bibliográfica sobre las metodologías que se deben tomar en cuenta y que se recomienda utilizar a la hora de realizar la entrega continua de software, así como la estructura que debería tener el entorno para implementarla. Los resultados de esta investigación consisten en una serie de recomendaciones entre las que destacan el uso de pipelines en servidores de automatización que son los encargados de orquestar las acciones que se deben tomar en un algoritmo para obtener una entrega satisfactoria, esto de la mano de las pruebas automatizadas de

código, que, como ya es común depende del tipo de proyecto y las necesidades de la empresa, se elige un tipo específico de pruebas que se debe realizar al proyecto para que las entregas de los avances realizados por los programadores sean satisfactorios, en este caso se enfoca como ejemplo en las pruebas de aceptación, y como se venía adelantando en los objetivos de la investigación, hacia el final del artículo se habla de los beneficios de esta práctica sin perder de vista los desafíos que supone al mismo tiempo.

En el artículo científico [7] el objetivo principal es la realización de una evaluación a catorce diferentes analizadores de código estático con el afán de determinar elementos de los mismos como por ejemplo el soporte que daba cada uno para los diferentes tipos de errores y la diferencia de sus resultados frente al resto. Para lograr realizar este experimento se tomó catorce participantes que corrieran los diferentes analizadores estáticos de código sobre un mismo código fuente y se evaluaron varios aspectos funcionales de cada herramienta para poder evidenciar las diferencias que hay entre una herramienta y otra, y conocer el margen entre las mismas, para ello se definieron métricas de evaluación, así como escenarios de prueba. Los resultados de la investigación demostraron que existe una gran diferencia entre una herramienta y otra, debido al soporte a los diferentes tipos de errores a los que fueron expuestas las herramientas, los resultados son bastante variados y se contó con gran variedad de escenarios, datos de pruebas, elementos de evaluación y ponderaciones. Como conclusión de esta investigación se destaca el enfoque técnico y funcional que tuvo la misma, debido a que aspectos como usabilidad y capacidad de integración también son importantes y deben evaluarse a la hora de realizar una selección de herramientas de análisis de código estático, así como destacó que la efectividad con la que una herramienta detecta errores depende y varía en función de la complejidad del código con lo cual el comportamiento de las diferentes herramientas varía dependiendo el escenario.

1.2 BASES TEÓRICAS CIENTÍFICAS

1.2.1 ARQUITECTURA DE UN ENTORNO DE CI/CD

De manera general el proceso mediante el cual se realiza integración y entrega continua de software empieza cuando los desarrolladores realizan un commit en una rama determinada que se encuentra asociada a la rama correspondiente a un entorno determinado, este evento es escuchado por un disparador en un servidor de automatización el cual se encuentra conectado al repositorio, este disparador hace que se ejecute un trabajo de automatización en el cual se realizan pruebas automatizadas de código fuente, pruebas de control de calidad, pruebas de extremo a extremo, entre otros.

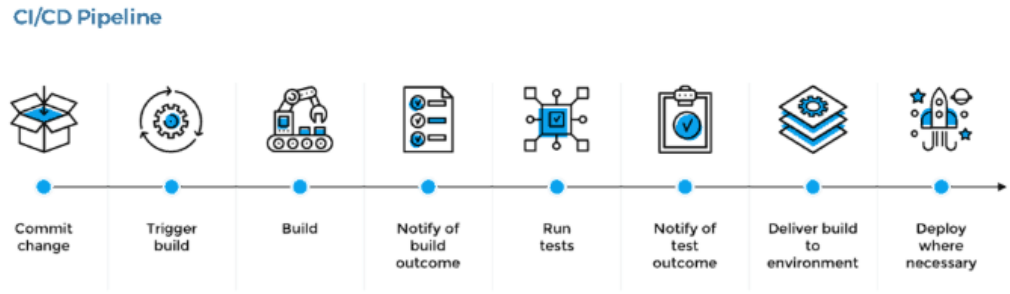

Figura 1. Algoritmo de CI/CD [1].

Posteriormente se realiza la integración de estos cambios a la rama correspondiente al entorno en el cual se requiera a través de una conexión remota desde el servidor de automatización hacia el servidor para realizar la entrega de manera automática. El despliegue es una extensión de la entrega en el cual en lugar de hacer el despliegue de manera manual según lo considere conveniente un operador humano, se hace de manera automatizada a través del servidor automatizado, previa la realización de pruebas adicionales que aseguren un despliegue exitoso. En esta fase se realizan de manera común pruebas automatizadas de integración, de aceptación y de extremo a extremo.

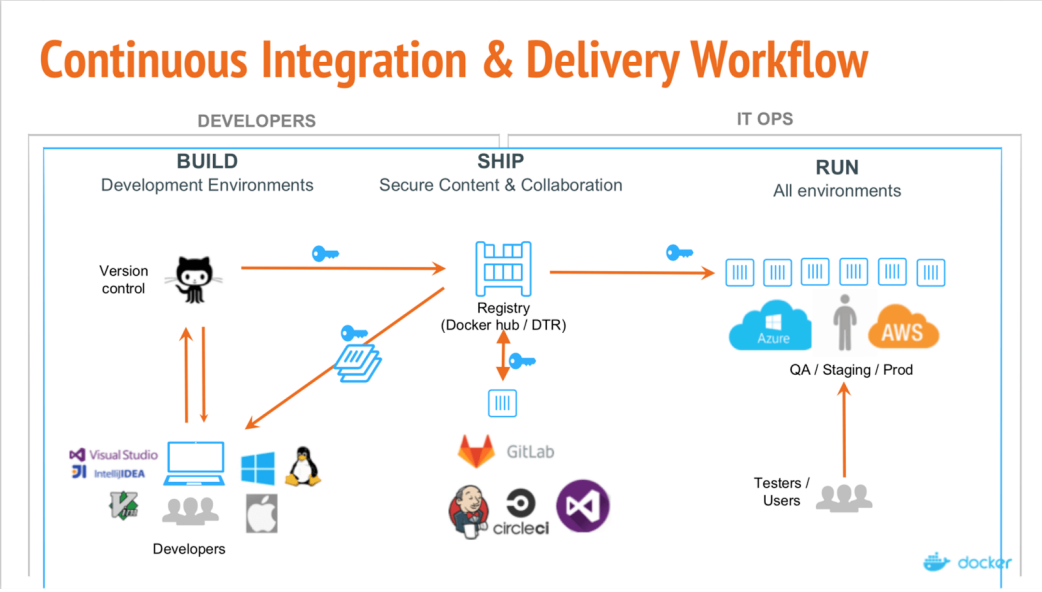

Figura 2. Entorno de CI/CD [2].

En los entornos de CI/CD intervienen distintos tipos de herramientas que se complementan e integran para conformar un entorno de integración, entrega y despliegue continuo, dentro de estos tipos de herramientas se encuentran los repositorios como github o bitbucket, servidores de automatización tales como Jenkins o gitlab, herramientas de pruebas como SonarQube o Cypress, contenedores de software como Docker o Podman.io, y servicios en la nube como AWS o Google Cloud.

1.2.2 DevOps

DevOps es un conjunto de herramientas y procedimientos que tiene como objetivo principal es reducir al máximo el tiempo que requiere realizar cambios en un sistema ya sea en desarrollo o en producción, al mismo tiempo que se asegura de que se mantenga la calidad, integridad y confianza del sistema. Para ello se trata de cerrar la brecha existente entre el desarrollo, pruebas y operaciones relacionadas con el despliegue, centrándose en la automatización de los procesos de despliegue [3].

1.2.3 Metodologías ágiles de DevOps

Las metodologías como la integración, entrega y despliegue continuo nacen como metodologías que son usadas por la metodología de desarrollo ágil DevOps, esto dada la necesidad de que se unifiquen las prácticas de desarrollo con las prácticas de despliegue y operaciones de software con la finalidad de realizar mayor cantidad de entregas y en menor cantidad de tiempo, mediante el uso de estas metodologías que a su vez usan herramientas que permiten la automatización de estos procesos [6].

La integración continua y entrega continua son parte de las metodologías utilizadas por la cultura DevOps para mejorar el rendimiento y calidad de las entregas que se realizan, para de este modo también poder obtener una rápida retro alimentación por parte de los clientes y usuarios a quienes va dirigido el desarrollo, la integración, entrega y despliegue continuo son fases del despliegue automatizado, pueden haber despliegues manuales, como integraciones y entregas manuales, la automatización de estos procesos se conoce como despliegue continuo [4].

1.2.3.1 Integración continua

La integración continua permite a los desarrolladores trabajar de manera coordinada, realizar cambios en el código de un proyecto e integrar dichos cambios no solo en la rama de cada programador, sino que a través de herramientas de automatización integra dichos cambios directamente en la rama maestra del repositorio, sin tener que realizar cada integración de manera manual, los equipos de desarrollo tienen varios desarrolladores trabajando en el código cada día, y cada uno de ellos hace varias integraciones diariamente, por lo que la cantidad de tiempo ahorrada entre realizar este proceso de manera manual y hacerlo de manera automática, es significativa. Al ser una parte del despliegue continuo [8].

La integración continua es una práctica mediante la cual los desarrolladores integran de manera constante los avances realizados de manera individual, utilizando un repositorio como GIT o Mercurial, pero para llevar a cabo una

integración continua es necesario servirse de otras herramientas, para poder testear el código antes de poder realizar la integración, también se puede analizar el código a ser integrado con una plataforma de control de calidad, todo esto de manera automática a través de un servidor de automatización, de otro modo este proceso llevaría muchas horas e incluso días [9].

1.2.3.2 Entrega continua

La entrega continua es una extensión de la integración continua y consiste en que además de integrar de manera automática los cambios en el repositorio, estos a través de una instrucción manual se implementan directamente en el servidor de pruebas o de producción, haciendo que los cambios tanto funcionales como de interfaz sean visibles con cada proceso de despliegue que se lleve a cabo, este proceso de despliegue no se hace con los cambios que hace cada programador en su propia rama, o la rama del cambio, sino que se realiza después de cada integración en la rama maestra, ya que son esos cambios los que a través de herramientas de pruebas de control de calidad y pruebas que buscan evitar errores funcionales, los que pueden desplegarse en el servidor de pruebas o de producción [10].

La entrega continua además de permitirnos ver en el servidor de producción ya que esta práctica se enfoca en que el código siempre esté listo para la puesta en producción de aquello que fue anteriormente integrado en el repositorio, permite que la retroalimentación sea casi instantánea con el o los clientes, estas entregas llevan por detrás todo un proceso automatizado de pruebas de código de todo tipo, sobre todo aquellas pruebas que permiten prevenir problemas de integración o aceptación de los cambios, para realizar una entrega continua de manera correcta es necesario realizar antes una correcta integración continua, en otras palabras estas prácticas van de la mano [4].

1.2.3.3 Despliegue continuo.

El despliegue continuo es un paso adicional a la integración y entrega continua, dentro de todo este proceso de automatización esta es una nueva fase, en esta fase la automatización se aplica al despliegue automático, que en la fase de entrega continua se realiza de manera manual, pero para llegar a realizarlo de manera automática primero se deben tomar algunas precauciones que normalmente un operador de devops tomaría antes de enviar los cambios a despliegue en servidor de producción, estas precauciones son un algoritmo que se debe llevar a cabo para asegurar que los nuevos cambios no llegue para romper la aplicación y que a pesar de ser un proceso automático, cada despliegue se realice de manera exitosa, esta fase de despliegue automático no la llevan a cabo todos los equipos de desarrollo o todas las empresas que utilizan integración y entrega continua, ya que depende mucho de la lógica del negocio tomar la decisión de realizar el proceso de despliegue de manera automática o no [9].

1.2.3.4 Software Pipelines

Las tuberías de desarrollo o Pipelines en inglés son un canal de procesos por los cuales debe pasar un software antes de ser desplegado de manera automatizada, estos procesos son una expresión de los conceptos de integración, entrega y despliegue del software ya que a través de las fases que se llevan a cabo automáticamente se realiza la integración con los repositorios y sus ramas, se realizan pruebas automatizadas, pruebas de distintos tipos, revisiones de código y por último la entrega del mismo en los ambientes de desarrollo y producción, permitiendo que el proceso en general se eficiente y repetible [11].

Estas tuberías de desarrollo se usan en los servidores de automatización como Jenkins para programar los procesos que se deben realizar antes de realizar una entrega de software, Jenkins por ejemplo implementa documentos que tienen una sintaxis determinada con los cuales se puede programar la tubería de desarrollo para ser leída por el servidor directamente en el directorio del proyecto y que de este modo se realicen los procesos de manera automatizada [11].

1.2.4 Herramientas para el uso de las metodologías ágiles de DevOps

La integración, entrega y despliegue continuo para su implementación hacen uso de diferentes herramientas para lograr la automatización de estos procesos, existen de cada tipo de herramientas, muchas opciones, los tipos de herramientas utilizadas que son más comunes son: uso de contenedores, realización de red de microservicios, uso de servidores de automatización, servidores de control de calidad QA, uso de plataformas para pruebas de código y repositorios de código, estos son los actores que intervienen en la estructura del uso de metodologías ágiles de DevOps, conforman el entorno de integración, entrega y despliegue continuo de software.

1.2.4.1 Tecnología de contenedores.

Como se puede ver en la página web de Docker, ellos definen un contenedor como una unidad de software estándar, y por simple que parezca es ahí donde radica la importancia y el gran desarrollo que ha generado esta tecnología, ya que a comparación de la virtualización es posible mantener a través de un contenedor un entorno configurado, con una versión determinada de módulos listo para hacer correr sobre él la aplicación que sobre ese entorno se desarrolló y se desea desplegar, en virtualización es mucho más complicado mantener las versiones de los módulos para un entorno óptimo, más aún cuando estas versiones ya no se encuentran disponibles, la tecnología de contenedores también hace posible una más fácil y rápida configuración de los módulos a utilizar, lo que permite una mayor facilidad para la migración de aplicaciones entre diferentes entornos, también permite un aislamiento controlado entre el entorno que corre sobre el contenedor y el externo, lo cual hace que sea más fácil proteger los diferentes servicios que se pueden mantener en una red de contenedores, otra característica importante de esta tecnología es que permite realizar la arquitectura de micro servicios, en la cual se tienen múltiples contenedores que brindan un servicio aislado e independiente que puede servir para una o varias aplicaciones al mismo tiempo de manera eficiente además ya que los contenedores a diferencia de las

máquinas virtuales ocupan muchos menos recursos y son más rápidas para el arranque de los servicios [12].

La forma más común de implementar esta tecnología es a través de una máquina física o virtual con un sistema operativo Linux o Windows que sirven como base en donde se debe instalar la tecnología de contenedores para que sobre el mismo levante los contenedores que van a formar parte de la infraestructura que necesita la o las aplicaciones que se requieran implementar, existen algunas marcas de contenedores que se pueden utilizar para implementar tecnología de contenedores, en esta investigación analizaremos las siguientes:

Docker

Docker es una empresa que se dedica a proveer el software necesario para el uso de sus contenedores, también posee un repositorio donde se almacenan y se pueden usar las imágenes para la creación de contenedores basados en las mismas, este repositorio además permite la creación de imágenes personalizadas a partir de contenedores construidos con otras imágenes, que a su vez se basan en diferentes plataformas o sistemas operativos que hace que sea bastante dinámico el uso de las mismas y que estas se ajusten a diferentes necesidades, la herramienta derivada de esta plataforma de servicios de contenedores que más se utiliza para realizar entornos de integración y entrega continua mediante el establecimiento de micro servicios es Docker compose, el cual permite levantar una red de contenedores interconectados para formar el entorno de integración y entrega continua a través de diferentes servicios [13].

Podman.io

Podman.io es un proveedor de contenedores muy similar a Docker, el cual a diferencia de Docker no utiliza un demonio de ejecución para poder funcionar, es de código abierto y tiene una amplia comunidad que está muy integrada con el desarrollo de la herramienta e implementación de funcionalidades, Podman presenta ciertas compatibilidades con Docker, ya que permite crear contenedores a partir de Dockerfiles por ejemplo, también posee una amplia gama de imágenes que proporciona a través de una API, al igual que Docker, Podman permite subir imágenes a su repositorio de imágenes que hayan sido creadas a partir de un Podmanfile por cualquier usuario. Podman está tomando fuerza entre los competidores de Docker, sin embargo, aún es relativamente una herramienta

nueva que tiene un gran potencial, pero que aún tiene camino por delante y se encuentra en constante desarrollo [14].

RKT (Rocket)

Es un motor de tecnología de contenedores desarrollado de manera principal para entornos de producción modernos basados en la nube, se puede integrar con Kubernates de Google, ya que este se encarga de la orquestación de contenedores de algunos tipos incluido RKT, a diferencia de Docker, RKT no posee un demonio central para su ejecución, sino que tiene un pod que se ejecuta directamente en el modelo de proceso clásico de Unix y es una herramienta presentada de manera inicial por CoreOs en Diciembre de 2014, es además una herramienta de código abierto que se encuentra disponible en Github [15].

LXC (Linux containers)

Se trata de una tecnología de contenedores desarrollada por Linux y otros colaboradores con el objetivo de proveer un entorno neutral para el desarrollo de contenedores Linux, a través de una API y otras herramientas permite crear y administrar contenedores utilizando principalmente el kernel de Linux, está formado con algunos componentes separados tales como Python, Go, Ruby, y otras herramientas que permiten controlar los contenedores, LXC también es de código abierto con licencia GNU y cuenta con soporte extendido [16].

1.2.4.2 Repositorios de código fuente.

Los repositorios de código son herramientas utilizadas por los desarrolladores de software para llevar un control, un versionamiento y registro de cambios del código fuente de software, facilita el desarrollo en equipo dado que permite la integración de los cambios por medio del uso de ramas, por medio de las cuales se lleva una organización de las diferentes áreas de trabajo en el desarrollo del software, esto también permite que el código fuente de un proyecto descanse en un lugar seguro en la nube y de esta manera también salvaguardar que el mismo no se pierda en el caso de un cyber ataque a los computadores donde se desarrolla

el software, por daños en el hardware o software de los computadores o servidores de la empresa, institución o personales en donde se desarrolla el software [17].

Un sistema de control de versiones tiene un sistema de archivos que organiza las copias de un mismo archivo en distintas versiones, hasta llegar a su versión final, esta es una forma de llevar un registro de los cambios, en donde se puede regresar a un registro de una versión anterior para hacer uso de una línea de código en específico, poder justificar cambios con base en esos registros, entre otros muchos usos que permiten estas herramientas [18].

Existen varios tipos de repositorios de código fuente, pero a día de hoy los más populares y utilizados a nivel mundial son Git y Mercurial, tanto para un SCM (source control managment) como para otro existen diferentes opciones de herramientas, en esta investigación se evaluarán las siguientes:

Github

Es el repositorio de código que recientemente ha sido comprado por Microsoft, es uno de los repositorios con mayor comunidad y popularidad del mundo, con más de 40 millones de usuarios, y más de 2.1 millones de empresas y organizaciones utilizándolo en la actualidad, hasta hace poco la versión gratuita solo permitía tener repositorios públicos y si se deseaba hacerlos privados tenía un costo, sus planes gratuitos son limitados en ciertos aspectos pero el número de repositorios tanto públicos como privados que se pueden crear son ilimitados, los demás planes que poseen costo están orientado a empresas u organizaciones.

Las características más importantes de Github son su capacidad de permitir CI/CD con su integración con varias herramientas para este fin, análisis de código automático nativo, control de versiones, integración con visual studio code, seguridad del código, entre otros [17].

Bitbucket

Antes de que Github permitiera la creación gratuita de repositorios privados, Bitbucket ya lo hacía desde mucho antes, esta plataforma permite el control de versiones, revisión del código fuente, permite CI/CD, se integra con otras herramientas de su casa (Atalassian) tales como Jira y Trello, la gestión de las ramas de desarrollo, entre otras funcionalidades, sus precios son bastante flexibles y también cuenta con una amplia comunidad detrás, con millones de usuarios, su código no es de acceso abierto a diferencia de Github [18].

Gitlab

Gitlab además de ser un repositorio de código y administrador de versiones, también posee funcionalidades para realizar todo el ciclo de DevOps, pero desde el enfoque de su función de SCM (source code managment) permite la colaboración del equipo de desarrollo mediante la integración de código mediante ramas, al igual que Bitbucket y Github se encuentra basado en GIT, permite el rastreo de código a través del sistema de versionamiento, emisión de gráficos e informes sobre el versionamiento, entre otras muchas funcionalidades que provee Gitlab en su función como SCM [19].

1.2.4.3 Micro servicios

Los micro servicios parten de la tecnología de contenedores, debido al dinamismo que estos permiten, se pueden generar redes de contenedores para ser servicios para las aplicaciones, estos servicios están interconectados para lograr hacer que funcione una aplicación, esto anteriormente se realizaba de manera en que en el mismo entorno virtual se instalaban todos los servicios y esto hace que la aplicación sea más vulnerable y que por ejemplo en un ataque contra el servidor todos estos servicios se caigan, esto se soluciona a través del uso de los micro servicios también.

Los micro servicios además pueden estar escritos en diferentes lenguajes de programación ya que lo que exponen son las API's para comunicarse entre servicios, también pueden exponer estas API's para ofrecer el servicio de manera externa hacia el público en general o hacia su consumo por parte de otras empresas, los servicios pueden ser de distintos tipos y para diferentes usos, por ejemplo para albergar una base de datos, para plataformas de control de calidad de código, para servidores de automatización u otros tipos de servidores, entre otros [20].

Los micro servicios son un enfoque en la arquitectura de software, por otro lado, está el enfoque tradicional, llamado "monolítico", como se había explicado antes, en este enfoque los servicios se encuentran integrados en el entorno principal y uno de los problemas de este enfoque es que al estar todo unido, en el momento en el que se realice un cambio en alguna parte de la aplicación, se deberá volver a desplegar en producción todo el código aunque no se hayan realizado cambios en los demás módulos, lo cual es ineficiente y puede tomar mucho más tiempo que al realizar el cambio a un servicio de la red de micro servicios del otro enfoque, ya que estos se pueden desplegar de manera independiente y con mayor rapidez.

1.2.4.4 Servidores de automatización

Los servidores de automatización como su nombre lo indica, automatizan procesos dentro de un servidor o de manera externa a través de comandos que se ejecutan de manera local o remota, por medio de llaves ssh establecen una conexión y en base a los permisos que se le otorgue al usuario que se le ha dado a estos servidores tendrán mayor o menor libertad de realizar acciones en pos de automatizar un proceso que se desencadena a partir de eventos de distintos tipos, el más común es a través de la sucesión de eventos en un repositorio de código, con lo cual se genera la integración continua.

Para la integración y entrega continua los servidores de automatización son una herramienta fundamental, ya que sin ellos los procesos de entrega e integración no serían automatizados, con lo cual no es continua, como se mencionó anteriormente estos servidores permiten escuchar eventos que se han realizado en

los repositorios de código, esto se hace mediante extensiones que se instalan en estos servidores para poder conectarse con estos repositorios siempre y cuando se proporcionen las credenciales necesarias cuando se trata de repositorios privados, luego se configura el "trabajo" que tiene que realizar el servidor de automatización cuando se gatille el evento en el repositorio, ahí es cuando a través de comandos se le indica a la máquina que algoritmo debe llevar a cabo, en cuyo caso se desencadena la entrega continua de software mediante la conexión por medio de SSH al servidor donde se despliega el código [21].

Algunos servidores de automatización que se analizarán en esta investigación son los siguientes:

Jenkins

Es un servidor de automatización extensible ya que posee un repositorio de plugins por medio de los cuales se integra con herramientas que le permiten realizar más tareas y procesos de automatización en integración con dichas herramientas, se puede utilizar para automatizar ciertos procesos o ser el centro de integración y entrega continua para un proyecto a través de la integración con repositorios de código y su conexión por SSH con servidores donde se entrega el código fuente [22].

Azure DevOps

Es una herramienta para realizar integración continua, que los equipos de desarrollo compartan código, entre otras funciones, sin embargo, es una herramienta de uso privado, proporciona una versión gratuita, pero es por tiempo limitado, también provee de un servicio de alojamiento de GIT ilimitado para hacer solicitudes de extracción, además de herramientas ágiles, esta herramienta no está enfocada en la automatización, pero permite la integración continua debido a su integración con GIT [23].

GitLab

GitLab posee una serie de herramientas que hace que, todo el ciclo de DevOps se pueda realizar con esta plataforma, es así que al ser un repositorio no es necesario conectarse con uno, lo cual permite la integración y entrega continua de manera directa y más sencilla que con otras herramientas, si bien es una plataforma que se puede usar de manera gratuita, este paquete gratuito es bastante limitado y sus opciones de paga tienen muchas más funcionalidades y características que incluyen integración y entrega continua, implementación progresiva, entre otras cosas [24].

Bamboo

Bamboo es una plataforma derivada de Bitbucket que es un versionador de código fuente al igual que Github o GitLab, al igual que GitLab al tener acceso directo al repositorio donde se encuentra el código fuente, la integración y entrega continua se realiza de manera directa para este repositorio, hace una competencia directa con Jenkins que es el servidor de integración y entrega continua más popular y se ha establecido a lo largo de los últimos cinco años, además se integra de manera directa con Jira, la herramienta de gestión de proyectos de la misma empresa que Bitbucket y Bamboo, Atlassian, al igual que GitLab y Azure es una herramienta pagada, pero a diferencia de GitLab este no tiene un plan gratuito, sólo posee un plan bastante económico y otro enfocado en el uso empresarial y a gran escala de la herramienta [25].

1.2.4.5 Plataformas de control de calidad.

Un analizador de estático de código forma parte de un entorno de integración y entrega continua para asegurar la calidad del código mediante una serie de pruebas que se le realizan al código fuente que se sube al repositorio que detecta una serie de errores y malas prácticas de programación que se deben corregir antes de liberar una nueva versión de la aplicación que se trata de desplegar, por lo general

estos analizadores de código soportan una gran cantidad de lenguajes de programación, es necesario verificar que la herramienta que se trata de utilizar soporta el lenguaje que se utiliza en el desarrollo de las aplicaciones que se desean implementar, los diferentes análisis estáticos de código fuente que estas herramientas implementan son las siguientes:

- Análisis de alias
- Análisis de punteros
- Análisis de figuras
- Análisis de escape
- Análisis de acceso a vectores
- Análisis de dependencias
- Análisis de control de flujo
- Análisis de flujo de datos
- Análisis de variable viva
- Análisis de expresiones disponibles
- Análisis de uso definido de cadena

Estos analizadores estáticos de código detectan cosas como código duplicado, código redundante, código muerto, complejidad de métodos innecesarios, entre otros, cosas que por lo general se le puede pasar por alto a un programador pero que le resta calidad al código en tiempo de ejecución, por lo tanto este tipo de herramientas puede detectar este tipo de problemas así como errores graves de compilación y ejecución, detecta incluso malas prácticas de programación, todo en pos de conseguir el desarrollo e implementación de código fiable y de calidad [4].

Algunas de las herramientas para análisis estático de software que se analizan en esta investigación son las siguientes:

SonarQube

Es un analizador estático de código fuente multilenguaje, soporta actualmente 27 lenguajes de programación, permite encontrar problemas con el código antes de

que este se integre y entregue en un entorno de CI/CD, esta herramienta permite encontrar y reparar problemas tales como errores lógicos, de sintaxis, código redundante, código duplicado, código demasiado complejo, entre otros.
Esta herramienta también permite encontrar vulnerabilidades de seguridad como inyección SQL, contraseñas codificadas y errores mal controlados, se pueden personalizar las reglas para el análisis de código pese a que las configuraciones por default funcionan bien para la mayoría de proyectos. Además, Sonarqube se integra con varios motores de integración continua, tales como Jenkins, Azure DevOps, Bamboo, TeamCity, entre otros [26].

PMD

Es un analizador estático de código al igual que Sonarqube, también trabaja con varios idiomas solo que en este caso los idiomas o lenguajes de código no son tan extensos, algunos de los lenguajes más populares que analiza esta plataforma son: Java, C, C++, PHP, Python, Go, Ruby, Matlab, entre otros. Algunas de las fallas que detecta en el análisis de código son las variables no utilizadas, bloques vacíos, creación innecesaria de objetos, detección de código innecesario, código duplicado, entre otros. Es una herramienta de uso gratuito y de fácil instalación para Linux y MacOs, pero no para Windows [27].

Veracode

Es también un analizador estático de código fuente, tiene un gran enfoque en seguridad de código, escaneando vulnerabilidades de seguridad, previniendo piratas informáticos y otras amenazas maliciosas, una característica importante que es permite el escaneo en entornos de CI/CD tanto de desarrollo como de producción, posee una gran variedad de tipos de escaneo, tales como pruebas de alcance, velocidad, pruebas en aplicaciones móviles, web o escritorio, permite probar múltiples aplicaciones a la vez de manera simultánea sin hacer colas, algunos de los lenguajes más importantes que soporta son Java, PHP, Scala, Kotlin, Perl, Python, Visual C++, Cobol, entre otros [28].

1.2.4.6 Pruebas automatizadas de código.

Las plataformas de automatización integran herramientas para testeo de código de muchos tipos, haciendo que en lugar de utilizar una cantidad de distintas herramientas para probar código, se haga por medio de una sola plataforma que incluya todos esos tipos de pruebas que se deben realizar al código [28], algunos de los tipos de pruebas que estas plataformas permiten realizar son los siguientes:

- Pruebas de extremo a extremo
- Pruebas de integración
- Pruebas unitarias
- Pruebas de aceptación
- Pruebas basadas en datos (tdd's)
- Pruebas de API rest
- Entre otros….

Además, estas plataformas de pruebas automatizadas de código incluyen otras herramientas para pruebas de código que a su vez se especializan en tipos concretos de pruebas, cada plataforma integra diferentes herramientas y tipos de pruebas de código automatizado, algunas de estas plataformas son:

Cypress.io

Cypress es una plataforma para realizar pruebas automatizadas de código que tiene una comunidad bastante amplia que la respalda, es una plataforma de código abierto lo cual permite que la comunidad implemente funcionalidades especiales como extensiones que permitan mejorar el soporte con algunos lenguajes, entre otros usos, Cypress hace énfasis en que está construido desde cero, ya que otras plataformas toman como base a Selenium para su construcción, por otro lado Cypress está conformado por otras herramientas para ejecutar varios tipos de pruebas, algunas de estas herramientas son Moca, ChaiJS, ChaiJQuery, Sinon.JS, SinonChai, entre otros [29].

Algunos de los tipos de pruebas que permite ejecutar Cypress son:

- Pruebas de punta a punta
- Pruebas de integración
- Pruebas unitarias

OpenTest

Es una plataforma de pruebas automatizadas de código fuente, es de código abierto al igual que Cypress, de uso gratuito y permite realizar pruebas tanto para aplicaciones web, Rest API's y aplicaciones móviles, a diferencia de Cypress, OpenTest si usa Selenium para realizar sus pruebas web, integra para sus pruebas de apps móviles con Appium, realiza pruebas basadas en palabras clave, pruebas de Rest API's con Apache Http client, entre otras funcionalidades, tiene una instalación bastante sencilla y su código fuente se encuentra en Github [30].

Robot Framework

Esta es una plataforma para pruebas de código también de código abierto, para diferentes tipos de pruebas de código como pruebas de aceptación y automatización de procesos robóticos, las funcionalidades para realizar pruebas de código de esta herramienta se pueden ampliar utilizando librerías de extensiones de Python o Java, el código de esta herramienta se encuentra alojado en Github, bajo licencia Apache 2.0 y la mayoría de sus bibliotecas también son de código abierto [31].

1.2.4.7 ISO/IEC 25010.

La ISO 25010 es un modelo a través del cual se puede evaluar la calidad de un producto de software a través de la determinación de una serie de características que determinan la calidad de un determinado producto de software, es así que la calidad del mismo se refiere a el grado con el que el software satisface las necesidades del usuario agregando valor, para ello se analizan determinadas características que ayudan a evaluar y poder calificar de manera formal un producto de software, la ISO 25010 evalúa aspectos como la funcionalidad, el rendimiento, seguridad, mantenibilidad, entre otros aspectos importantes que determinan la calidad de un producto de software [32].

Figura 3. Características de calidad evaluadas por la ISO/IEC 25010 [32]

Como se puede apreciar en la *Figura 3* la ISO 25010 se basa en la evaluación de ocho aspectos fundamentales como es la parte funcional, el desempeño, la compatibilidad, usabilidad, fiabilidad, seguridad, mantenibilidad y portabilidad, cada uno de los cuales se subdivide en más de un aspecto, cada una de estas características se deben evaluar para conocer la calidad del producto de software, y de este modo determinar o seleccionar una herramienta de manera formal.

CAPÍTULO II: MATERIALES Y MÉTODOS

2. TIPOS DE INVESTIGACIÓN

La investigación se basó en la evaluación de las herramientas utilizadas para el desarrollo de un entorno de integración, entrega y despliegue continuo de software, es por esta razón que se necesitó tener una vista general de distintos aspectos y características de estas herramientas, además de la selección de algunas de ellas para realizar un entorno funcional de CI/CD, por este motivo la investigación se calificó como descriptiva, aplicada, mixta, y transversal.

Descriptiva dado que se tuvo describir las características de cada herramienta que es objeto de análisis de esta investigación, para brindar una visión general y evaluar sus funcionalidades, aplicada en el sentido de que se tomaron avances tecnológicos ya desarrollados y ampliamente utilizados para realizar las evaluaciones y proponer algunas de ellas para la conformación de un entorno de CI/CD basado en estas evaluaciones, mixta ya que a través del análisis de investigaciones realizadas en los últimos años se obtuvo información que permitió realizar un buen análisis de cada aspecto de cada tipo de herramienta que se evalúa para luego aplicarlas en la creación del nuevo entorno que se utilizó para el desarrollo y puesta en producción del sistema médico de la PUCESE y por último transversal dado que el estudio se llevó a cabo para el sistema médico de la universidad y por un periodo de tiempo de 8 meses.

2.1 DESCRIPCIÓN DEL LUGAR

El proyecto se realizó en el departamento de TICS de la Pontificia universidad Católica del Ecuador con sede en Esmeraldas, ya que el entorno de integración y entrega continua se implementó como base para el sistema del departamento médico de la PUCESE, el cual se encarga de administrar la información de los pacientes que se hacen atender en este departamento, que comprende personal docente, administrativo, estudiantes y personal auxiliar que labora dentro de la institución.

2.2 MÉTODOS

Los métodos de análisis científicos que se utilizaron son analítico, sintético y experimental. El método analítico para poder realizar un análisis exhaustivo de cada una de las variables, características, funcionalidades y particularidades de las herramientas objetos de evaluación de esta investigación, dado que se descompone en las partes que conforman un entorno de CI/CD para poder realizar dicho análisis.

Se utilizó el método sintético por el hecho que a partir del análisis se seleccionaron un conjunto de herramientas para la conformación de un nuevo entorno de CI/CD para el desarrollo e implementación del sistema médico de la PUCESE, y por último experimental ya que a través de la experimentación, prueba y error de la implementación de las herramientas, su configuración e integración, se consiguió un entorno funcional que brinda servicios en un servidor donde las variables se encuentran controladas.

Adicionalmente se utilizó una entrevista semi estructurada para poder consultar los requerimientos del departamento de sistemas de la Pontificia Universidad Católica Sede Esmeraldas, que es en donde se implementó el entorno de CI/CD en base a la evaluación de las herramientas de CI/CD y en base a los requerimientos que el departamento tenga en cuanto a herramientas pre establecidas que se manejan dentro del departamento, tales como repositorios de código, tecnología de contenedores de software y plataformas de control de calidad.

2.3 INSTRUMENTOS

La investigación utilizó como herramienta la ISO 25010 [32], para realizar la evaluación de las herramientas tomando de las 8 características expuestas en la *Figura 3*, se han tomado en consideración para el desarrollo de esta investigación adecuación funcional, usabilidad y mantenibilidad. Características de las cuales se ha escogido en base a los requerimientos del departamento de TICS de la PUCESE las siguientes sub características:

- Completitud funcional: Grado en el cual el conjunto de funcionalidades cubre todas las tareas y los objetivos del usuario especificados.

  En este apartado para los servidores de automatización se divide en:

  - Conexión con repositorios.
  - Conexión con servidores remotos por ssh.
  - Plugins de integración con otras herramientas.
  - Pipelines y Jobs.

  Para el apartado de las herramientas de pruebas automatizadas de código se divide en:

  - Tipos de pruebas.
  - Lenguajes soportados para pruebas.
- Capacidad de aprendizaje: Capacidad del producto que permite al usuario aprender su aplicación. Este criterio se aborda para los servidores de automatización y para los marcos de trabajo de pruebas de la siguiente manera:
  - Documentación.
  - Comunidad.
  - Soporte.
- Reusabilidad: Capacidad de un activo que permite que sea utilizado en más de un sistema de software o en la creación de otros activos.

En la siguiente tabla 1 se puede apreciar cómo a partir de estos conceptos se formaron los criterios de evaluación para los servidores de automatización y las herramientas de pruebas automatizadas de código fuente.

Servidores de automatización		
Características	Sub características	Descripción
Completitud funcional	Conexión con repositorios	Conexión de la herramienta con todo tipo de repositorios de versionamiento de código fuente.
	Conexión con servidores por ssh	Facilidad que la herramienta provee para conectarse con servidores remotos para ejecutar comandos.
	Plugins de integración con otras herramientas	Cantidad de plugins de integración con otras herramientas posee.
	Pipelines y Jobs	Facilidades que la herramienta provee para crear Jobs y Pipelines.
Capacidad de aprendizaje	Documentación	Cantidad y calidad de documentación que la herramienta provee desde y para su comunidad.
	Comunidad	Actividad y cantidad de miembros de la comunidad que tiene la herramienta para brindar soporte y documentación.
	Soporte	Soporte que brinda la herramienta de manera oficial y a través de la comunidad para sus usuarios.
Reusabilidad	Capacidad de reusabilidad	Capacidad de utilizar los jobs y pipelines en otros entornos y proyectos.

Tabla 1. Descripción de los criterios de evaluación de servidores de automatización

Pruebas automatizadas de código		
Características	Subcaracterísticas	Descripción
Completitud funcional	Tipos de pruebas	Cantidad de pruebas que el marco puede realizar de manera automatizada.
	Lenguajes disponibles para pruebas	Cantidad de lenguajes que se encuentran disponibles para la realización de las pruebas
Capacidad de aprendizaje	Documentación	Cantidad y calidad de documentación que la herramienta provee desde y para su comunidad.
	Comunidad	Actividad y cantidad de miembros de la comunidad que tiene la herramienta para brindar soporte y documentación.
	Soporte	Soporte que brinda la herramienta de manera oficial y a través de la comunidad para sus usuarios.

Tabla 2. Descripción de los criterios de evaluación de pruebas automatizadas.

A continuación se presentan las escalas de ponderaciones utilizadas para la evaluación de las herramientas de servidores de automatización y de pruebas automatizadas de código fuente construidas en base a la investigación [33], debido a que en la misma se realiza una evaluación de herramientas de software, razón por la cual se ha considerado pertinente aplicar los rangos, escalas y terminología aplicada en esa investigación para adaptarla a la presente investigación, en combinación con los grados de importancia que se han establecido en base a los requerimientos del departamento de TICS de la PUCESE.

Escala	Significado	Grado de importancia
8-10	Criterio relevante	Alto
5-7	Criterio no indispensable	Medio
1-4	Criterio no necesario	Bajo

Tabla 3. Escala de ponderación de los criterios.

Características	Subcaracterísticas	Grado de importancia	Porcentaje
Completitud funcional	Conexión con repositorios	Bajo	5%
	Conexión con servidores por ssh	Alto	15%
	Plugins de integración con otras herramientas	Medio	10%
	Pipelines y Jobs	Alto	15%
Capacidad de aprendizaje	Documentación	Alto	15%
	Comunidad	Alto	15%
	Soporte	Alto	15%
Reusabilidad	Capacidad de reusabilidad	Medio	10%

Tabla 4. Valoración de los criterios de evaluación según su importancia servidores de automatización.

Características	Subcaracterísticas	Grado de importancia	Porcentaje
Completitud funcional	Tipos de pruebas	Alto	25%
	Lenguajes disponibles para pruebas	Bajo	10%
Capacidad de aprendizaje	Documentación	Alto	25%
	Comunidad	Alto	25%
	Soporte	Medio	15%

Tabla 5. Valoración de los criterios de evaluación según su importancia pruebas de código.

CAPÍTULO III: RESULTADOS

3 ANÁLISIS E INTERPRETACIÓN DE LOS RESULTADOS

En este capítulo se detalla el núcleo de la presente investigación, en donde se identifican las herramientas que utiliza el departamento de TICS de la Universidad Católica Sede Esmeraldas y aquellas que van a ser objeto de evaluación aplicando la ISO 25010, posteriormente se realizar la evaluación y comparación de las características de la ISO para cada una de las herramientas propuestas para seleccionar una herramienta de cada tipo para formar y proponer un nuevo entorno de CI/CD como resultado de la investigación.

3.1 IDENTIFICACIÓN DE LAS HERRAMIENTAS DE CI/CD

Antes de poder evaluar las herramientas, realizar una selección e implementar las herramientas seleccionadas es necesario identificar aquellas herramientas pre establecidas que se utilizan dentro del departamento de TICS de la PUCESE y que junto a las herramientas seleccionadas van a conformar un stack de Devops para el entorno de CI/CD del sistema médico de la universidad.

Esta identificación se hace a través de una entrevista semi estructurada dirigida al Mgt. Marc Grob, director del departamento de TICS de la PUCESE, quien nos respondió preguntas que contribuyen a la identificación de las herramientas que son utilizadas dentro del departamento para desarrollar los sistemas de la universidad, a continuación, se presenta los resultados de la entrevista que se puede visualizar en el apartado de anexos:

De los resultados se identificaron las siguientes herramientas:

- Bitbucket como repositorio de código fuente.
- Docker como tecnología de contenedores de software.
- SonarQube como analizador estático de código fuente para control de calidad.

3.2 EVALUACIÓN Y COMPARACIÓN DE LAS HERRAMIENTAS DE CI/CD

Para poder seleccionar las herramientas para formar el entorno de CI/CD es necesario evaluar varios aspectos definidos en la metodología, estos aspectos miden la completitud funcional de las herramientas, la interoperabilidad con otros servicios, la capacidad de aprendizaje de las herramientas y la reusabilidad de los servicios en otros proyectos. Sin embargo, al ser aspectos que no pueden ser medido de manera cuantitativa, se realizó una revisión bibliográfica a fondo de cada herramienta para poder evaluar cada una de las características contempladas en esta investigación.

Las herramientas que se van a evaluar por parte de los servidores de automatización son:

- Jenkins
- Gitlab
- Bamboo

Por parte de las herramientas de pruebas automatizadas de código fuente que se van a evaluar a continuación son:

- Cypress.io
- OpenTest
- Robot Framework

Cabe resaltar que para esta evaluación se han tomado las últimas versiones estables liberadas durante el segundo semestre del 2018.

3.2.1 SERVIDORES DE AUTOMATIZACIÓN

3.2.1.1 Jenkins

3.2.1.1.1 Completitud funcional

Conexión con repositorios

Jenkins soporta varias herramientas de control de versiones, soporta la mayoría y más comunes, principalmente aquellas basadas en Git y Mercurial, pero también utiliza herramientas como CVS, Subversion, Perforce y Clearcase, sin embargo no es compatible con todos los tipos de repositorios [34].

Jenkins	
Característica	Valoración
Conexión con repositorios	9

Conexión con servidores por ssh

Jenkins permite conectarse con servidores remotos por ssh haciendo la configuración del túnel ssh a través de la consola para luego en el Job o pipeline ejecutar las acciones por medio de comandos de Shell que se ejecutarán directamente en el servidor remoto, Jenkins también permite realizar este proceso a través de plugins que se encuentran disponibles en el repositorio de plugins que gestionan las llaves ssh y permiten la conexión por ssh [35].

Jenkins	
Característica	Valoración
Conexión con servidores por ssh	9

Plugins de integración con otras herramientas

Jenkins soporta una gran cantidad de plugins para la conexión con otras herramientas y la gestión de los procesos que este servidor puede automatizar, tales como la conexión con servidores remotos, integración con herramientas de pruebas, entre otros.

Posee más de 1700 plugins que han sido realizados por Jenkins, pero en mayor medida por su comunidad [35].

Jenkins	
Característica	Valoración
Plugins de integración con otras herramientas	10

Pipelines y Jobs

Los pipelines y Jobs forman parte del ecosistema de Jenkins, los Jobs pueden hacerse desde la interfaz gráfica de la herramienta, de la misma manera que los pipelines, sin embargo, estos últimos se pueden hacer también a través de un archivo de configuración con el formato de Jenkins el cual se agrega en la raíz del proyecto, Jenkins al conectarse con el repositorio reconoce el archivo y forma el pipeline [11].

Jenkins proporciona un alto nivel de dinamismo en la construcción de sus pipelines como se puede apreciar en su documentación [11], tiene una muy amplia variedad de herramientas que permiten a los desarrolladores generar pipelines que pueden ser probadas, que pueden ser ejecutadas en paralelo, que tienen compatibilidad con los diferentes lenguajes de programación, entre muchas otras funcionalidades que se encuentran descritas en su documentación.

Jenkins	
Característica	Valoración
Pipelines y Jobs	10

3.2.1.1.2 Capacidad de aprendizaje

Documentación

La documentación de Jenkins en su página oficial se encuentra disponible en inglés y chino, en donde se encuentra información bastante completa sobre la instalación de Jenkins, el uso de pipelines, conexión con repositorios, entre otros.

Además, provee de una serie de tutoriales en donde se muestra cómo realizar algunos ejemplos de casos de uso de la herramienta, así como la documentación para comprender el uso de la sintaxis de Jenkins para la creación de los archivos de configuración de los pipelines.

Fuera de la información oficial de Jenkins se encuentra una gran cantidad de información en la red sobre el uso de Jenkins en diferentes escenarios, así como una

gran cantidad de videos tutoriales en diferentes plataformas de uso gratuito y de pago.

Jenkins	
Característica	Valoración
Documentación	10

Comunidad

La comunidad de Jenkins realiza diferentes tipos de actividades y conforman una parte importante de la herramienta dado que proveen de numerosos plugins multiuso para Jenkins, además se puede conocer a otros desarrolladores dentro de la comunidad, la comunidad provee de un soporte al resolver problemáticas de otros desarrolladores, así como resolver errores de plugins hechos por la comunidad, permite realizar soporte de traducción a otros idiomas, entre otras características de esta comunidad [36].

Jenkins	
Característica	Valoración
Comunidad	9

Soporte

El soporte de la herramienta lo realiza en un primer nivel la comunidad con su foro y su resolución de preguntas frecuentes, errores y problemas. En segunda instancia se puede recibir soporte por parte del equipo de soporte de Jenkins por correo electrónico, contacto a través de la página web, chats entre otros [36].

Jenkins	
Característica	Valoración
Soporte	9

3.2.1.1.3 Reusabilidad

Capacidad de reusabilidad

Jenkins tiene la capacidad de ser extensible a través de la construcción de plugins en la plataforma, puede ser reusable a través de Docker, dado que los contenedores son portátiles entre los entornos y se puede guardar la información de los Jobs en un repositorio, además los pipelines son totalmente reusables y se pueden versionar junto con el código fuente, además Jenkins permite crear servidores en cualquier entorno y utilizar cualquier versión de sus servidores con lo cual se evitan problemas de versiones, lo que lo vuelve altamente reusable.

Jenkins	
Característica	Valoración
Capacidad de reusabilidad	9

3.2.1.2Gitlab

3.2.1.2.1 Completitud funcional

Conexión con repositorios

Gitlab permite la conexión con todo tipo de repositorios, ademas, al ser una herramienta que cumple con varias funciones dentro del ciclo de Devops puede alojar el código fuente de un software en un repositorio propio de Gitlab, dado que Gitlab también es un administrador de repositorios Git [37].

Gitlab	
Característica	Valoración
Conexión con repositorios	10

Conexión con servidores por ssh

Las conexiones con servidores remotos por ssh en Gitlab se realizan a través de ejecutores, que son parte de los corredores de los pipelines de Gitlab, en ellos se configura las conexiones por ssh a servidores remotos y se definen también las acciones a realizar una vez establecida la conexión [38].

Gitlab	
Característica	Valoración
Conexión con servidores por ssh	9

Plugins de integración con otras herramientas

Gitlab proporciona integraciones con multiples herramientas en diferentes aspectos que pueden llegar a ser necesarios en el ciclo de Devops, como herramientas rastreadoras de problemas externas como Jira, Redmine o Bugzilla. Herramientas de autenticación como Kerberos, proveedores de OAuth2 entre otros, integración continua con Jenkins y otros tipos de integraciones.

Estas integraciones no se dan en forma de plugins desarrollados por la herramienta y su comunidad, sino que se da en forma de integraciones que pueden utilizarse sin instalaciones adicionales dentro del uso de la herramienta y que son proporcionadas por Gitlab a diferencia de herramientas como Jenkins [39].

Gitlab	
Característica	Valoración
Plugins de integración con otras herramientas	6

Pipelines y Jobs

Los pipelines y trabajos en Gitlab van de la mano, además Gitlab define un componente adicional que son las etapas, en las cuales se definen cuando se van a ejecutar los Jobs o trabajos que pueden estar de manera independiente o formando parte de un pipeline o tubería. Una serie de trabajos forman parte de una etapa, si estos se ejecutan sin ningún problema se avanza con la siguiente etapa hasta terminar el pipeline, de lo contrario la ejecución se detiene.

Los pipelines se pueden definir a través de la interfaz gráfica o a través de archivos con extensión tipo yml en donde se configuran las etapas con sus respectivos trabajos, corredores y ejecutores [40].

Al igual que Jenkins y tal como se evidencia en su documentación Gitlab provee varias herramientas y funcionalidades para proveer a los desarrolladores, la interfaz gráfica de Gitlab provee al desarrollador visualización de los estados de las ejecuciones de estos pipelines lo cual es una funcionalidad relevante, sin embargo en este punto Jenkins ofrece mayores funcionalidades y provee una mayor capacidad de dinamismo.

Gitlab	
Característica	Valoración
Pipelines y Jobs	9

3.2.1.2.2 Capacidad de aprendizaje

Documentación

Gitlab provee a través de su landing page una amplia documentación detallada sobre todas las funcionalidades que forman parte de la misma, establece guías en donde explica cómo poner en funcionamiento las diversas herramientas para realizar Devops o la creación de un repositorio. Cada una de las integraciones son explicadas para su puesta en marcha, a través de tutoriales y ejemplos de uso en diferentes ambientes.

Además, existe mucha información en línea relacionada a Gitlab, así como información acerca de la resolución de errores relacionados con la herramienta [37].

Gitlab	
Característica	Valoración
Documentación	9

Comunidad

Gitlab provee de una detallada documentación para colaboradores que quieran formar parte de la comunidad, y explica cómo desarrollar funcionalidades extra tanto para usarlas de manera privada como para exponerlas al público, existen dos roles dentro de la comunidad, que son las de usuario en donde se puede realizar informes de abuso, formar parte de discusiones, entre otros.

Su contraparte es el rol de administrador en donde se puede modificar el código fuente de Gitlab que es abierto para desarrollar funcionalidades y personalizarlas a gusto [41].

Gitlab	
Característica	Valoración
Comunidad	10

Soporte

La comunidad a través de la gestión de errores y las discusiones en donde se tratan problemas que los usuarios tienen relacionados con Gitlab, es el primer nivel de soporte, en el segundo nivel de soporte se encuentra los administradores de la comunidad, y en última instancia se encuentra el equipo de soporte técnico de Gitlab.

Cabe resaltar que existen múltiples contactos con Gitlab y que desde su página la manera más sencilla de hacerlo es a través del chat en vivo que se proporciona.

Gitlab	
Característica	Valoración
Soporte	9

3.2.1.2.3 Reusabilidad

Capacidad de reusabilidad

Los documentos que definen los pipelines en Gitlab son altamente reusables y fácilmente modificables para que funcionen con otros proyectos, de igual manera que con Jenkins Gitlab permite guardar las configuraciones y a través de los contenedores desplegar los entornos de CI/CD en otros servidores.

Gitlab	
Característica	Valoración
Capacidad de reusabilidad	9

3.2.1.3 Bamboo

3.2.1.3.1 Completitud funcional

Conexión con repositorios

En la documentación oficial de la casa de Bamboo (Atlassian) se encuentra que los repositorios a los que la herramienta se puede integrar son limitados, los principales repositorios se encuentran dentro de la siguiente lista:

- Bitbucket (Atlassian)
- Git
- Github
- Mercurial
- Subversion
- CVS

Bamboo además ofrece la integración forzosa con repositorios no soportados a través de complementos que permiten la integración, y para ello ofrece una guía en su documentación [42].

Bamboo	
Característica	Valoración
Conexión con repositorios	8

Conexión con servidores por ssh

Bamboo permite una interfaz gráfica para configurar una conexión SSH con servidores remotos para poder ejecutar tareas a través de la línea de comandos, en donde se pueden ejecutar archivos de tipo script para ejecutar una serie de pasos en el servidor remoto, iniciar y detener servicios, y todo lo que se pueda hacer a través de línea de comandos el usuario con el que se ha conectado al servidor remoto, sin embargo este proceso lleva una mayor cantidad de pasos y se vuelve más complejo que en el caso de Gitlab y Jenkins [43].

Bamboo	
Característica	Valoración
Conexión con servidores por ssh	8

Plugins de integración con otras herramientas

Bamboo cuenta con casi 200 plugins que se encuentran a la venta en el Marketplace de complementos para diferentes aplicaciones de Atlassian en su página oficial, cabe destacar que al ser Bamboo una herramienta de paga y que su código es de tipo propietario la comunidad de Bamboo no puede realizar plugins o complementos para integración con otras herramientas, algunos de estos complementos del Marketplace se encuentran actualmente sin soporte y es la comunidad quien puede dara soluciones sobre ciertos tipos de errores [44].

Bamboo	
Característica	Valoración
Plugins de integración con otras herramientas	7

Pipelines y Jobs

Los pipelines en Bamboo no se realizan directamente en la herramienta, Bamboo forma parte un ecosistema complementado con Bitbucket y otras herramientas de Atlassian, es de este modo que, es Bitbucket el que tiene disponible el servicio de creación, administración y ejecución de pipelines, en Bamboo se llevan a cabo procesos como los trabajos e integraciones, además de brindar una visión general del proceso de Devops, estos pipelines se configuran a través de archivos de configuración con extensión .yml y cuenta con templates pre definidos para cada tipo de lenguaje o servicio como PHP o Docker [45].

Las funcionalidades y herramientas que proporciona Bamboo en este punto son básicas e intermedias y existe una brecha entre sus dos competidores.

Bamboo	
Característica	Valoración
Pipelines y Jobs	8

3.2.1.3.2 Capacidad de aprendizaje

Documentación

Bamboo ofrece una documentación bastante completa en su página web, esta está dividida por versiones de la herramienta, en esta documentación se puede encontrar información sobre cómo empezar a utilizar la herramienta, ofrece guías de implementación, detalle sobre buenas prácticas, administración de las funcionalidades, actualizaciones y sus respectivas notas, así como las especificaciones y preguntas frecuentes sobre el uso común de Bamboo [45].

Bamboo	
Característica	Valoración
Documentación	9

Comunidad

Atlassian de manera general cuenta con casi cuatro millones de usuarios, de los cuales en promedio entre diez mil y quince mil usuarios se encuentran en línea, cuenta con más de doscientos grupos y casi tres mil eventos. Esta comunidad se divide para cada uno de los productos de Atalassian como Bitbucket, Jira, Bamboo, entre otros [46].

Bamboo	
Característica	Valoración
Comunidad	9

Soporte

El soporte de Bamboo al igual que con la mayoría de herramientas de software se encuentra principalmente en foros de internet y la comunidad de Atlassian, existe un foro de general de Bamboo y otro foro específicamente para desarrolladores de Bamboo, además a través de la página oficial de la herramienta se puede realizar una solicitud de soporte para lo cual es necesario contar con una cuenta de Atlassian y estar loggeado [46].

Bamboo	
Característica	Valoración
Soporte	8

3.2.1.3.3 Reusabilidad

Capacidad de reusabilidad

Los pipelines realizados como archivos de configuración con extensión. yml son totalmente reusables en cualquier ambiente dentro de Bitbucket o Bamboo, al tener una sintaxis propia de Atlassian estos archivos no son soportados fuera de estas plataformas.

Bamboo	
Característica	Valoración
Capacidad de reusabilidad	7

Una vez realizada la revisión bibliográfica a fondo de las herramientas con el enfoque en las sub características de la norma ISO 25010, se procedió a realizar las correspondientes ponderaciones en base a los resultados de esta revisión, tal y como se especifica en la tabla 6 y 7.

Servidores de automatización				
		Jenkins	Gitlab	Bamboo
Características	Subcaracterísticas	Valoración	Valoración	Valoración
Completitud funcional	Conexión con repositorios	9	10	8
	Conexión con servidores por ssh	9	9	8
	Plugins de integración con otras herramientas	10	7	7
	Pipelines y Jobs	10	9	8
Capacidad de aprendizaje	Documentación	10	9	9
	Comunidad	9	10	9
	Soporte	9	9	8
Reusabilidad	Capacidad de reusabilidad	9	9	7

Tabla 6. Ponderaciones de los servidores de automatización.

Servidores de automatización				
		Jenkins	Gitlab	Bamboo
Características	Subcaracterísticas	Valor porcentual	Valor porcentual	Valor porcentual
Completitud funcional	Conexión con repositorios	4,5%	5%	4%
	Conexión con servidores por ssh	13,50%	13,50%	12%
	Plugins de integración con otras herramientas	10%	7%	7%
	Pipelines y Jobs	15%	13,5%	12%
Capacidad de aprendizaje	Documentación	15%	13,50%	13,50%
	Comunidad	13,50%	15%	13,50%
	Soporte	13,50%	13,50%	12%
Reusabilidad	Capacidad de reusabilidad	9%	9%	7%
Total		93,50%	90,50%	84%

Tabla 7. Ponderación porcentual de los servidores de automatización.

3.2.2 PRUEBAS AUTOMATIZADAS DE CÓDIGO FUENTE

3.2.2.1Cypress.io

3.2.2.1.1 Completitud funcional

Tipos de pruebas

Cypress tiene un enfoque hacia las aplicaciones web que se ejecutan en un navegador, para ello define la ejecución de tres tipos principales de pruebas a realizar a una aplicación web [47]:

- Pruebas de punta a punta
- Pruebas de integración
- Pruebas unitarias

Cypress	
Característica	Valoración
Tipos de pruebas	8

Lenguajes disponibles para pruebas

Las pruebas en Cypress se escriben bajo el lenguaje Javascript, el lenguaje sobre el cual está construido esta herramienta, sin embargo, en su documentación indica que puede realizar pruebas para toda aplicación web que se ejecuta en un navegador independientemente del frontend o backend de esta [48].

Cypress	
Característica	Valoración
Lenguajes disponibles para pruebas	10

3.2.2.1.2 Capacidad de aprendizaje

Documentación

La documentación de Cypress se divide en siete secciones:

- Visión general: explica los motivos principales para seleccionar esta herramienta sobre otras similares.
- Empezando: habla sobre la instalación y primeros pasos después de la instalación de la herramienta.
- Conceptos básicos: explica el funcionamiento de la sintaxis que utilizar la herramienta, la interacción con elementos del frontend de la aplicación web entre otros.
- Tablero: habla sobre el control del tablero que proporciona Cypress para monitorizar las pruebas grabadas y la organización de esta información.
- Guías: como su nombre lo indica proporciona guías sobre cómo realizar pruebas de extremo a extremo, unitarias o de integración, entre otras guías.

- Herramientas: habla sobre complementos y funcionalidades adicionales de la herramienta que permiten entre otras cosas realizar acciones como la integración con IDE's.

- Referencias [48].

Cypress	
Característica	Valoración
Documentación	9

Comunidad

Cypress tiene una página web adicional dedicada a su comunidad, se encuentra disponible en 15 idiomas, se encuentra dividido en secciones como: blog, discusiones recientes, artículos, un ranking de líderes dentro de la comunidad con una valoración independiente y contactos con los miembros de la comunidad [49].

Cypress	
Característica	Valoración
Comunidad	9

Soporte

El soporte de Cypress se basa en el uso de los recursos que la documentación de la herramienta pone a disposición de sus usuarios, tales como las preguntas frecuentes, la publicación de errores o bugs en su repositorio de github, chat en vivo con miembros de la comunidad o el uso de stackoverflow. Sin embargo, no se obtiene soporte directamente desde Cypress dado que no se especifica un contacto con un equipo técnico [49].

Cypress	
Característica	Valoración
Soporte	9

3.2.2.2OpenTest

3.2.2.2.1 Completitud funcional

Tipos de pruebas

OpenTest proporciona una gran variedad de tipos de pruebas de código, tales como:

- Pruebas funcionales
- Pruebas de app móviles
- Pruebas de API REST
- Pruebas basadas en palabras clave
- Pruebas de ejecución paralela
- Pruebas basadas en datos
- Pruebas distribuidas
- Pruebas de extremo a extremo
- Referencia [30].

OpenTest	
Característica	Valoración
Tipos de pruebas	9

Lenguajes disponibles para pruebas

Los lenguajes de programación que constituyen el backend, frontend, frameworks, entre otros son independientes del uso de OpenTest al igual que en el caso de Cypress dado que este framework de pruebas interactúa de manera externa con el aplicativo web, móvil o servicio REST [30].

OpenTest	
Característica	Valoración
Lenguajes disponibles para pruebas	10

3.2.2.2.2 Capacidad de aprendizaje

Documentación

La documentación de OpenTest tiene una estructura simple dividida en tres secciones principales, las cuales son: documentos, referencias y tutoriales. En la sección de documentos se encuentran guías detalladas sobre una gran cantidad de temas, estos temas abordan la instalación, primeros pasos, arquitectura de la herramienta, y diferentes temáticas relacionadas a los tres tipos principales de pruebas que realiza esta herramienta: web, móvil y API's [30].

OpenTest	
Característica	Valoración
Documentación	8

Comunidad

OpenTest no cuenta con una comunidad oficial asociada, cuenta con resolución de errores a través de la página de su repositorio en github, y la información que usuarios proporcionan a través de blogs en línea como stackoverflow o reddit.

OpenTest	
Característica	Valoración
Comunidad	6

Soporte

No se encontró referencias sobre el soporte que brinda OpenTest para su herramienta.

OpenTest	
Característica	Valoración
Comunidad	3

3.2.2.3Robot.io

3.2.2.3.1 Completitud funcional

Tipos de pruebas

Robot.io es un framework de pruebas que permite la escritura y ejecución de pruebas de tipo [31]:

- Pruebas de API REST.
- Pruebas de componentes.
- Pruebas funcionales.
- Pruebas de integración.
- Pruebas de aceptación.

OpenTest	
Característica	Valoración
Tipos de pruebas	9

Lenguajes disponibles para pruebas

Robot.io también es independiente del lenguaje de programación que utilizan las herramientas web, móvil o de escritorio multiplataforma que se desee probar, dado que su contacto es externo. Cabe resaltar que la herramienta se encuentra escrita en el lenguaje Python y su sintaxis es de alto nivel para el desarrollo de las pruebas [31].

OpenTest	
Característica	Valoración
Leguajes disponibles para pruebas	10

3.2.2.3.2 Capacidad de aprendizaje

Documentación

La documentación se encuentra contenida en la página web de Robot framework, la documentación se centra en los temas relacionados con la instalación e introducción de la herramienta y su puesta en marcha, la creación de datos de pruebas, la ejecución de los diferentes casos de pruebas, la extensibilidad del marco de trabajo de Robot, detalle de las herramientas de apoyo y enlaces relacionados [31].

OpenTest	
Característica	Valoración
Documentación	8

Comunidad

La comunidad de Robot framework es reducida, tienen grupos de correos y un foro alojado en Google groups, adicionalmente los usuarios responden preguntas y proponen temas en foros en línea y a través de su página del repositorio en github [50].

OpenTest	
Característica	Valoración
Comunidad	8

Soporte

La principal fuente de soporte de la herramienta es su comunidad, sus foros, grupos de correo y la ayuda de foros externos en línea, adicionalmente un punto de contacto que se expone en la página oficial es su cuenta de twitter [50].

OpenTest	
Característica	Valoración
Soporte	8

Una vez realizada la revisión bibliográfica a fondo de las herramientas con el enfoque en las sub características de la norma ISO 25010, se procedió a realizar las correspondientes ponderaciones en base a los resultados de esta revisión, tal y como se especifica en la tabla 8 y 9.

Pruebas automatizadas				
		Cypress	OpenTest	Robot.io
Características	Subcaracterísticas	Valoración	Valoración	Valoración
Completitud funcional	Tipos de pruebas	8	9	9
	Lenguajes disponibles para pruebas	10	10	10
Capacidad de aprendizaje	Documentación	9	8	8
	Comunidad	9	6	8
	Soporte	9	3	8

Tabla 8. Ponderaciones de pruebas automatizadas de código

Pruebas automatizadas				
		Cypress	OpenTest	Robot.io
Características	Subcaracterísticas	Valoración	Valoración	Valoración
Completitud funcional	Tipos de pruebas	20%	22,50%	22,50%
	Lenguajes disponibles para pruebas	10%	10%	10%
Capacidad de aprendizaje	Documentación	22,50%	20%	20%
	Comunidad	22,50%	15%	20%
	Soporte	13,50%	4,50%	12%
Total		88,50%	72%	84,50%

Tabla 9. Ponderación porcentual de pruebas automatizadas de código

3.3 IMPLEMENTACIÓN DEL ENTORNO DE CI/CD EN EL DEPARTAMENTO DE TICS

Entorno construido con Bitbucket, Docker, Jenkins, SonarQube y Cypress como resultado de las entrevistas realizadas al encargado de sistemas del departamento de TICS de la PUCESE y de la evaluación de las herramientas de servidores de automatización y pruebas automatizadas de código luego de la aplicación de la norma ISO/IEC 25010.

El esquema del entorno de forma básica es este:

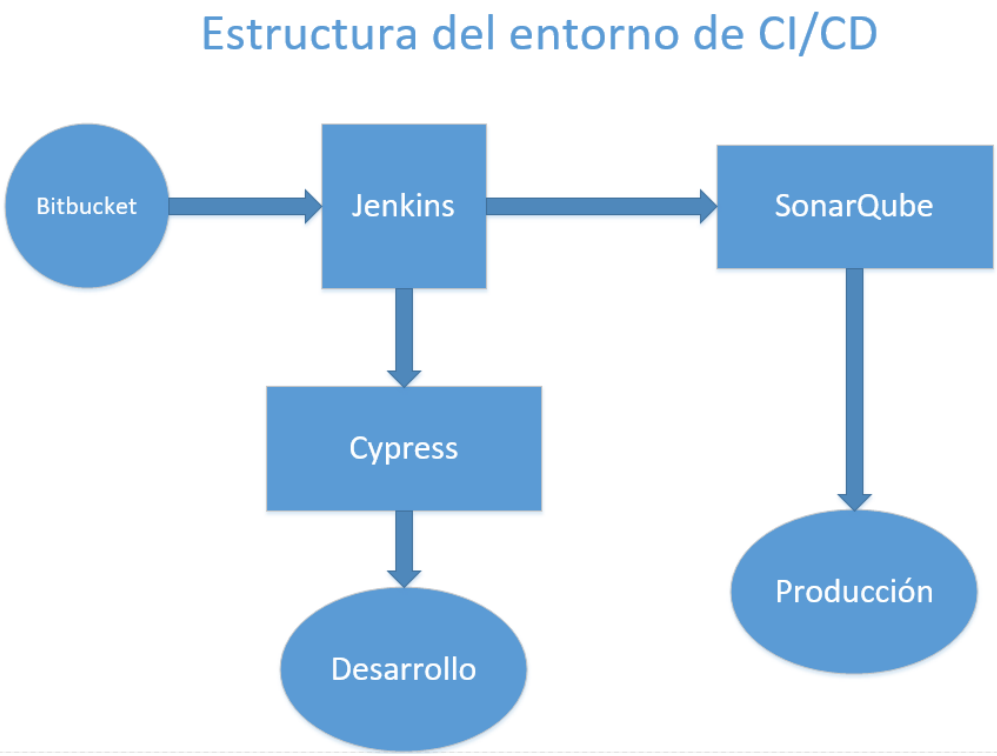

Figura 4. Estructura del entorno de CI/CD

Todo este esquema va sobre un entorno de contenedores de Docker en el cual se levantan los servicios, con sus volúmenes para que exista una persistencia de datos, el archivo Docker-Compose con el que se levantó del entorno es el siguiente:

```
version: "2"
services:
    server:
        image: akiles94/ubuntu-apache-php
        ports:
            - "80:80"
            - "443:443"
        volumes:
            - /htdocs/prasismedi:/var/www/html/
        links:
            - db
        networks:
            - default
    db:
        image: mysql:5.7
        ports:
            - "3306:3306"
        environment:
            MYSQL_ROOT_PASSWORD: ********
        networks:
            - default
    phpmyadmin:
        image: phpmyadmin/phpmyadmin
        links:
            - db:db
        ports:
            - 8000:80
    jenkins:
        image: jenkins/jenkins
        user: root
        ports:
            - 8080:8080
            - 50000:50000
        volumes:
            - /htdocs/jenkins_home:/var/jenkins_home
            - /htdocs/prasismedi:/var/www/html/
    sonarqube:
        image: sonarqube:5.6
        ports:
            - 9000:9000
        volumes:
            - /htdocs/sonar/conf:/opt/sonarqube/conf
            - /htdocs/sonar/data:/opt/sonarqube/data
            - /htdocs/sonar/logs:/opt/sonarqube/logs
            - /htdocs/sonar/extensions:/opt/sonarqube/extensions
```

Ilustración 1. Código del archivo Docker-Compose del entorno de CI/CD.

Para adaptarse a los servidores y requerimientos de la universidad el servicio no se levantó en una sola instancia sino en diferentes instancias dentro del administrador de servidores del departamento de TICS de la PUCESE.

Se crearon 4 instancias en un servidor dedicado al sistema médico de la PUCESE a las cuales se pueden acceder por ssh mediante el puerto 35:

- Base de datos: 172.19.31.5
- DevOps: 172.19.31.6
- Producción: 172.19.31.7
- Pruebas: 172.19.31.4

3.3.1 CONFIGURACIÓN DE CI/CD EN JENKINS.

Jenkins es el eje central que orquesta la integración, entrega y despliegue continuo del software del sistema médico de la PUCESE, a través de este servidor de automatización se configuraron tres trabajos (Jobs) principales en los cuales Jenkins se integra con el servidor de desarrollo, el servidor de producción, pruebas automatizadas con Cypress y SonarQube como se puede visualizar en la Figura 5.

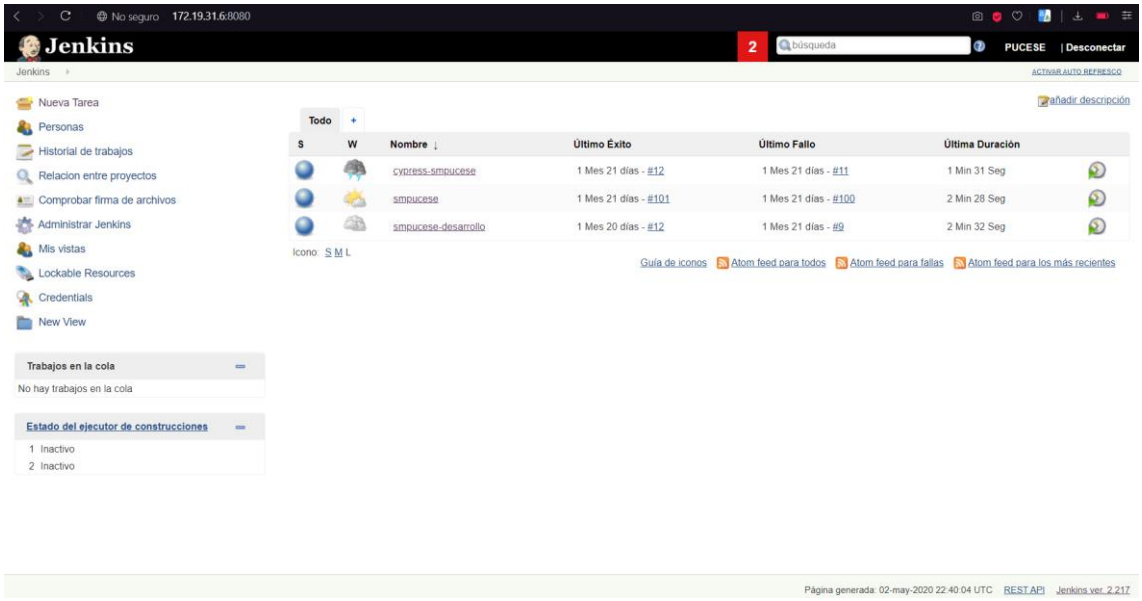

Figura 5. Dashboard de Jenkins, listado de trabajos (Jobs).

Para que Jenkins se integre con SonarQube hace falta ir a la configuración del sistema de Jenkins, ingresar al apartado de plugins e instalar el plugin de SonarQube para Jenkins. Luego es necesario configurar Jenkins ingresando la URL del servidor de SonarQube para realizar la integración como se muestra en la Figura 6. Una vez

configurado el plugin de SonarQube para Jenkins se puede ejecutar el escáner de SonarQube sobre un directorio determinado a través de los trabajos de Jenkins.

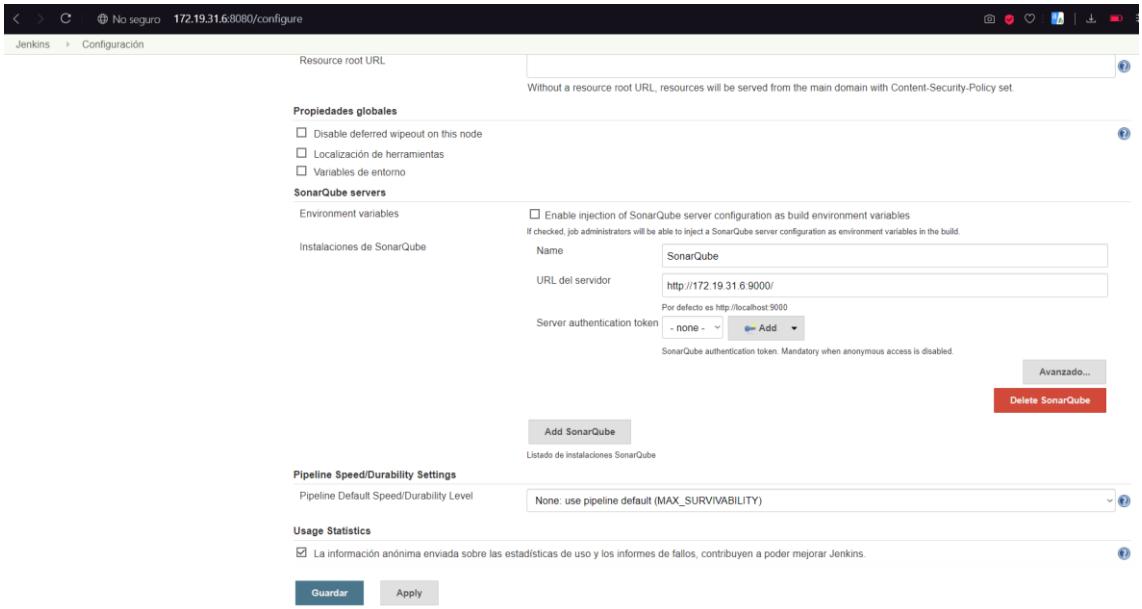

Figura 6. Configuración del plugin de SonarQube con Jenkins.

Como se puede ver en la Figura 7, los trabajos de Jenkins poseen un listado de construcciones o ejecuciones de los trabajos a modo de historial, desde el cual se puede revisar cada uno con detenimiento, para observar de manera rápida lo que sucede con cada uno de ellos se tienen indicadores que representan el estado de esa ejecución en particular, si es de color azul significa que la ejecución fue satisfactoria, si es de color rojo significa que la ejecución fue fallida y si está en color negro significa que la ejecución fue abortada. También se pueden realizar una serie de acciones desde un panel de opciones sobre el historial en donde destacan las opciones de “construir ahora” y “configurar”, en los cuales se puede ejecutar al instante el trabajo actual y configurar los detalles del mismo respectivamente.

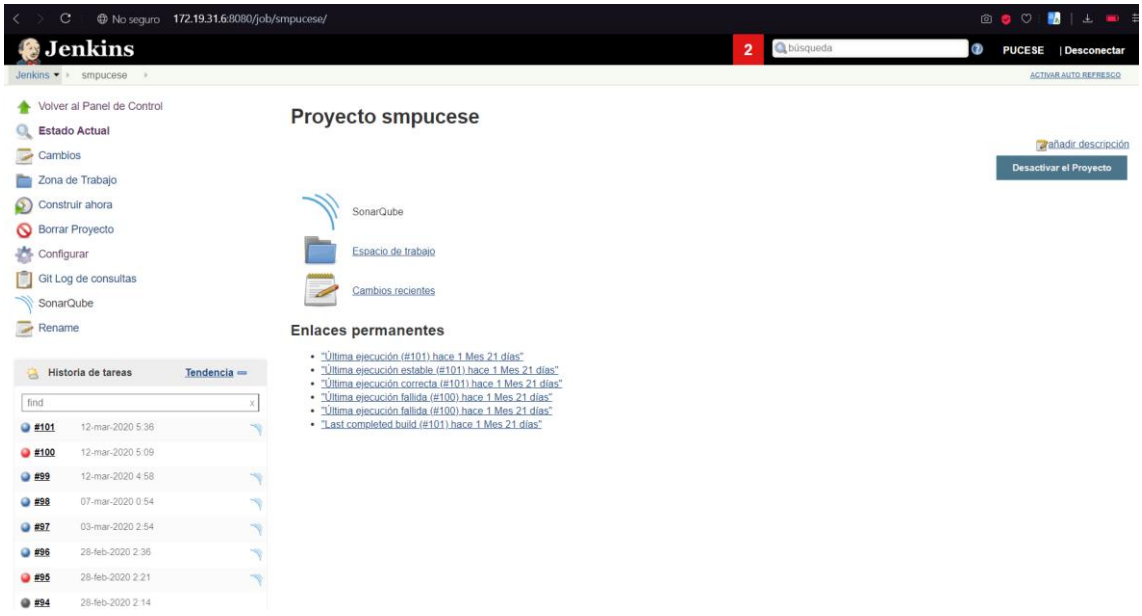

Figura 7. Panel de control de un trabajo en Jenkins.

En las Figura 8, Figura 9 y Figura 10 se puede evidenciar la configuración del trabajo correspondiente al servidor de producción del sistema médico PUCESE, en el cual se integra con el repositorio de código fuente para escuchar un evento de cambios en el repositorio que se estará revisando cada quince minutos, con nomenclatura de Jenkins (H/15 * * * *), en el cual cada vez que se realice un cambio en el mismo se establece una conexión por SSH con el servidor de producción remoto en el cual se tiene el código fuente que ejecuta el software a través de un servidor web de tipo apache. En donde se ejecutan dos acciones principalmente correspondientes a la fase de entrega de CI/CD:

- Ejecución de escáner de SonarQube para control de calidad.
- Entrega de los cambios de la rama de desarrollo del repositorio hacia el servidor de desarrollo.

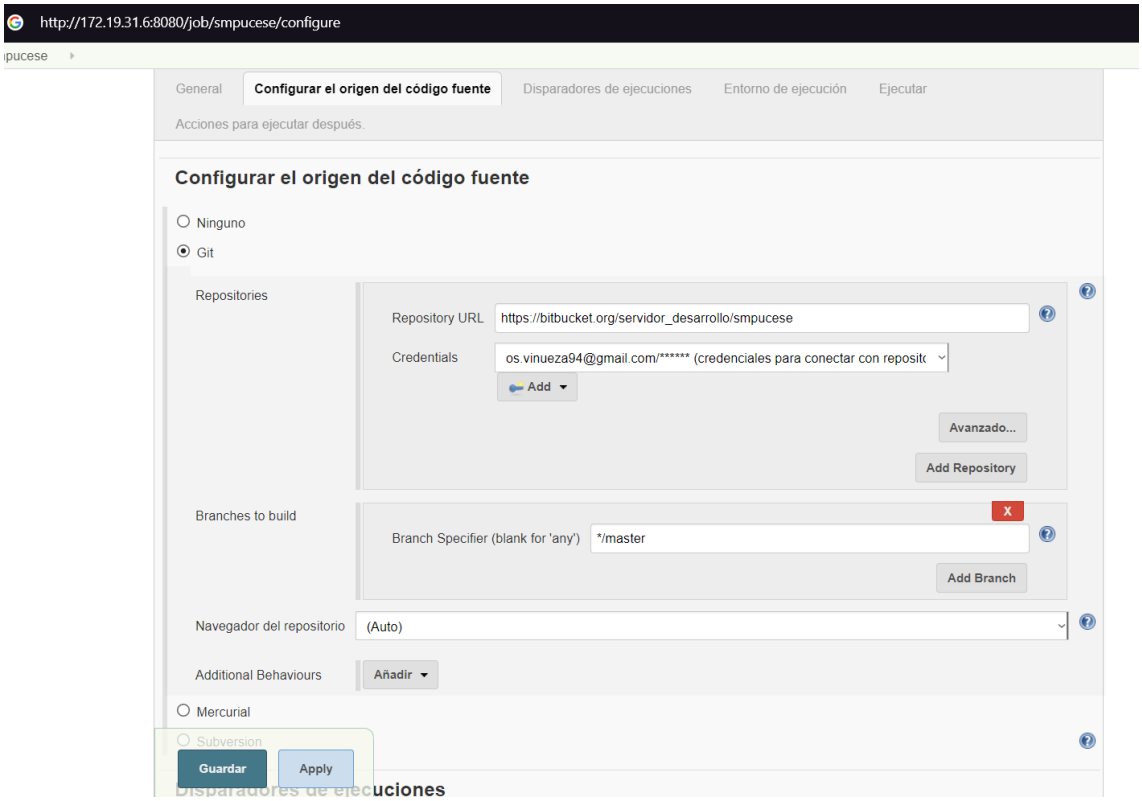

Figura 8. Integración de Jenkins con repositorio de código fuente en Bitbucket.

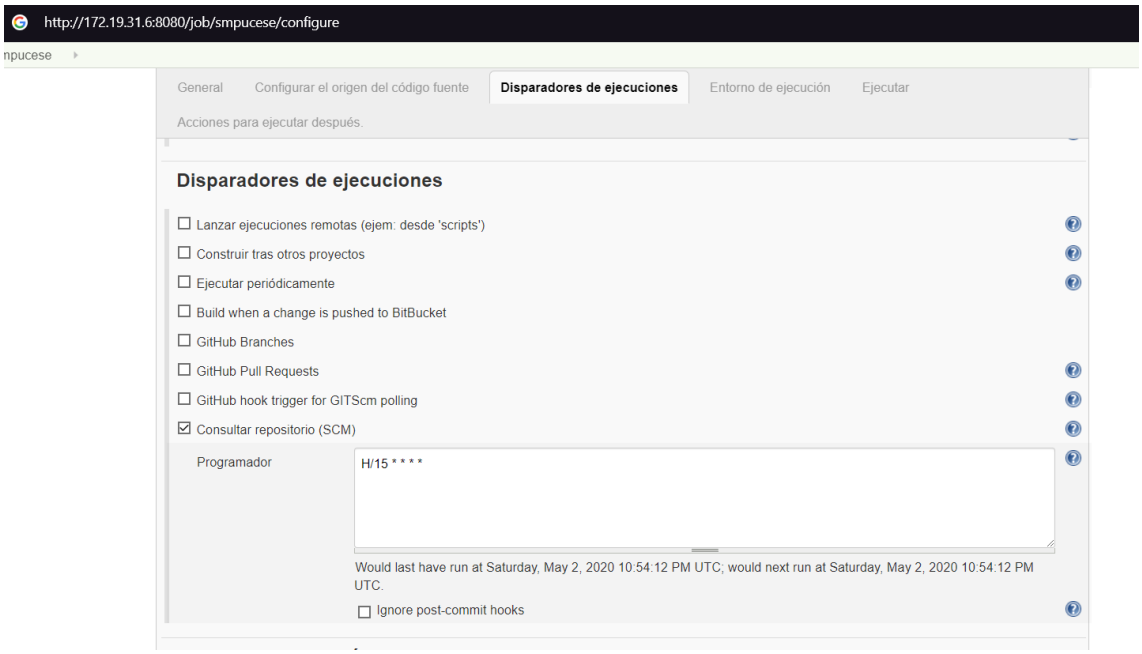

Figura 9. Configuración del disparador para escucha de eventos en repositorio SCM.

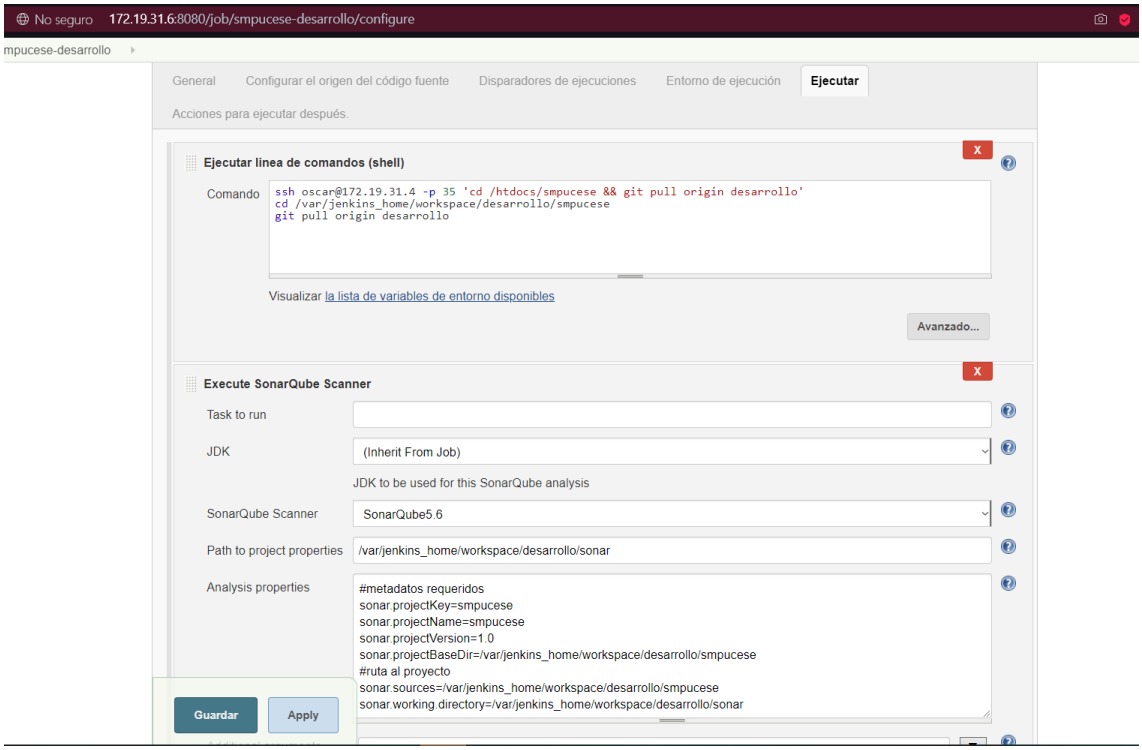

Figura 10. Conexión por SSH con servidor remoto de desarrollo o de producción.

Cabe resaltar que el mismo procedimiento de trabajo para la conexión con el servidor remoto de desarrollo se realiza para la conexión y despliegue de la rama de producción en el repositorio hacia el servidor de producción. El resultado de estas ejecuciones se visualizó a través de las salidas de consola como se aprecia en la Figura 11 y Figura 12.

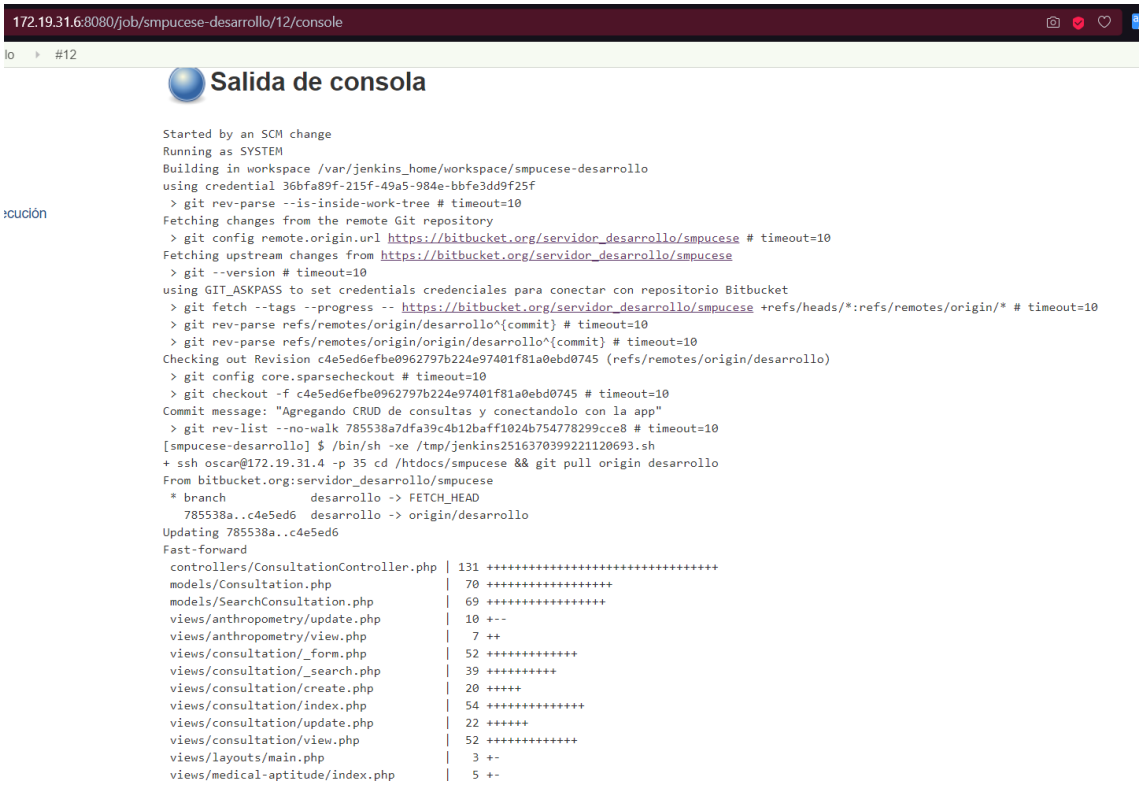

Figura 11. Salida de consola de ejecución de trabajo de entrega y despliegue 1.

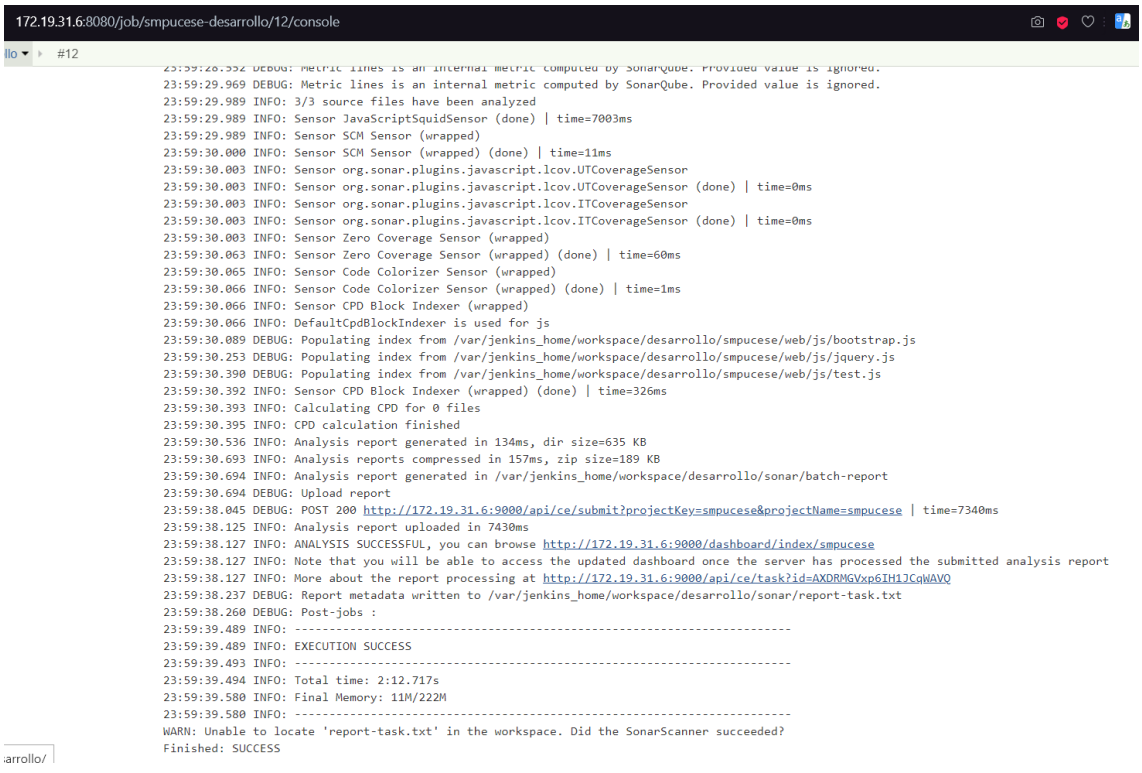

Figura 12. Salida de consola de ejecución de trabajo de entrega y despliegue 2.

Por último, en las Figura 13 y Figura 14 se puede evidenciar que el tercer trabajo configurado en el servidor de automatización Jenkins para culminar la fase de despliegue de CI/CD es aquel que se encarga de realizar las pruebas de extremo a extremo al código fuente cada vez que se realiza un cambio al repositorio que contiene el código fuente de las pruebas automatizadas.

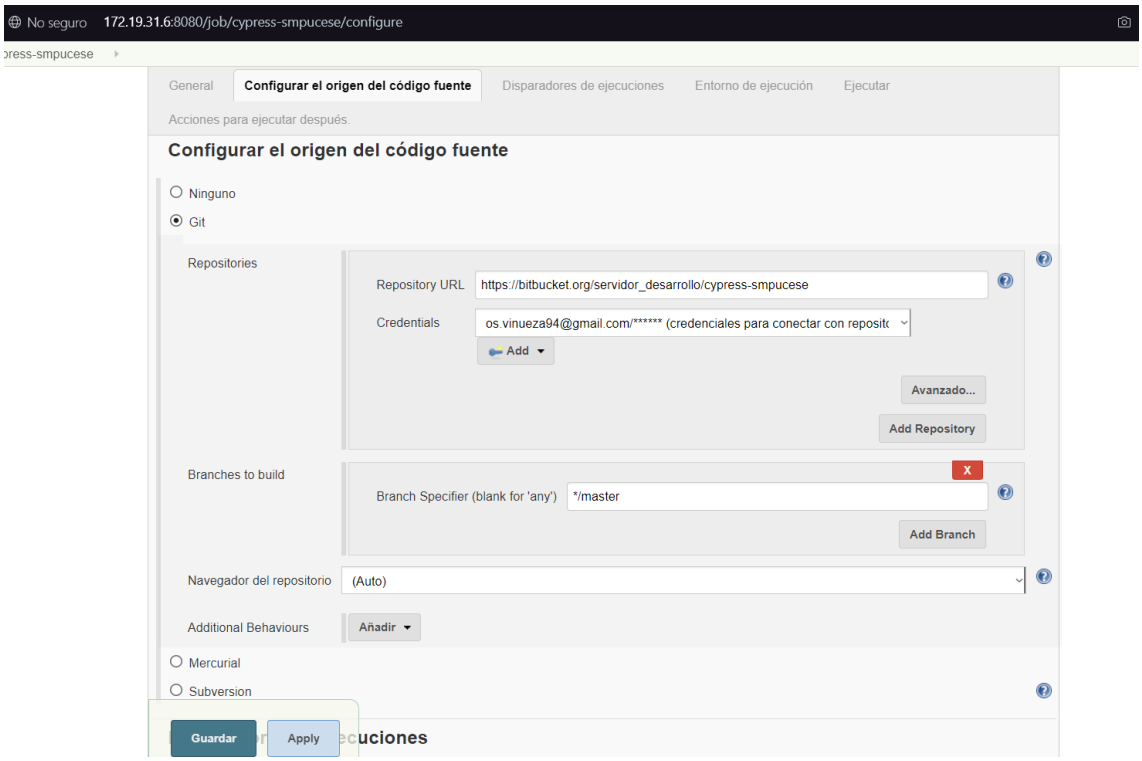

Figura 13. Integración con repositorio con el código fuente de las pruebas de extremo a extremo.

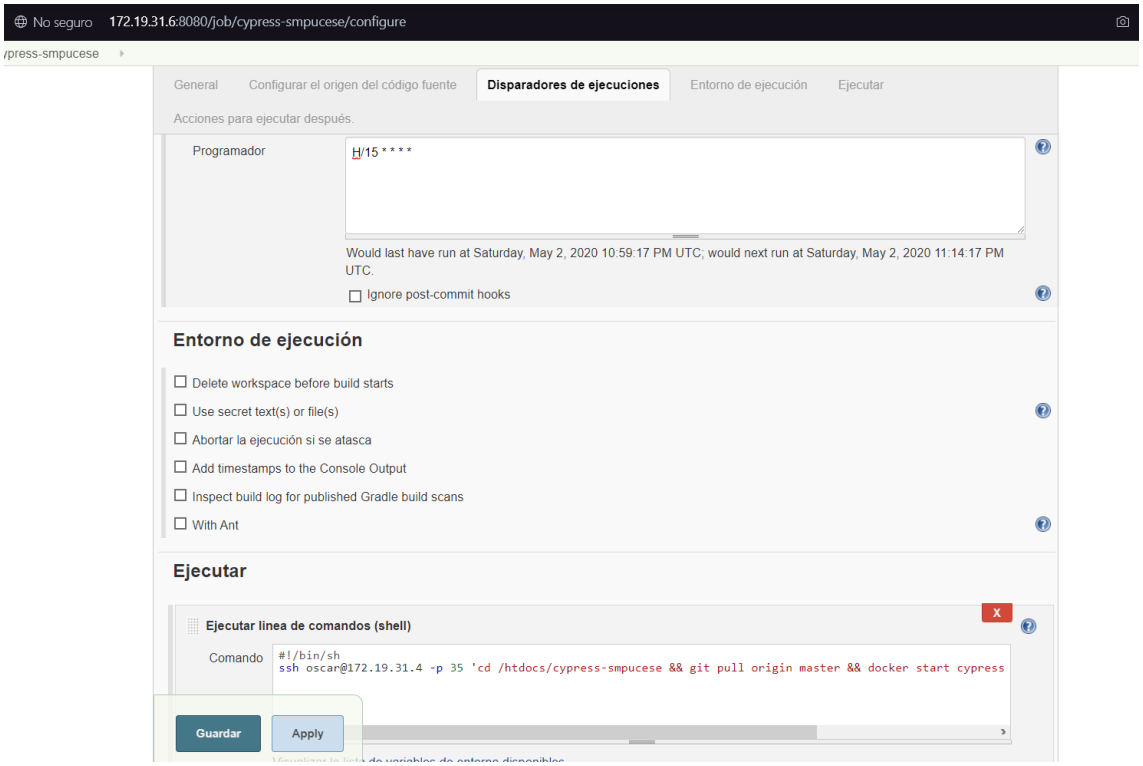

Figura 14. Consulta a repositorio SCM y ejecución de conexión SSH para ejecución de pruebas.

El informe sobre los resultados obtenidos luego de ejecutar las pruebas de extremo a extremo a través de Cypress se quedan reflejadas en la salida de consola de cada ejecución del trabajo de Cypress en Jenkins, como se puede evidenciar en la Figura 15 y Figura 16.

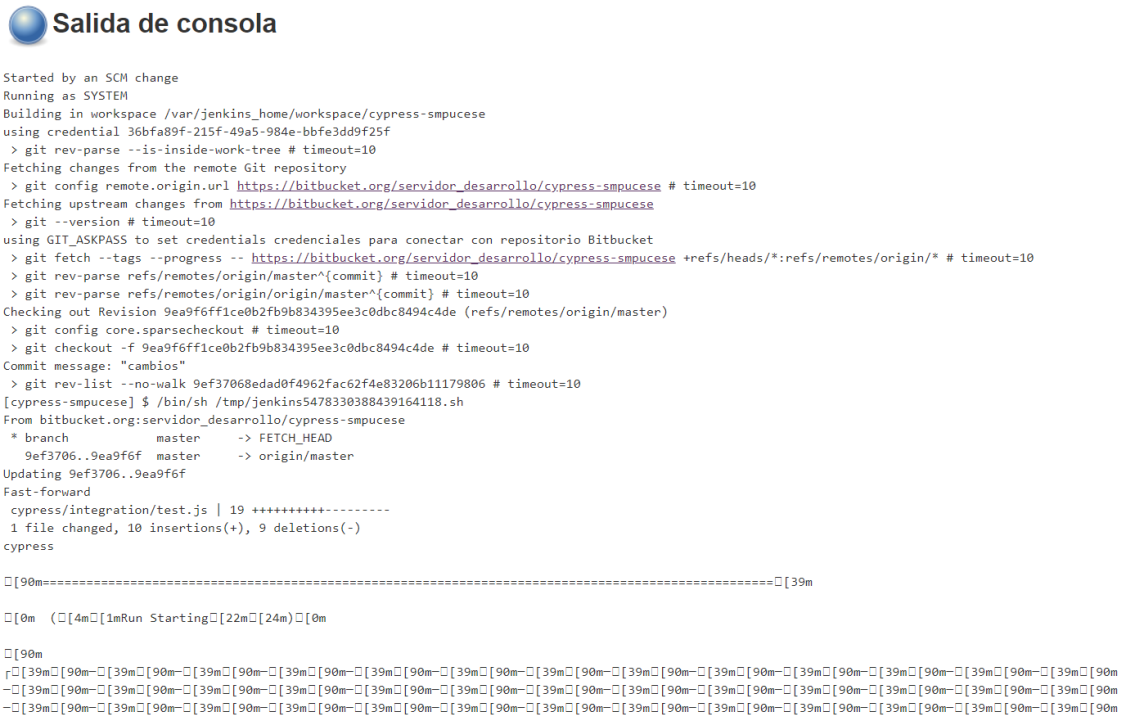

Figura 15. Salida de consola de Cypress en Jenkins 1.

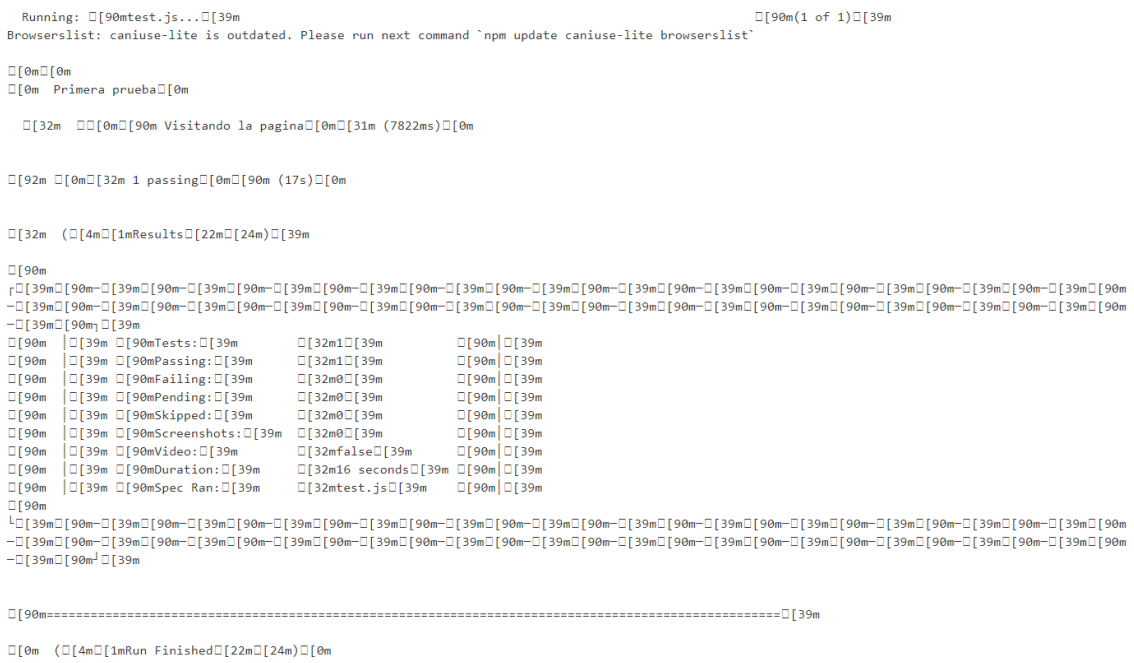

Figura 16. Salida de consola de Cypress en Jenkins 2.

Para medir los resultados del entorno de CI/CD implementado en los servidores del departamento de TICS de la PUCESE y con la finalidad de realizar el desarrollo del sistema médico de esta misma institución, se hizo un conteo de las ejecuciones de los trabajos realizados por el servidor de automatización desde su implementación, hasta la finalización del desarrollo y puesta en producción del sistema médico.

Construcciones de trabajos de Jenkins en entorno de CI/CD			
Construcciones	Servidor de desarrollo	Servidor de producción	Total
Erroneas	18	0	18
Canceladas	7	2	9
Correctas	181	63	244

Tabla 10. Métricas obtenidas a partir de la implementación del entorno de CI/CD

Cabe resaltar que por cada ejecución Jenkins genera un reporte de la compilación de cada trabajo y paso realizado dentro de los mismos, es así que en el servidor de automatización quedan registrados dichos reportes y pueden ser constatados y verificados en los servidores de la PUCESE, estos reportes incluyen los resultados de las pruebas automatizadas de código de tipo End to End y resultados del analizador estático de código fuente que se utilizó para el control de calidad del código fuente.

CAPÍTULO IV: DISCUSIÓN

En el estudio realizado por Gallaba [5] se tiene un enfoque basado en el estudio y mejoramiento de los entornos de CI/CD, esto con el afán de limitar o eliminar los problemas ocasionados por una mala creación o uso del mismo, en esta investigación el enfoque se basa en crear una primera versión de un entorno de CI/CD para la PUCESE dado que es el primero en su tipo en el área de TICS y de este estudio surge un precedente para la creación de entornos similares para otros proyectos. Sin embargo, este enfoque no ha dejado de lado el estudio de las mejores prácticas para el desarrollo del presente entorno, prueba de esto es la evaluación realizada para la selección de las herramientas.

De la investigación [4] se tuvo un enfoque bastante similar, dado que en la misma se da una especial atención al diseño y composición de los entornos de CI/CD y para ello es necesario tomar y analizar los requisitos de la empresa o institución donde se desea realizar este proyecto con este fin, la similitud con esta investigación no termina ahí, sino que los resultados de la misma culminan con un marco de trabajo en donde se recomienda el uso de varias herramientas de cada tipo para la conformación de los entornos, de donde se puede evidenciar que aquellas seleccionadas por ese estudio se encuentran seleccionadas también en este.

En la investigación realizada por L. Chen [20] se evalúan los alcances de la entrega continua de software y sirve de guía para desarrolladores que estén en búsqueda de la implementación de un entorno donde se realice entrega continua de software. Dentro de las afirmaciones de Chen se recomienda el uso de pipelines para la gestión de los trabajos en los servidores de automatización, sin embargo en la presente investigación se implementaron los trabajos en el servidor de automatización como procesos individuales que tenían una relación secuencial entre sí, esto se realizó de esta manera dada la magnitud del proyecto, dado que se trata de un proyecto bastante limitado, desarrollado para un número reducido de usuarios y que se iba a mantener de manera local en la universidad.

En el artículo científico [7] se realiza con el afán de realizar una evaluación de herramientas para el análisis de código estático, a diferencia de esa investigación en donde se realiza con un enfoque práctico, esta investigación tiene un enfoque más bien teórico, sin embargo se pudo apreciar que se llegaron a resultados similares dado que se hallaron grandes diferencias entre unos analizadores de código estático y otros. Esto se debe a que estos analizadores tienen funcionalidades variadas y existen una gran cantidad de tipos de pruebas, además de que la madurez de unas herramientas frente a otras hace que esta diferencia sea mucho más marcada dado que la documentación y la comunidad de cada herramienta se forja a lo largo de los años.

CAPÍTULO V: CONCLUSIONES

Lo expuesto a lo largo de este trabajo de investigación, permite arribar a las siguientes conclusiones:

- Mediante la entrevista realizada al jefe de TICS de la PUCESE fue posible identificar las herramientas preferidas y seleccionadas por este departamento para llevar a cabo el entorno de CI/CD.

- Se encontró una gran similitud entre algunas de las herramientas que se evaluaron en esta investigación y en algunos casos como lo es el de los servidores de automatización fue bastante complicado llegar a una elección debido a la gran competencia entre las herramientas.

- De la revisión de otras investigaciones se llegó a la conclusión de que además de la implementación del entorno de CI/CD también es necesario tomar en cuenta otros aspectos para que el entorno funcione como el uso y mantenimiento del mismo, así como la seguridad del entorno, el hecho de que haya un entorno de pruebas y de producción implica que no siempre se deben comportar igual dado que en el entorno de desarrollo se deben enfocar las configuraciones para hacer las pruebas y en el entorno de producción las configuraciones se deben enfocar en la seguridad.

- A través de los resultados se pudo automatizar el proceso de integración, entrega y despliegue de software en el desarrollo y en la puesta a producción del sistema médico de la PUCESE, en cada una de sus fases, desde la integración con los repositorios de código fuente, pasando por la automatización de las tareas hechas por Jenkins, así como las pruebas y despliegue en los servidores de desarrollo y producción.

- La ISO 25010 permitió evaluar las herramientas de manera satisfactoria en cada una de las características que se consideraron de mayor importancia para la evaluación de este tipo de herramientas y se halló que uno de los aspectos más importantes tiene que ver con la documentación y soporte de cada herramienta, lo cual se encuentra estrechamente relacionado con la madurez de la misma y que sin estos no se podrían aprovechar las funcionalidades de la herramienta aunque esta sea la mejor de todas.

- Las herramientas seleccionadas por el departamento de TICS de la PUCESE tuvieron total compatibilidad con las demás seleccionadas a través de la evaluación con la norma ISO 25010 dado que son herramientas ampliamente utilizadas a día de hoy como Docker o el repositorio de código fuente basado en GIT, Github.

- Se pudo hallar además de los resultados que el uso de la herramienta de Docker, permite ampliamente reusar los entornos de desarrollo y producción para ser utilizados en otros proyectos similares, en donde sólo ciertas partes del mismo es necesario cambiar dadas las especificaciones propias del nuevo proyecto.

CAPITULO VI: RECOMENDACIONES

Se recomienda realizar un mejoramiento continuo de los entornos de desarrollo y producción dado que a diario aparecen mejores prácticas de desarrollo de estos entornos, así como nuevas funcionalidades de cada una de las herramientas que componen un entorno de CI/CD. También se recomienda realizar más evaluaciones de nuevas herramientas que van apareciendo con el pasar del tiempo y herramientas que no se han contemplado en esta investigación para obtener siempre el mejor set up para los entornos.

Se recomienda al departamento de TICS de la PUCESE migrar sus servicios que se mantienen en un servidor de manera local, a servicios basados en la nube, para poder acceder a mejores prácticas de desarrollo y producción, como el uso de herramientas que permitan automatizar el proceso de creación de instancias de servidor en la nube, con la finalidad de poder balancear la carga de los servicios de manera automática con herramientas como Terraform, pudiendo ofrecer a los usuarios de estos servicios la mejor experiencia, eficiencia y rapidez, así como también se ha comprobado que los servicios basados en servidores locales a largo plazo son más costosos que los servicios alojados en la nube, además de ser más seguros y tener menos latencia.

ANEXOS

Entrevista semi estructurada

1. ¿Cómo selecciona las tecnologías a utilizar en el departamento de TICS de la PUCESE?

   Principalmente tienen que estar compatibles con la infraestructura existente. Para elegir nuevas tecnologías a implementar se guía en las mejores prácticas y generalmente una evaluación previa de varios “candidatos”

2. ¿Se utiliza más de una tecnología del mismo tipo en el departamento de TICS de la PUCESE?

   Se busca limitar la cantidad de diferentes tecnologías. Sin embargo, el constante cambio tecnológico exige que se necesita constantemente evaluar nuevas tecnologías

3. ¿Qué tecnología utiliza el departamento de TICS como repositorio de código fuente?

   Por el momento estamos trabajando a bitbucket

4. ¿Qué tecnología de contenedores de software utiliza el departamento de TICS?

   Docker

5. ¿Qué tecnología para análisis estático de código se utiliza en el departamento de TICS?

   En producción no tenemos todavía un analizador de código, pero hemos evaluado sonarqube como candidato potencial

6. ¿Qué tipo de servidores se utiliza en el departamento de TICS de la PUCESE?

   Tenemos servidores propios que estamos manejando con Proxmox y contamos con 2 servidores en la nube.

Bibliografía

[1] “Comprender los conceptos básicos de CI / CD - Por arnés.” [Online]. Available: https://hackernoon.com/understanding-the-basic-concepts-of-cicd-fw4k32s1. [Accessed: 17-Mar-2020].

[2] “Integración de CI / CD con Docker Enterprise Edition - Resumen del seminario web de demostración - Docker Blog.” [Online]. Available: https://www.docker.com/blog/ci-cd-with-docker-ee/. [Accessed: 17-Mar-2020].

[3] D. Marijan, M. Liaaen, and S. Sen, “DevOps Improvements for Reduced Cycle Times with Integrated Test Optimizations for Continuous Integration,” *Proc. - Int. Comput. Softw. Appl. Conf.*, vol. 1, pp. 22–27, 2018.

[4] M. Soni, “End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery,” *Proc. - 2015 IEEE Int. Conf. Cloud Comput. Emerg. Mark. CCEM 2015*, pp. 85–89, 2016.

[5] K. Gallaba, “Improving the Robustness and Efficiency of Continuous Integration and Deployment,” *Proc. - 2019 IEEE Int. Conf. Softw. Maint. Evol. ICSME 2019*, pp. 619–623, 2019.

[6] L. Chen, “Continuous Delivery: Huge Benefits, but Challenges Too,” *IEEE Softw.*, vol. 32, no. 2, pp. 50–54, Mar. 2015.

[7] A. Delaitre, B. Stivalet, E. Fong, and V. Okun, “Evaluating bug finders - Test and measurement of static code analyzers,” *Proc. - 1st Int. Work. Complex Faults Fail. Large Softw. Syst. COUFLESS 2015*, pp. 14–20, 2015.

[8] L. Chen and P. Power, “Continuous Delivery Huge Benefits, but Challenges Too.”

[9] J. Serrano, “Integración, entrega y despliegue continuo. Diferencias y similitudes.” [Online]. Available: https://geeks.ms/jorge/2019/02/25/integracion-entrega-y-despliegue-continuo-diferencias-y-similitudes/. [Accessed: 04-Dec-2019].

[10] M. Fowler, “Integración continua.” [Online]. Available: https://www.martinfowler.com/articles/continuousIntegration.html. [Accessed: 02-Dec-2019].

[11] “Pipeline.” [Online]. Available: https://jenkins.io/doc/book/pipeline/. [Accessed: 16-Feb-2020].

[12] “What is a Container? | App Containerization | Docker.” [Online]. Available:

https://www.docker.com/resources/what-container. [Accessed: 16-Dec-2019].

[13] “Microservices Delivery & Management with Containers | Docker.” [Online]. Available: https://www.docker.com/solutions/microservices. [Accessed: 24-Dec-2019].

[14] “Podman | podman.io.” [Online]. Available: https://podman.io/. [Accessed: 16-Feb-2020].

[15] “rkt, un motor contenedor basado en estándares y con mentalidad de seguridad.” [Online]. Available: https://coreos.com/rkt/. [Accessed: 24-Dec-2019].

[16] “Contenedores Linux - LXC - Introducción.” [Online]. Available: https://linuxcontainers.org/lxc/introduction/. [Accessed: 24-Dec-2019].

[17] “Características · Las herramientas adecuadas para el trabajo · GitHub.” [Online]. Available: https://github.com/features. [Accessed: 24-Dec-2019].

[18] “Bitbucket | La solución Git para equipos profesionales.” [Online]. Available: https://bitbucket.org/product/. [Accessed: 24-Dec-2019].

[19] “Producto | GitLab.” [Online]. Available: https://about.gitlab.com/product/source-code-management/. [Accessed: 24-Dec-2019].

[20] L. Chen, “Microservices: Architecting for Continuous Delivery and DevOps,” *Proc. - 2018 IEEE 15th Int. Conf. Softw. Archit. ICSA 2018*, pp. 39–46, 2018.

[21] “Servidores de automatización.” [Online]. Available: https://technet.microsoft.com/es-es/6wx53dax(v=vs.85). [Accessed: 16-Dec-2019].

[22] “Jenkins.” [Online]. Available: https://jenkins.io/. [Accessed: 24-Dec-2019].

[23] “Azure DevOps Server | Microsoft Azure.” [Online]. Available: https://azure.microsoft.com/es-es/services/devops/server/. [Accessed: 24-Dec-2019].

[24] “The first single application for the entire DevOps lifecycle - GitLab | GitLab.” [Online]. Available: https://about.gitlab.com/. [Accessed: 24-Dec-2019].

[25] “Bamboo: servidor de integración continua y compilación.” [Online]. Available: https://www.atlassian.com/es/software/bamboo. [Accessed: 24-Dec-2019].

[26] “Código de Calidad y Seguridad | SonarQube.” [Online]. Available: https://www.sonarqube.org/. [Accessed: 16-Dec-2019].

[27] “PMD.” [Online]. Available: https://pmd.github.io/. [Accessed: 24-Dec-2019].

[28] “Análisis estático (SAST) | Veracode.” [Online]. Available:

https://www.veracode.com/products/binary-static-analysis-sast. [Accessed: 24-Dec-2019].

[29] “Bienvenido | Cypress Developer Community.” [Online]. Available: https://community.cypress.com/welcome. [Accessed: 19-Apr-2020].

[30] “OpenTest: herramienta de automatización de pruebas para web, dispositivos móviles y API.” [Online]. Available: https://getopentest.org/. [Accessed: 24-Dec-2019].

[31] “Marco de robot.” [Online]. Available: https://robotframework.org/. [Accessed: 24-Dec-2019].

[32] “ISO 25010.” [Online]. Available: https://iso25000.com/index.php/normas-iso-25000/iso-25010. [Accessed: 15-Jan-2020].

[33] D. E. Ramos, “Diseño De Un Modelo De Evaluación De La Calidad De Productos De Software, Basado En Métricas Externas Y Usabilidad Aplicado a Un Caso De Estudio,” p. 115, 2016.

[34] “Documentación de usuario de Jenkins.” [Online]. Available: https://jenkins.io/doc/. [Accessed: 12-Apr-2020].

[35] “Complementos de Jenkins.” [Online]. Available: https://plugins.jenkins.io/. [Accessed: 12-Apr-2020].

[36] “Participa y contribuye.” [Online]. Available: https://jenkins.io/participate/. [Accessed: 12-Apr-2020].

[37] “Documentos de GitLab | GitLab.” [Online]. Available: https://docs.gitlab.com/ee/README.html. [Accessed: 18-Apr-2020].

[38] “SSH | GitLab.” [Online]. Available: https://docs.gitlab.com/runner/executors/ssh.html. [Accessed: 18-Apr-2020].

[39] “Integraciones de GitLab | GitLab.” [Online]. Available: https://docs.gitlab.com/ee/integration/. [Accessed: 18-Apr-2020].

[40] “Tuberías de CI / CD | GitLab.” [Online]. Available: https://docs.gitlab.com/ee/ci/pipelines/. [Accessed: 18-Apr-2020].

[41] “Documentos de usuario | GitLab.” [Online]. Available: https://docs.gitlab.com/ee/user/. [Accessed: 18-Apr-2020].

[42] “Vinculación a repositorios de código fuente - Documentación de Atlassian.” [Online]. Available: https://confluence.atlassian.com/bamboo/linking-to-source-code-repositories-671089223.html. [Accessed: 19-Apr-2020].

[43] “Uso de la tarea SSH en Bamboo - Atlassian Documentation.” [Online].

Available: https://confluence.atlassian.com/bamboo/using-the-ssh-task-in-bamboo-306348532.html. [Accessed: 19-Apr-2020].

[44] “Bamboo apps | Atlassian Marketplace.” [Online]. Available: https://marketplace.atlassian.com/search?product=bamboo. [Accessed: 19-Apr-2020].

[45] “Documentación de bambú - Documentación de Atlassian.” [Online]. Available: https://confluence.atlassian.com/bamboo. [Accessed: 19-Apr-2020].

[46] “Soporte de bambú | Bamboo - más reciente | Atlassian Support.” [Online]. Available: https://support.atlassian.com/bamboo/. [Accessed: 19-Apr-2020].

[47] “¿Por qué ciprés? El | Documentación de Cypress.” [Online]. Available: https://docs.cypress.io/guides/overview/why-cypress.html#Who-uses-Cypress. [Accessed: 24-Dec-2019].

[48] “Preguntas generales | Documentación de Cypress.” [Online]. Available: https://docs.cypress.io/faq/questions/general-questions-faq.html#Do-you-support-X-language-or-X-framework. [Accessed: 19-Apr-2020].

[49] “Support. | cypress.io.” [Online]. Available: https://www.cypress.io/support/. [Accessed: 19-Apr-2020].

[50] “GitHub - robotframework / robotframework: marco de automatización genérico para pruebas de aceptación y RPA.” [Online]. Available: https://github.com/robotframework/robotframework. [Accessed: 19-Apr-2020].