Pontificia Universidad Católica del Ecuador
Seréis mis testigos

ESMERALDAS

Ingeniería en Tecnologías de la Información

Título del proyecto: Propuesta de implementación de un proceso de despliegue automatizado, previo pruebas y análisis de código automatizado

Previo al grado académico de Ingeniero en Tecnología de la Información

Línea de investigación: Software y Tecnología

Autor: Kevin José Gracia Orejuela

Asesor: Mgt. Marc Grob

Mayo, 2022

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El desarrollo de software en la actualidad exige velocidad para adaptarse al ritmo acelerado de consumo de los usuarios, la necesidad de crear productos rápidamente está latente en el día a día, además se busca maximizar la productividad de los desarrolladores lo que lleva a emplear estrategias de automatización, por tanto la presente investigación tuvo como objetivo adoptar este enfoque mediante la elaboración de un entorno de desarrollo, entrega y despliegue continuo de modo que también se puedan obtener los beneficios de la implementación de estas prácticas en los procesos de desarrollo de software en el Departamento de TI de la PUCESE.

La investigación realizada constó de una revisión bibliográfica de la literatura previa donde se logró identificar la forma en que otras empresas están implementando canales de integración, entrega y despliegue continua, así como las herramientas que están siendo utilizadas también se tomaron los requerimientos del contexto del Departamento de TI de la PUCESE la cual tiene un equipo de desarrollo, este insumo fue utilizado como base para la elaboración de la propuesta una vez que se aplicaron los criterios de la norma ISO 25010 que tuvo su posterior implementación al establecer este entorno de desarrollo, entrega y despliegue continuo previo a pruebas y análisis de código automatizado.

Los criterios aplicados de la norma ISO mediante los cuales se evaluaron las distintas formas de implementación de canales de integración y despliegue continuo fueron Completitud Funcional, Capacidad de Aprendizaje, Capacidad de ser usado, Reusabilidad, Modularidad mediante estos se pudo distinguir cual forma de elaboración se adaptaba más la investigación realizada.

Los resultados determinaron el tipo de pipeline acompañado de las herramientas que según el caso se adaptan de forma adecuada al proceso automatizado de pruebas y análisis de código estático lo que de acuerdo con lo investigado puede brindar mejoras en el aseguramiento de calidad en proyectos de software de equipos pequeños, así como la implementación de despliegues automáticos.

1. INTRODUCCIÓN

1.1. DESCRIPCIÓN DEL PROBLEMA

Las metodologías de desarrollo de software tradicionales pueden presentar grandes retos como: la inconsistencia entre los ambientes de desarrollo y producción, limitaciones de tiempo para entregas frecuentes, dificultades para administrar múltiples configuraciones y versiones de aplicaciones en los servidores, mayor riesgo de errores debido al aumento de intervenciones manuales lo que se traduce en costos más altos y periodos de tiempo más largos [1].

Por tal motivo la carrera de la automatización en el ciclo de desarrollo de software ha llegado a casi todas las fases de este proceso desde su construcción, pruebas de funcionamiento e incluso a las más críticas como la puesta en producción o despliegue. Escribir programas que sean capaces de resolver ciertas problemáticas solo con código hoy en día no es suficiente, siempre hay que ir más allá de solo escribir código [2]

Al mismo tiempo la oferta de servicios y herramientas para realizar tareas automatizadas ha ido en aumento partiendo desde Jenkins [3] instalado en servidores on-premise, hacia la gestión de automatización como servicio on-demand en los mismos repositorios donde se almacena el código fuente es así como empresas de este dominio como Github, Gitlab, Atalassian, y proveedores de servicios en la nube como Amazon Web Services, Azure y muchas otras compañías también brindan soluciones de este tipo.

No obstante, en un entorno de TI en rápida evolución, muchas organizaciones y desarrolladores buscan probar y lanzar, productos y servicios digitales, más rápido y a menor costo apoyándose en herramientas y procesos que adapten a su modo de trabajo, limitación de recursos, giro de negocio, y demás variables posibles es así que no existe un estándar para realizar estas tareas y elegir un conjunto de tecnologías para este propósito se ha vuelto una labor complicada [2]

En la actualidad a pesar de que existen diversos modelos de procesos en desarrollo del software disponibles, de lo que se ha hecho uso durante años, se sigue sin resolver por

completo el problema de las liberaciones tardías. Mientras tanto el software se vuelve constantemente más grande, complejo y requiere de una alta calidad [4].

Por otro lado, los desarrolladores se enfrentan diariamente en la tarea de implementar nuevas funcionalidades en el software agregando código, pero no solo a eso si no también deben tratar con los errores que se presentan en el camino y luego, probar ellos mismos esta funcionalidad. Sin embargo, no importa si los desarrolladores validan sus cambios por sí mismos o confían en los comentarios de otros a través de revisiones de código, ambos casos requieren humanos que realicen la inspección.

Este componente humano introduce un considerable esfuerzo manual y también es muy propenso a errores, porque la identificación de fallas en un conjunto de cambios en el código requiere vasta experiencia en programación [5].

En consecuencia, surge DevOps que es una práctica emergente que se ha adoptado en el ciclo de desarrollo de software. Se centra en la convergencia de estándares entre los equipos de desarrollo y el equipo de operaciones y busca mejorar la cooperación entre ambos equipos, de ahí el origen del término [6].

Sin embargo, no hay consenso sobre la definición de lo que es DevOps. Wiedemann et al. [7] enfatiza que uno de los mayores desafíos en la industria es la falta de un concepto formal para DevOps. A falta de este concepto se tiende a confundir las acciones que se quieren lograr con esta práctica.

1.2. JUSTIFICACIÓN

La presente investigación se basa en las necesidades actuales en el ciclo de desarrollo de software, lo que demandan las empresas a nivel mundial para satisfacer los requerimientos de sus clientes en el menor tiempo posible, rediseñar sus procesos y adoptar nuevas metodologías que mejoren la calidad de los sistemas y aceleren las entregas de este, como la integración, prueba, mejora y despliegue continuo.

De esta forma se planteará la propuesta de implementación de un proceso DevOps completo en un nuevo proyecto de software usando herramientas populares de código abierto o propietario para evidenciar los beneficios que tienen sobre el desarrollo de software.

A través de este proyecto se brindará información valiosa a los desarrolladores de software tanto de empresas como independientes para tener como referencia que herramientas usar en un proceso de pruebas, despliegue y análisis de código automatizado. Por tanto, esta investigación tendrá repercusiones positivas debido a que los desarrolladores podrán tomar una decisión fundamentada y ser más ágiles, así pues, se denota una influencia directa de este proyecto al ecosistema del desarrollo de software.

Al plantear y ejecutar este estudio en el departamento de TI de la PUCESE permitirá un cambio de paradigma en los procesos de programación de los sistemas en cuestión que aquí se realizan y puede resultar muy beneficioso no solo en fases concretas de desarrollo sino también en implementaciones y despliegues de estos, así como su mantenibilidad y escalabilidad.

1.3. OBJETIVOS

1.3.1. Objetivo General

Elaborar una propuesta de implementación un proceso de pruebas, análisis y despliegue automatizado de software mediante el uso de herramientas de integración, implementación y distribución continua CI/CD con el fin de mejorar la calidad del código y agilizar la entrega de funcionalidades en el departamento de TI de la PUCESE.

1.3.2. Objetivos Específicos

- Identificar distintas formas de elaborar canales de integración y despliegue continuo CI/CD para el aseguramiento de la calidad del código.
- Comparar distintas opciones de canales de integración y despliegue continuo.
- Conocer los procesos existentes de desarrollo de software en el departamento de TI de la PUCESE.
- Realizar la propuesta de implementación de un canal CI/CD para el ciclo de desarrollo de software en el departamento de TI de la PUCESE.

1.4. MARCO TEÓRICO

1.4.1. Agile

Agile, es un término usado para describir un enfoque del desarrollo de software que consiste en un método iterativo e incremental, el cual en cada ciclo de iteración se ejecutan una serie de acciones como colaboración, aprendizaje y planificación continua para hacer unas entregas parciales del producto [8].

De forma que el Manifiesto Agile [9] propone 4 principios esenciales para mejorar el enfoque del desarrollo de software:

- Individuos e interacciones sobre procesos y herramientas.
- Software funcionando sobre documentación extensiva.
- Colaboración con el cliente sobre negociación contractual.
- Respuesta ante el cambio sobre seguir un plan.

Así el desarrollo de software ágil propone mejoras orientadas a la rapidez y construcción de productos de mejor calidad.

1.4.2. DevOps

DevOps, este término consiste en una unión de dos palabras más, desarrollo y operaciones, es un enfoque en el que las personas que trabajan en el área de desarrollo y las personas del área de operaciones cooperan estrechamente[10] , siendo el objetivo principal mejorar y agilizar el proceso de desarrollo de software y empleando prácticas orientadas a entregas de software en corto tiempo y ciclos de retroalimentación constantes.

Microsoft [11] se refiere a DevOps como una práctica que permite coordinar y colaborar entre roles que antes se encontraban separados como desarrollo, operaciones de TI, ingeniería de calidad y seguridad para crear productos mejores y más confiables [11] .

Finalmente, en Amazon Web Services [12] se afirma que, bajo un modelo de DevOps, los equipos de desarrollo y operaciones ya no están “aislados”. A veces, los dos equipos se fusionan en uno solo, donde los ingenieros trabajan en todo el ciclo de vida de la aplicación, desde el desarrollo y las pruebas hasta la implementación y las operaciones, y desarrollan una variedad de habilidades no limitadas a una única función.

1.4.3. Relación DevOps y Agile

Estas dos metodologías de desarrollo de software no se encuentran en conflicto la una con la otra, pero si tienen enfoques y actúan en procesos diferentes. También ambas puedes coexistir y ofrecen más eficiencia a los equipos, cuando se utilizan juntas. Por un lado, la cultura DevOps se centra en maximizar la eficiencia aumentando la automatización de proceso y por otro la filosófica del método ágil consiste en dar respuesta rápida a los errores que surgen en el proceso de desarrollo y adaptabilidad constante al ritmo cambiante de las necesidades del software [13].

1.4.4. SRE

Ingeniería de Confiabilidad del Sitio (Site Reliability Engineering) es un término acuñado por Treynor Sloss de Google [14] de donde se considera que es un rol de trabajo que consiste en implementar un conjunto de prácticas asociadas a la filosofía que describe DevOps, y está considerado como una definición concreta de un puesto de trabajo donde se trata los procesos de operaciones como un problema de software, por lo tanto la misión fundamental de esta posición es mantener los sitios y sistemas en correcto funcionamiento, evitar los fallos y que se encuentren disponibles la mayor cantidad de tiempo posible.

1.4.5. Versionamiento de Código

El versionamiento de código se realizar con la ayuda de herramientas que permiten a las personas o desarrolladores trabajar de forma simultánea en un proyecto o en una misma base de código, así cada desarrollador edita su propia copia de los archivos y luego los comparte con el resto del equipo [15] , de esta forma se mantiene un historial de cambio ya que se está registrando la informacion de cuando se realizaron y quien los realizó.

1.4.6. Repositorio

Un repositorio es donde se encuentra contenido todo el código fuente de un proyecto es decir sus distantes carpetas y archivos, pero además está el historial de cambios en cada uno de estos archivos que se pudieron realizar a lo largo del tiempo [16]. Generalmente se encuentra en un servicio en línea como GitHub [15] u otras alternativas existentes, así cada desarrollador se puede hacer una copia local, realizar los cambios y luego confirmarlos al repositorio principal.

1.4.6.1. Git

Git es la herramienta que permite realizar el control de versiones, tiene diversos comandos y funcionalidades de las cuales los desarrolladores hacen uso para manejar y compartir el código relacionado con los cambios que se realizan [17].

1.4.6.2. Principales comandos de Git [17]

- **Init**: Inicia un repositorio de git dentro de un directorio ya existente, es decir añade todo lo necesario para hacer uso del sistema de control de versiones
- **Clone:** Descarga todos los archivos de un repositorio creando una copia local
- **Add:** Rastrea los cambios que se han realizado en los archivos o los prepara
- **Commit:** Previamente ejecutado el comando git add, con commit se confirman los cambios y se registra el cambio en el historial de proyecto
- **Branch:** Consiste en una bifurcación del proyecto en otra línea de cambios, existe la rama principal llamada master o main y demás ramas que se pueden crear de acuerdo a la necesidad
- **Merge:** Permite la combinación de varias ramas de desarrollo, con el fin de unir los distintos cambios generados en cada una de ellas
- **Pull:** Actualiza la rama de desarrollo local en referencia a la misma rama que se encuentra de forma remota en el servicio que alberga el repositorio
- **Push:** Envía al repositorio remoto los cambios de una rama que se realizaron de forma local

1.4.6.3. Estrategias de Ramificación

Git debido a su flexibilidad ofrece distintas formas de organizar el flujo de trabajo en los equipos de desarrollo para gestionar los diversos, estos flujos de trabajo, no existe un proceso estándar para interactuar con git pues se realiza de acuerdo a las necesidades del equipo producto de software, estas estrategias y flujos de trabajos se basan en la forma en que se usan las ramas de git, creándolas en función de un propósito definido acuerdo con el flujo de trabajo establecido , la aplicación de estas estrategias dependerán de varios factores como el tamaño del equipo, la frecuencia con la que se hacen los despliegues a producción, manejo de conflictos y otros más, en continuación se mencionarán las principales:

- Flujo de trabajo centralizado: existe una única rama en el repositorio desde y hacia donde todos los desarrolladores confirman sus cambios[18].
- Ramas de funcionalidades: consiste en que cada funcionalidad agregada al código se gestionará y confirmarán sus cambios en una nueva rama y luego se unirá a la rama principal (master/main) [18].
- GitFlow: Esta estrategia crea diversas ramas ajustadas a un propósito y con una conveción de nombres predefinidas, cuenta con dos ramas principales "master" la rama principal y "develop" donde se colocan los cambios que ya se encuentran listos para producción, también ramas de apoyo como ramas de funciones "features", de liberación "realeases" y de arreglos "hotfixs" [19].

1.4.7. Aseguramiento y Control de Calidad de Software (SQA/SQC)

Las aplicaciones de software juegan un rol importante en las Tecnologías de la Información y para que un sistema sea confiable y eficaz, la calidad del software es muy importante [20], existen diversas formas de evaluar la calidad de un software, pero debido al carácter de la investigación se abordan aquellas relacionadas al funcionamiento de su código fuente.

Las pruebas son una parte importante en el área de Aseguramiento de Calidad de Software (SQC) y es una actividad en la que su objetivo básico es detectar y resolver problemas técnicos en el código fuente del software [20].

Las pruebas de software son consideradas como una colección de técnicas, procedimientos y herramientas que esta relacionadas al software, es un proceso mediante el que se asegura que todas las actividades realizadas por el software son las que esperan obtener [20]. Probar software abarca muchos aspectos y consideraciones en esta investigación se busca un uso de lo estrictamente necesario en cuanto a pruebas para satisfacer las necesidades de la propuesta a realizar.

Pruebas Automatizadas

Es el uso de herramientas para construir pruebas que se ejecuten automáticamente y así asegurar la calidad en el desarrollo del software [21]. El uso general de estas herramientas consiste en escribir las pruebas con código y luego ejecutarlas con

distintos comandos que se pueden que se pueden acoplar en ciclos de integración continua.

1.4.7.1. Pirámide de Automatización de Pruebas

Este modelo fue propuesto por Cohn [8] y tiene como base a las pruebas unitarias dándoles mayor prioridad, luego las pruebas de servicios, y finalmente en la punta de la pirámide pruebas de Interfaz de Usuario según lo expuesto en la bliki de Fowler en [22] se pueden definir de la siguiente forma:

a) **Pruebas Unitarias:** Estas pruebas se aseguran de que una unidad en el código funciona como se esperaban, entendiendo como unidad una función o fragmento de código que no tiene dependencia con otro.
b) **Pruebas de Integración:** En estas pruebas se constata que la aplicación en cuestión mantenga la conexión correcta con otros servicios que pueden se externos como bases de datos, sistemas de archivos u otras aplicaciones**.**
c) **Pruebas de Interfaz de Usuario:** Se aseguran de que lo que se muestra en pantalla funciona como se espera, es decir al presionar botones o realizar distintas acciones, que pueden llegar a ser web.

1.4.7.2. Análisis estático de código

Es la tarea automatizada que realizan ciertas herramientas sobre el código fuente con el fin de detectar y comprobar fallos en el código como malas prácticas, errores, vulnerabilidades, no seguir estilos de código y enviar alertas para su corrección, estas herramientas verifican el código sin ejecutarlo [23].

1.4.8. Pipelines CI/CD

Una pipeline CI/CD es una manifestación automatizada del proceso que lleva el software desde el control de versiones hasta las manos de los usuarios. Cada cambio en el software pasará por este circuito que constituye un proceso complejo en su camino a la puesta en producción [24]. Previo a esta implementación una organización debió adoptar previamente todas las prácticas de Integración Continua hacia Entrega Continua y el posterior Despliegue Continuo, automatizando todos

los procesos [25]. De esta forma todas las etapas desde la construcción del software hasta la entrega se realizan en esta pipeline.

Como lo señala Hall [26] en no existe una forma estándar de implementar una pipeline, estas se diseñan y se implementan de acuerdo al software, dependiendo de muchos factores como es el conjunto de tecnologías con las que se construye, experiencia de los ingenieros, presupuesto y otros más, pero las etapas siempre persiguen lo detallados en los procesos CI/CD.

1.4.9. Integración Continua (CI)

La integración continua una práctica del desarrollo de software, en la que los miembros de un equipo incorporan y fusionan el código elaborado de forma constante con pasos previos de construcción y pruebas, puede ser varias veces en un mismo día [27], CI permite a las empresas de software tener ciclos de lanzamiento más cortos y frecuentes, mejorar la calidad del software y aumentar la productividad de sus equipos, esto se logra automatizando las tareas de integración como por ejemplo pruebas automatizadas necesarias para validar el correcto funcionamiento del código.

1.4.9.1. Etapas

Las fases consideradas en este punto deben ser automatizadas y dependen de las necesidades del software, se pueden dividir en:

1. Construcción (Build): El código fuente desarrollado debe convertirse en un sistema en ejecución y a menudo esto proceso implica compilación, edición de archivos, carga de datos, escribir comandos o lidiar con elección de opciones de configuración lo que resulta en pérdida de tiempo [28]. De este modo para agilizar este proceso se deben crear los scripts necesarios usando las herramientas adecuadas para que se realice de forma automática.
2. Pruebas (Test): Una vez realiza la construcción se tendrá un ejecutable, pero en referencia a lo argumentado por Fowler *"Un programa puede ejecutarse, pero eso no significa que haga lo correcto"* [29]. Por lo tanto, se deberán

escribir pruebas para comprobar si el funcionamiento es el esperado y poder avanzar a la siguiente etapa.

Fusión (Merge): consiste en un combinar los cambios de código de una rama que tienen la nueva funcionalidad hacia una rama base [30], esta fusión se realiza mediante una solicitud.

1.4.10. Entrega Continua (CD)

Entrega Continua o Continuous Delivery es la habilidad para desarrollar nuevas funcionalidades o cambios en el software que se pueden lanzar a producción de manera rápida y segura en cualquier momento [25]. Acorde con un estudio realizado a varias compañías que adoptaron entrega continua reportaron beneficios significativos como mejoras en tiempos de productividad, calidad de producto, eficiencia y la habilidad para hacer pruebas experimentales rápidamente [31].

Cabe destacar que la entrega continua no indica salidas automáticas a producción de hecho se lo hace de manera controlada en ocasiones de forma manual previo a una aprobación con lo que finalmente se entrega hacia ambientes productivos o de pruebas.

1.4.11. Despliegue Continuo (CD)

El despliegue continuo es un proceso que permite que los cambios en el código de una aplicación lleguen directa y automáticamente a entornos de producción [23] una vez superadas las pruebas que han sido definidas en los pasos previos, de esta manera los cambios llegan a los usuarios finales de forma periódica.

Tomando en cuenta [23] diversas empresas han logrado escalar el número de desarrolladores y el tamaño de la base de código, además que según este estudio los desarrolladores prefieren ciclos de liberación cortos que largos.

1.4.12. Estrategias de Despliegue

Una estrategia de despliegue o implementación es una forma de actualizar o de cambiar una aplicación [32]. Según la guía de Google [33] para garantizar un lanzamiento sin interrupción se debe tener en cuenta que hay que minimizar el

tiempo de inactividad en caso de ocurrir, resolver incidencias y poder regresar a estados anteriores.

- **Despliegue Básico:** Absolutamente todas las instancias de una aplicación se actualizan a una nueva versión, a causa de esto si ocurre un error es más difícil regresar al estado anterior, provocando posibles interrupciones [34].
- **Despliegue Progresivo:** Se dividen las instancias de la aplicación en subconjuntos y los nodos van recibiendo la actualización de forma incremental [34].
- **Despliegue Azul-verde:** Las instancias actuales son consideradas el entorno verde y las nuevas versiones de la aplicación el entorno azul, de modo que de acuerdo con el comportamiento de la aplicación el entorno verde se convierte en el entorno azul [24]. Se prevé que estos cambios de entornos se puedan realizar en unos pocos segundos
- **Despliegue Canary:** Teniendo un solo entorno de producción extenso se libera la nueva versión de la aplicación solo para una pequeña cantidad de usuarios por ejemplo el 2% [34].

En conclusión, el tipo de despliegue dependerá muchos aspectos como el tipo de infraestructura con la cual que cuenta la organización y las herramientas con las que la administra, así como el tipo de aplicación, número de usuario y otros factores, existen más tipos de despliegue, pero se ha mencionado los principales.

1.4.13. Herramientas de CI/CD

Para poder establecer una pipeline de CI/CD es necesario usar herramientas que tengan todas las capacidades para soportar los también llamadas canales de integración, las cuales son integración con un repositorio de código fuente, diferentes herramientas de automatización como de construcción, pruebas, análisis de código estático y despliegue [25], este tipo de herramientas pueden ser usadas en servidores on-premise llamándose también servidores de Integración y Despliegue

continuo o usar servicios en línea como CI/CD as a Services, las distintas opciones de este tipo de herramientas y las más conocidas a la fecha de realizar esta investigación son Jenkins, Gitlab CI/CD, Azure Pipelines, Bitbucket Pipelines, Bamboo, TeamCity todas estas tiene características similares difiriendo en ciertos detalles siendo unas open source y otras de pago.

1.5. ANTECEDENTES

La investigación del presente estudio se llevó a cabo mediante una revisión bibliográfica de artículos científicos en diversas fuentes bibliográficas tales como IEEE EXPLORE, ACM, WEB OF SCIENCE, SCOPUS relacionados directamente con el objeto de estudio "Implementación de un proceso de despliegue automatizado, previo pruebas y análisis de código automatizado" usando la cadena búsqueda (continouos delivery OR continuous continuous integration OR continuous deployment OR devops OR software pipelines) para recabar la información necesaria. Los presentes recursos considerados como insumos están comprendidos en un intervalo entre los años 2017 a 2021, y son los siguientes:

La primera investigación seleccionada titula "*Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices*" la cual tuvo como objetivo principal proporcionar un conjunto de conocimientos probatorios sobre estado del arte de las prácticas de continuas, es decir la integración, entrega e implementación continuas y las áreas potenciales de investigación. La metodología utilizada Revisión de Literatura Sistemática (SLR) proporciona una comprensión profunda de los desafíos de adoptar prácticas continuas, las estrategias y herramientas utilizadas para abordar estos retos. Entre los resultados aportados: una lista de factores críticos que deben considerarse cuidadosamente al implementar prácticas continuas en desarrollo de software, una guía basada en evidencia para seleccionar enfoques apropiados, herramientas y prácticas basadas en la idoneidad requerida para diferentes contextos [35]. Esta investigación se relaciona con el tema de estudio debido a que la información proporcionada será un punto de partida y de referencia para seleccionar las distintas estrategias y herramientas para la propuesta de implementación que se elaborará.

En el segundo antecedente relata de forma detallada cómo se implementa un entorno colaborativo entre equipos de desarrollo y de operaciones en la práctica y se realizaron estudios de casos en cinco contextos de desarrollo de software diferentes así la investigación titulada *"DevOps in practice: A multiple case study of five companies"* relata implementaciones exitosas de DevOps ya que se lograron sus beneficios, como lanzamientos rápidos y errores mínimos de implementación, dentro de los resultados se halló que el uso de un conjunto de herramientas para un canal de integración y despliegue continuo acelera la entrega de cambios en el software, la corrección de errores y el manejo de incidentes de producción así este estudio concluye que se debe establecer un equilibrio entre la velocidad de las entregas y la calidad del código [10]. Este antecedente guarda relación con la presente investigación debido a que los estudios de casos en las cinco distintas compañías dan conocer que herramientas y tecnologías están siendo usadas en entornos productivos por lo tanto su aporte es de suma importancia para alcanzar los objetivos que serán planteados.

Dentro de la tercera investigación revisada se encuentra la realizada por Zampetti et al. [36] la cual tiene por título *"How Open Source Projects use Static Code Analysis Tools in Continuous Integration Pipelines"* este estudio tuvo como objetivo investigar que herramientas están siendo utilizadas en determinadas proyectos de software y como están configuradas en su canal de integración continua, la metodología aplicada fue el estudio empírico y arroja como resultados que muchas de las fallas tienen que ver con poca implementación de estándares de código, y este concluye que los errores en la construcción debido a alertas del análisis de código estático se resuelven de una forma sencilla [36]. La relación de este estudio se da tanto en el uso de las herramientas de análisis de código automatizado debido a que dentro de la propuesta resultante del presente trabajo investigativo se necesita agregar una herramienta de este tipo, así también como en su implementación en la integración continua de tal modo que se tomará como referencia su contenido para el trabajo de investigación.

El cuarto artículo titulado *"Test Automation Process Improvement in a DevOps Team: Experience Report"* esta investigación tuvo como objetivo desarrollar un informe de la experiencia que tuvo un equipo de desarrollo al implementar una proceso de pruebas

automatizadas, se midieron varios aspectos como la satisfacción y productividad dando como resultado aumentos considerables en la rapidez de lanzamientos de nuevas características y funcionalidades del software, así como mejora en la calidad del código, disminución de errores e incidencias asociadas a comportamientos inusuales de sistema [21].Uno de los procesos dentro de esta investigación abarca el apartado de pruebas de software en este caso en particular pruebas automatizadas motivo por el cual el antecedente descrito tiene relación y aportará al haberse implementado estas pruebas en un equipo de desarrollo tal como se planteará en la propuesta de este proyecto tratando de conseguir los mismos objetivos orientados a la velocidad de desarrollo y calidad.

En el quinto antecedente tomado tiene el título: *"Adopting Continuous Integeration and Continuous Delivery for Small Teams"* este artículo tiene como objetivo identificar los elementos necesarios para la adopción de estrategias CI/CD en los equipos, también trata de entender la relación entre los distintos conceptos y prácticas se requieren para lograr dicha adopción en equipos pequeños y concluye que el problema más importante al implementar integración y despliegue continuo en este tipo de equipos es el consumo de tiempo en las pruebas, por lo que propone que los canales de integración sean muy cortos así alcanzarán mejoras en el desarrollo [37]. Este artículo se relaciona con la investigación precisamente porque se busca hacer la propuesta en un equipo de desarrolladores de similares características y debido a que adoptar estas estrategias también depende de los contextos como el del presente trabajo.

2. METODOLOGÍA

2.1. Delimitación de la investigación

Delimitación espacio temporal

La propuesta de implementación del proceso de despliegue automatizado, previo pruebas y análisis de código automatizado, fue diseñada para el departamento de TI de la PUCESE, ubicada en la ciudad de Esmeraldas, Ecuador. Teniendo en cuenta ello, se desarrolló un pipeline de integración continua que contó con pruebas y análisis de código automatizado. Para este desarrollo se seleccionaron las herramientas idóneas empezando por la plataforma de repositorio de código que se usaba y tenía las funcionalidades necesarias para elaborar integración y despliegue continuo.

Por lo tanto, para abordar el estudio en su etapa de investigación y análisis de información recopilada en los artículos estudiados en el segundo trimestre del 2021. Luego se ejecutará la fase de desarrollo, donde se seleccionaron las herramientas y procesos que finalmente estarán plasmados en la elaboración de la propuesta estableciendo como límite el mes de diciembre de 2021, así se tuvo previsto que el proyecto entregue resultados tangibles luego de cinco meses de trabajo.

2.2. Tipos de investigación

El presente proyecto de investigación debido a su índole y en consecuencia con los objetivos se compone de distintos tipos de investigación en sus distintas etapas, de tipo bibliográfico por la revisión realizada para los antecedentes, así como exploratorio por su nivel de profundidad y cualitativo por el tipo de evaluación a realizar con la información recopilada.

Es de tipo bibliográfico porque para la concepción del proyecto se tomó como bases estudios hallados en distintos bases de datos científicas como IEEE EXPLORE, ACM, WEB OF SCIENCE, SCOPUS así también como para el desarrollo de su marco teórico, dichos estudios aportan evidencia e información de alta relevancia a la investigación.

También se enmarca esta investigación en el tipo exploratorio porque se realizarán evaluaciones a distintas herramientas de pruebas automatizadas y plataformas de alojamiento de código con el fin de obtener la información necesaria para definir cuáles son las adecuadas para la propuesta a elaborar. Además, esta investigación servirá como referencia para futuros proyectos relacionados a la implementación de herramientas de integración y despliegue continuo.

Para finalizar otro tipo de estudio que será utilizado es el cualitativo ya que al evaluar las características de las herramientas y al obtener la información con respecto a la investigación se medirán parámetros de estas características, analizando así aspectos como similitudes, diferencias, ventajas y desventajas de estas.

2.3. Métodos y técnicas

2.3.1. Métodos

En la presente investigación para lograr los objetivos planteados se emplearán los siguientes métodos de análisis científicos: analítico, comparativo, cualitativo. La forma en que se ha contemplado el uso del método analítico es debido a que es necesario identificar las características y elementos que contienen las herramientas a estudiar para elaborar la propuesta de implementación.

Se tomó el método comparativo porque se hará una diferenciación entre las cualidades que puedan poseer o no las distintas herramientas y procesos que tienen la capacidad de elaborar canales de integración continua y despliegue automatizado.

El último método asociado a este estudio es el cualitativo pues se necesitan medir ciertos aspectos subjetivos, pero de gran importancia relacionados a las tecnologías y herramientas analizadas.

2.3.2. Técnicas

La técnica a utilizar en este trabajo investigativo será la entrevista, que se llevará a cabo en el Departamento de TI de la PUCESE donde se entablará un diálogo por separado con tres personas que laboran en dicho lugar el jefe del área y dos desarrolladores de software siguiendo un modelo de entrevista semiestructurado con

preguntas previamente elaboradas pero que se pueden adaptar al contexto, esto permitirá obtener información valiosa en consecuencia al uso del método cualitativo, sobre lo que se espera como resultado para la implementación de las herramientas y el proceso evaluado.

2.4. Población y muestra

2.4.1. Población

En esta investigación la población considerada consta del jefe del departamento de TI de la Pontificia Universidad Católica del Ecuador Sede Esmeraldas y dos desarrolladores de software del mismo departamento.

2.5. Descripción de instrumentos

El instrumento a utilizar será el cuestionario previamente elaborado de preguntas abiertas para realizar la entrevista semiestructurada en el caso del jefe del departamento de TI se usará un cuestionario y con los dos desarrolladores de software el instrumento será distinto.

Adicional como herramienta evaluadora de calidad de las distintas propiedades de las tecnologías que integrarán la propuesta y para garantizar que los resultados sean excelentes y se sigan altos estándares se usará la norma ISO/IEC 25010 [38] que permitirá comparar las características a tener en cuenta en cada herramienta al momento de realizar la elección, así pues entre las características principales y derivadas de esta norma se han tomado las siguientes para el apartado de Adecuación Funcional: Completitud Funcionalidad así se sabe si las herramientas cumplen todo lo que el usuario debe realizar, en la característica de Usabilidad: Capacidad de Aprendizaje con esto se podrá determinar la cantidad de documentación, guías y soporte para el uso de la herramienta, y finalmente en Portabilidad: Capacidad para ser Reemplazado para de esta forma medir en esta característica que prestaciones brindan las distintas herramientas.

2.6. Técnicas de procesamiento y análisis de datos

En la presente investigación se recopilará la información mediante la entrevista donde se usará un dispositivo para grabar la voz, luego estos datos serán transcritos a texto donde serán revisados y divididos con respecto a las preguntas planteadas, como uso de

repositorios de código, herramientas de pruebas, formas de despliegue utilizadas, así se podrá distinguir e interpretar las respuestas de los participantes, después de esto se procesará con el programa Microsoft Excel donde se usarán tablas para hacer un análisis profundo de los datos así como las comparaciones necesarias para la investigación para ello también se presentará los resultados en gráficos como pasteles para que su apreciación sea sencilla y comprensible.

Finalmente, todas las conclusiones de estas acerca del proceso realizado con los datos serán redactadas analizando más a detalle cada una de las respuestas y así interpretar de mejor forma lo mostrado previamente en los gráficos.

2.7. Normas éticas

La investigación planteada será realizada con originalidad y de forma personal, respetando las normativas éticas, así también el trabajo a realizar estará ligado a los distintos reglamentos y normas de la Pontificia Universidad Católica del Ecuador Sede Esmeraldas para proyectos de grado. De igual forma se respetarán los derechos de autor de cada uno de los artículos estudiados en esta investigación. Por último, se mantendrá la confidencialidad de la información que será utilizada únicamente para fines investigativos de este proyecto.

3. RESULTADOS

3.1. IDENTIFICACIÓN DE DISTINTAS FORMAS DE CANALES CI/CD

Pipeline de sitio web estático: Un sitio web estático es una aplicación que sirve contenido en las tecnologías fundamentales de la web HTML, CSS, Javascript, por tanto cualquier aplicación construida con un marco de trabajo o biblioteca moderna como React Angular, Vue, puede terminar siendo una web estática que no requiere estar soportada por un servidor web tradicional como Apache o Nginx para ser accesible, únicamente los archivos con el código correspondiente.El funcionamiento de una pipeline en un proceso de desarrollo de software empieza por los desarrolladores haciendo un push o pull request hacía un servicio de repositorios en línea, posterior a este paso se construye y se prueba la aplicación según los pasos que se definan, luego con los cambios actualizados se hace el despliegue automático hacia el servicio de hosting de sitios estáticos que se use y se publica a los usuarios finales. Esta fase de despliegue es una capa de abstracción de como cada servicio de este tipo procedimiento y lo automatiza.

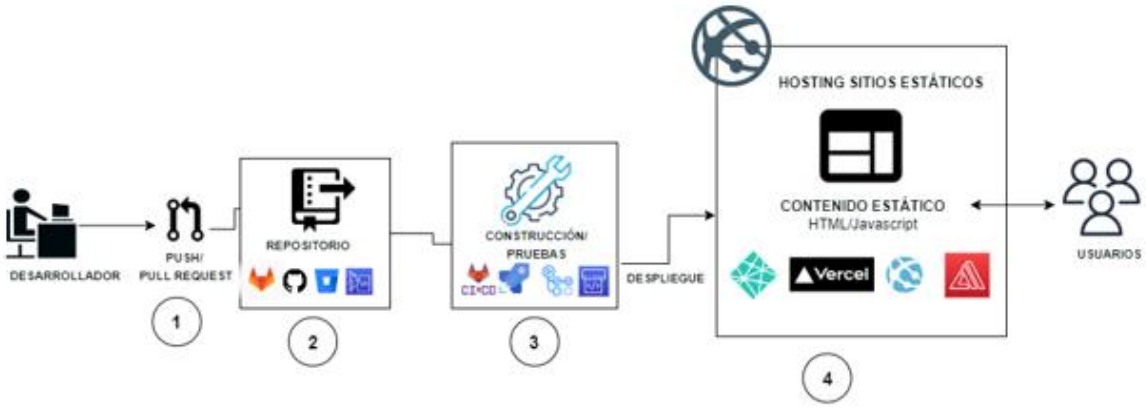

Figura 1 : Pipeline de sitio web estático

Pipeline Estándar: Este tipo de solución está orientado aplicaciones web un poco más completas donde hay procesamiento del lado de servidor, requiere una compilación y ejecución más robusta, la mayoría de estas soluciones integran contenedores en sus fases de integración continua para poder ejecutar los pruebas necesarias tanto unitarias como de integración y pruebas de extremo a extremo en caso de ser aplicaciones bases en interfaz de usuario, en el y despliegue continuo para colocar las aplicaciones en funcionamiento en los distintos ambientes , por tanto este tipo de pipeline requiere de más herramientas con más pasos para el aseguramiento de calidad y control de errores, así como monitoreo una vez puesta la aplicación en producción.

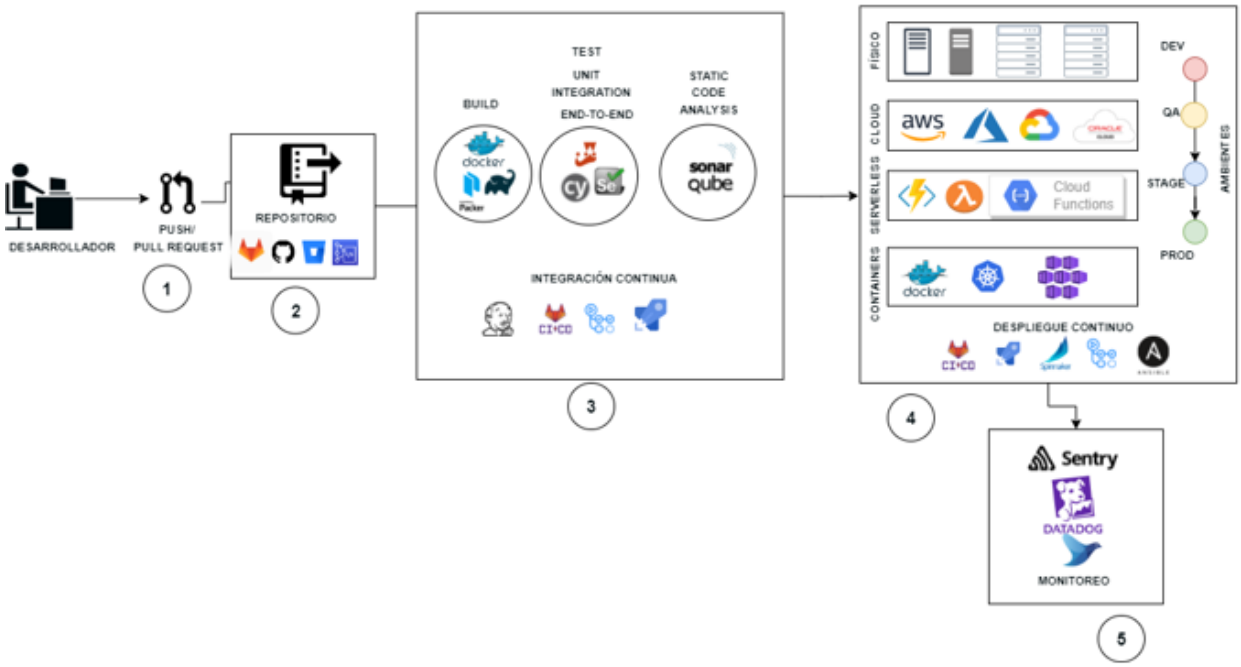

Figura 2 : Pipeline estándar

Pipeline con Infraestructura como código: Debido a las necesidades de reproducir ambientes de forma exacta la infraestructura que soporta ciertas aplicaciones también se versiona en código que hace posible esta creación, así pues, se puede crear y destruir con facilidad en conjunto con la aplicación integrándola en una pipeline de este tipo donde el código fuente manejado en el repositorio determina si se realizan cambios en la infraestructura que están en los distintos ambientes.

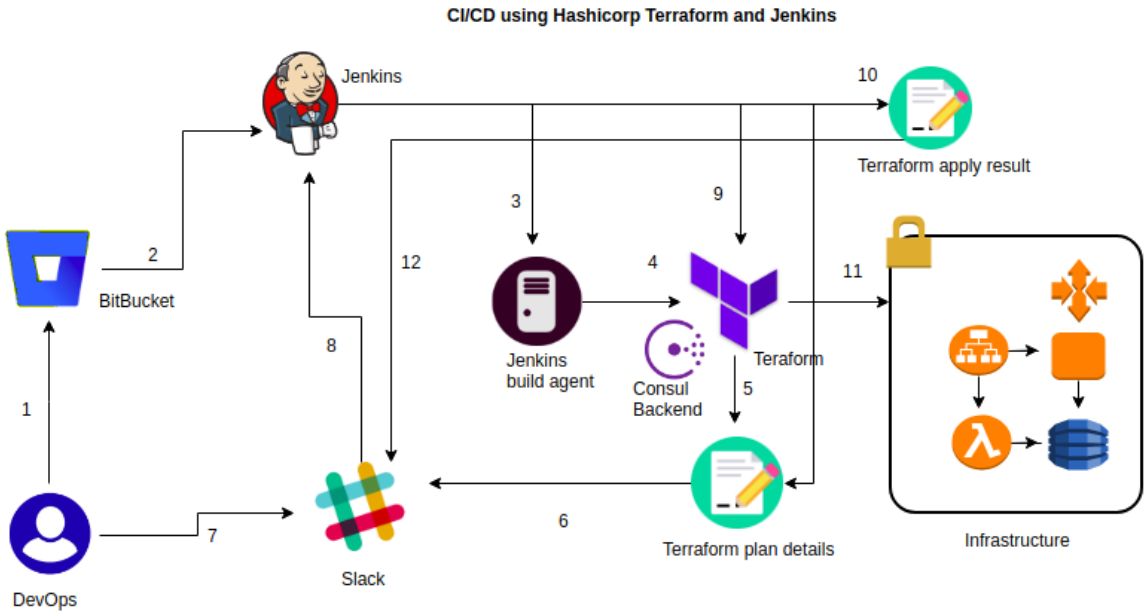

Figura 3: Pipeline con Infraestructura como Código [35]

Pipeline con ML/AI: Este tipo de solución es utilizada para automatizar con integración y despliegue continuo procesos correspondientes a Machine Learning e Inteligencia Artificial, también al tratamiento de datos.

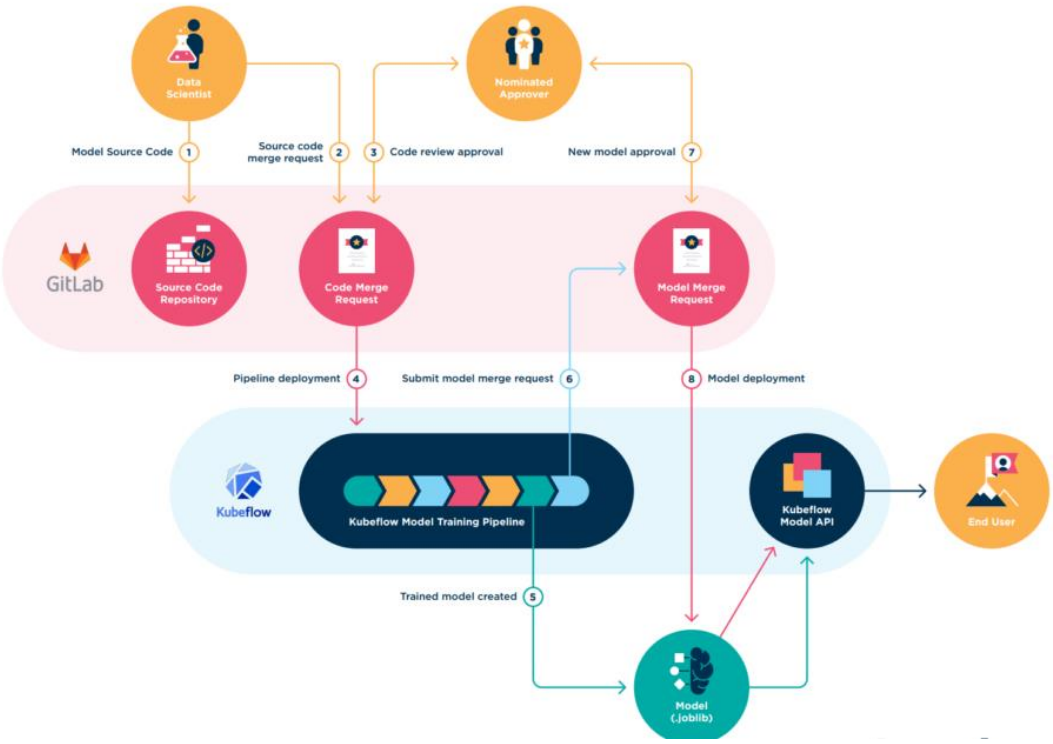

Figura 4: Pipeline con ML/AI [40]

Pipeline End to End (E2E): Estas son soluciones integrales con un solo proveedor Cloud el cual puede ser Amazon Web Services, Google Cloud Platform, Azure, estas plataformas proveen de todos los servicios y herramientas para construir una pipeline de principio a fin, por lo que se puede gestionar integración y despliegue continuo añadiendo solo unos pocos servicios de terceros en ciertas tareas.

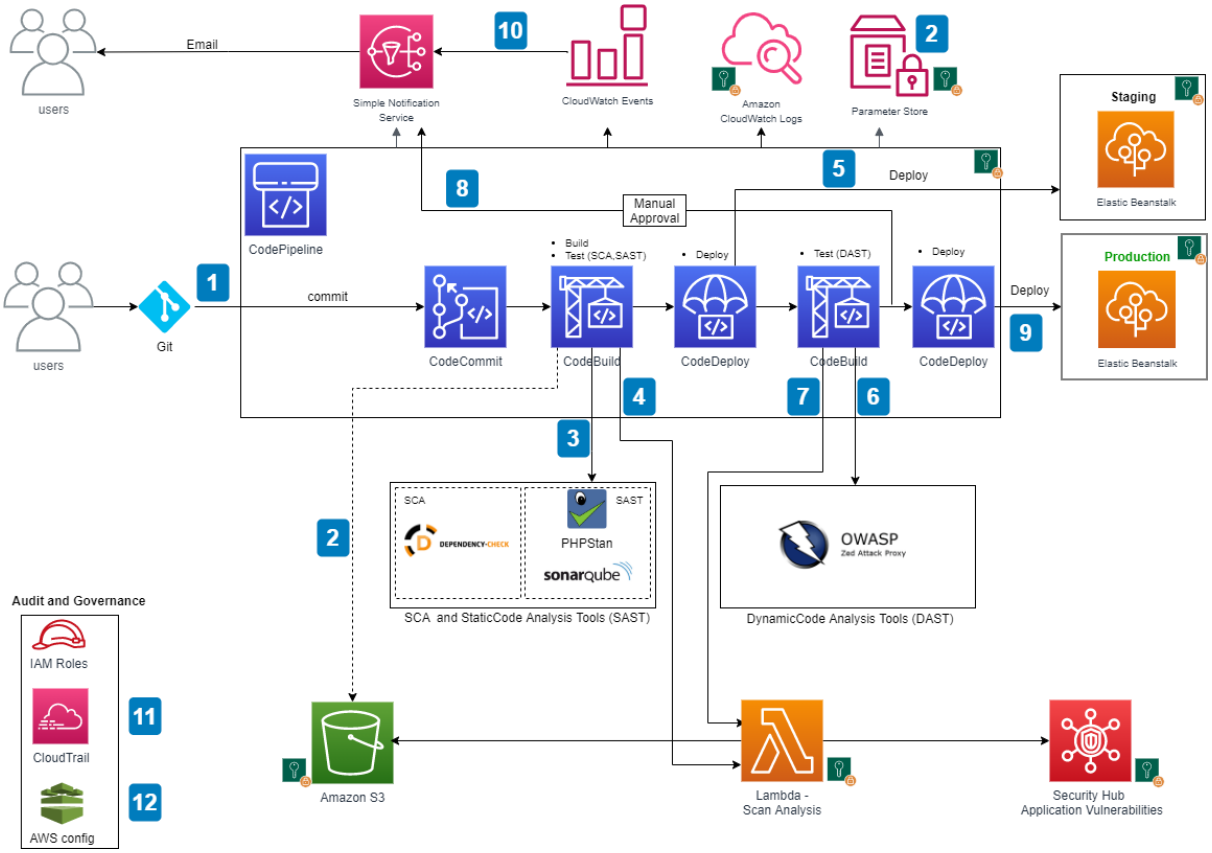

Figura 5: Pipeline End to End [41]

3.2. COMPARAR DISTINTAS OPCIONES DE CI/CD

Pipeline de sitio web estático: en lo que corresponde a la característica de **Adecuación Funcional** se ha tomado el parámetro **Completitud Funcional** al evaluar esta opción con este parámetro se encontró que no provee configuraciones de acceso para realizar despliegues en nubes específicas o servidores propios, su despliegue se realiza únicamente en un servicio de hosting de sitios estáticos por lo que su abanico de opciones se reduce, otra característica encontrada es que los proyectos desplegadas con este tipo de pipeline debe estar desarrollados únicamente con las herramientas web de la parte frontal de las aplicaciones HTML, CSS y JavaScript lo que también cerra la posibilidad a usar otro tipo de lenguajes de programación para el desarrollo de aplicaciones más complejas con backend o procesamiento del lado del servidor por lo tanto el criterio de aceptación es bajo, otro parámetro que se consideró para analizar esta opción es **Capacidad de Aprendizaje** en donde se identificó que para este tipo de pipeline la documentación es amplia, además que es un proceso con una complejidad baja debido que las configuraciones en sus pasos son sencillo así su nivel en este criterio es alto, también se analizó la **Capacidad para ser usado** este caso operar una pipeline básica no requiere de conocimientos especializados por lo que operarla se vuelve una tarea sencilla por lo tanto tiene un nivel alto en este tópico, debido a esto se adapta cumple satisfactoriamente este tópico, también se analizó el apartado de **Reusabilidad** para este caso la opción estudiada se adapta con un criterio de aceptación media debido a que si se pueden construir más software con este tipo de pipeline pero solo con un número limitado de tecnologías por lo que no alcanza un grado mayor de satisfacción en este punto, otra característica de análisis es la **Modularidad** como se ha observado en la investigación una pipeline es en sí un sistema integrados por un número determinado de pasos para los cuales se emplea distinto software en cada uno de ellos para esta característica se ha identificado que el grado de aceptación es medio debido a que si se puede reemplazar las distintas parte que lo componen como el manejador de repositorios de código, los servicios de prueba y el servicio de alojamiento.

Pipeline Estándar: para este tipo de flujo en las características analizadas se han obtenido los siguientes hallazgos en el apartado de **Completitud Funcional** debido a que esta opción

tiene la capacidad de integrarse a múltiples lenguajes de programación, herramientas de pruebas personalizadas, y despliegue en todo tipo de servicios y servidores de cualquier proveedor como propios, el nivel de aceptación califica como alto porque el usuario podrá cumplir con todas las tareas que desee realizar; otro punto analizado es la **Capacidad de Aprendizaje** en donde se pudo evidencia que existe documentación a disposición de como realizar los distintos procedimientos para implementar este tipo de pipeline, así como una comunidad lo suficientemente grande la cual colabora con el desarrollo y soporte de las mismas, en la característica de evaluación **Capacidad para ser usado** se identificó que las operaciones se pueden realizar con facilidad, es decir no existe barrera de entrada para usarlo por lo tanto el nivel de aceptación para este punto es un nivel alto; en el apartado de **Reusabilidad** el hallazgo obtenido es que esta opción se adapta muy bien a la construcción de software de distintas características y esta opción puede ser empleada e integrarse a la creación de más de un software al incluirse en el ciclo de desarrollo del mismo, por tanto se identificó que el nivel de aceptación en este punto es alto; **Modularidad** en esta característica debido a que esta opción es versátil en cuanto a sus componentes dando posibilidad de cambiarlos por otros de acuerdo a las necesidades y el contexto y generando un impacto mínimo el grado de aceptación se ha considerado alto.

Pipeline con Infraestructura como código: según las características seleccionadas en la herramienta evaluadora se han identificado de siguiente modo como esta opción es aceptable con estos criterios para lo cual en **Completitud Funcional** se identificó que la herramienta provee de la creación de infraestructura únicamente proveedores cloud y las pruebas realizadas no se pueden personalizar como desea el usuario, por el contrario crear infraestructura es una tarea que el usuario no busca realizar de acuerdo al contexto en el que se encuentra por tanto el nivel de aceptación para este tipo de pipeline es bajo; para el siguiente criterio **Capacidad de Aprendizaje,** si bien este tipo de herramientas de IaaC se encuentran bien documentadas, no se ha identificado grandes comunidades a la cuales acudir en caso de dudas y que generen soporte por que el indicador de aceptación se ha considerado como nivel medio, para el apartado de **Capacidad para ser usado** se identificó que este tipo de pipeline en cuanto a su operación se puede realizar con facilidad pero requiere de ciertas configuraciones y detalles los cuales requieren un buen grado de experiencia de los desarrolladores por lo tanto en nivel de aceptación se ha considerado como medio; en la

evaluación de **Reusabilidad** de este opción se obtuvo que si se puede emplear en la creación de otro software pero tiene que seguir ciertos lineamientos no presenta tanta versatilidad por lo que su nivel de aceptación se ha considerado como medio; para la característica de **Modularidad** debido a que este tipo de pipeline necesita funcionar con herramientas específicas y la variación de alguna de ellas si genera un impacto se ha considerado que el nivel de aceptación es bajo.

Pipeline End to End con AWS: en esta opción la evaluación de la característica de **Completitud funcional** se ha determinado que puede cumplir con todas la tareas que requiere el usuario en cuestión por tanto su nivel de aceptación es alto; por otra parte en la característica de **Capacidad de Aprendizaje** debido a que estos servicios si están bien documentado por sus creadores no cuentan con una comunidad grande ni su uso es muy extendido por lo tanto la evaluación en este punto tiene un nivel de aceptación medio, en el siguiente criterio evaluado **Capacidad para ser usado** se identificó que el producto se puede operar con facilidad pero requiere conocimientos específicos de las herramientas por parte de los usuarios por lo que su nivel de aceptación se ha considerado como medio; para el criterio de **Reusabilidad** de acuerdo con la evaluación realizada se puede emplear la misma pipeline para la creación de otro software sí pero se debe mantener el mismo entorno de la nube por lo que el criterio de aceptación para este apartado se ha determinado con un nivel medio; finalmente en la característica de **Modularidad** para esta opción se ha considerado que es complicado cambiar ciertas piezas debido a que están atadas a un mismo proveedor por tanto el nivel de aceptación considerado es bajo.

Tipos de Pipelines	Usabilidad					
	Capacidad de Aprendizaje			Capacidad para ser usado		
	Alto	Medio	Bajo	Alto	Medio	Bajo
Pipeline de sitio web estático	X			X		
Pipeline Estándar	X			X		
Pipeline con Infraestructura como código		X			X	
Pipeline con ML/AI						
Pipeline End to End con AWS		X			X	

Tabla 1 Comparativa de tipos de pipelines de acuerdo con el criterio de Usabilidad

Tipos de Pipelines	Mantenibilidad					
	Reusabilidad			Modularidad		
	Alto	Medio	Bajo	Alto	Medio	Bajo
Pipeline de sitio web estático		X			X	
Pipeline Estándar	X			X		
Pipeline con Infraestructura como código		X				X
Pipeline End to End con AWS		X				X

Tabla 2 Comparativa de tipos de pipelines de acuerdo con el criterio de Mantenibilidad

Tipos de Pipelines	Adecuación Funcional		
	Completitud Funcional		
	Alto	Medio	Bajo
Pipeline de sitio web estático			X
Pipeline Estándar	X		
Pipeline con Infraestructura como código			X
Pipeline End to End con AWS	X		

Tabla 3 Comparativa de tipos de pipelines de acuerdo con el criterio de Adecuación Funcional

3.3. RESUMEN DE ENTREVISTA

En el proceso de la investigación se tenía previsto aplicar la técnica de la entrevista para recopilar la mayor información posible y determinar con el mejor grado de exactitud que se pueda las necesidades del Departamento de TI de la PUCESE para elaborar la propuesta a plantear que se dará como resultado de todo lo mencionado en este documento. Se hizo una entrevista semiestructurada con las mismas preguntas a dos colaboradores del Departamento, los cuales cumplían los roles de Desarrollador de Software y Administrador de Sistemas respectivamente donde como resultados se pudieron determinar varios aspectos relevantes como el número de involucrados en tareas de desarrollo de software es de 2 a 3 personas, así también para labores de administración y operación de sistemas, infraestructura, servidores y demás aspectos relacionados se cuenta con 2 personas, por tanto el equipo completo que trabaja sobre el software se integra de como máximo 5 personas, siendo un equipo relativamente pequeño. La interacción existente entre las personas de desarrollo y las de operaciones a pesar de comprenderse como labores separadas la comunicación es efectiva y los ciclos de retroalimentación son cortos debido a que el espacio físico y tamaño del equipo se presta para una colaboración inmediata.

También se pudo identificar que el gestor de repositorios utilizado actualmente es Gitlab pero que antes se hacía uso de Bitbucket así fue confirmado por ambos colaboradores, en cuanto a los lenguajes de programación utilizados se coincidió en el uso de PHP con el framework Yii2 pero que en la actualidad se está usando Nodejs con el framework Express para desarrollo del lado del servidor (backend) y que se prevé el uso de Reactjs como tecnología de frontend, también el uso de bases de datos relacionales como MariaDB, Postgress, SQL Server y finalmente como herramienta de contenedores Docker. También se logró identificar que las aplicaciones y servicios que provee el Departamento para sus usuarios son alojadas mediante contenedores en máquinas virtuales alojadas en su propio Datacenter con un clúster de Proxmox así como otra máquina virtual en el Datacenter del proveedor Cedia entre las cuales se reparten entornos de producción y preproducción.

En cuanto al control de calidad que actualmente se emplea en el desarrollo de software es manual, se realizan pruebas de aceptación por parte de usuario, no existen aún procesos automatizados, y para la puesta en producción de los sistemas también se hace de forma manual accediendo a los servidores por conexiones remotas y bajando los cambios del repositorio mediante la herramienta git. Se identificó también se emplean herramientas como Zabbix y UptimeRobot para el monitoreo de la infraestructura y servicios, así como para la generación de alertas. Como último punto de la entrevista se recabó información acerca del proceso de seguridad en las aplicaciones el cual está cubierto a nivel de infraestructura mediante firewall, distintos bloqueos de peticiones, limitaciones para evitar ataques.

3.4. Propuesta de Implementación de Pipeline

Una vez indagado en los tópicos necesarios de la investigación tanto en las bases teóricas, como la recopilación de los requerimientos del lugar de aplicación de este trabajo investigativo mediante la técnica de la entrevista, se tiene una propuesta inicial de pipeline de integración y despliegue continuo para la automatización de ciertos procesos asociados a la construcción de software, con el objetivo de ahorrar tiempo en ciertos procesos que de momento se realizan de forma manual, para lo cual tomando como punto de partida lo manifestado por los entrevistados el desarrollador de software y el administrador de sistemas, se proponen las siguientes herramientas: mantener el uso de Gitlab como gestor de repositorios de código debido a que es la que se usa en la actualidad por lo tanto los usuarios se encuentran familiarizados con la herramienta y migrar a otra generaría cierta pérdida de recursos como, tiempo y productividad, así también se propone usar la estrategia de ramas de Git, llamada Gitlab Flow para una mejor gestión y separación de ramas de la base de código de acuerdo a las funcionalidades donde se creará una rama para cada una por ejemplo: “feature/example” y los ambientes que serán usados en las aplicaciones donde se tendrá las ramas “develop” para todo lo correspondiente a desarrollo y la rama “master” para producción, otra motivación para mantener el uso de Gitlab son las herramientas de Integración y Despliegue Continuo que ya vienen incluidas, también llamadas Gitlab CI/CD. Otra de las tecnologías consideradas en la propuesta es mantener el uso de Docker para el manejo de contenedores, según lo

expresado en la entrevista ya se tiene un tiempo considerable usando la herramienta tanto para desarrollo como para producción y las prestaciones han sido favorables, además debido a que en la actualidad está es la herramienta más usada. Siguiendo el flujo de canalización que debe tomar esta propuesta, en el apartado de integración continua se usará también Cypress.io para pruebas automatizadas del software debido a dos razones principales la primera es que se usa con el lenguaje de programación JavaScript del cual ya hacen uso los desarrolladores en el Departamento de TI de la Pucese, debido a que ya se han escrito varias aplicaciones con frameworks de este lenguaje y se prevé seguir haciendo uso del mismo, la segunda razón es por la capacidad de integración que tiene este herramienta para casi cualquier proyecto por lo que puede ser empleado con facilidad, otra herramienta incluida en la propuesta es SonarQube para análisis de código estático pues permite detectar rápidamente fallos en la codificación, malas prácticas y demás inconvenientes que se pueden mitigar con su uso.

4. DISCUSIÓN

En la investigación [35] que tuvo como objetivo dar un aproximación de como diversos equipos de desarrollo están utilizando prácticas de integración y despliegue, se evidenció la estructura y los componentes con los que debe contar un canal de integración y despliegue continuo, así también cuáles son las herramientas más utilizadas, con este insumo se partió hacia el análisis del contexto existente en el Departamento de TI de la PUCESE en cuánto a los requerimientos actuales en el campo desarrollo de software, con el objetivo de limitar los enfoques buscando entre los que se adecuarán con mayor precisión al marco establecido en esta investigación. Siendo así que una opción equilibrada entre lo sencillo y lo complejo sería la que alcance las mejores prestaciones para este caso.

Por su parte en el estudio [10] se obtuvo muestras a detalle de como 5 compañías están aplicando buenas prácticas de integración y despliegue continuo, además de como organizan y diseñan estos ambientes basados en el tamaño de sus equipos, las tecnologías que usan, los servicios que prestan y el tipo de clientes que tiene de modo que guarda una relación intrínseca con la presente investigación debido a que las necesidades son parecidas, para afianzar la relación este estudio concluye recomendando un balance entre velocidad y calidad del código lo cual se tiene en cuenta en la presente investigación debido a que las herramientas utilizadas permiten controlar la calidad del código y automatizar los procesos de lanzamiento a producción otorga este factor de velocidad pero estableciendo el balance mencionado anteriormente al rechazar las construcciones y despliegues que contengan errores o donde la calidad del código no cumpla con los estándares establecidos.

De acuerdo con [36] donde se plantea como están siendo implementadas herramientas análisis de código estático en canales de integración CI de forma que aproxima como se debería adoptar de forma efectiva y eficiente en otros entornos como el de esta investigación, así también recomienda que los desarrolladores deben fijar cuales son las comprobaciones que se adaptan al proyecto actual, de este modo se actuó en consecuencia con la investigación al realizar las configuraciones de la herramienta de análisis de código estático incluida en la propuesta donde se tuvo en cuenta los requerimientos específicos demandados por el Departamento de TI de la PUCESE y así fijar las validaciones adecuadas en el proyecto evitando ralentizar el proceso de CI o validaciones innecesarias.

En la publicación científica [21] se aborda a automatización de pruebas en entornos de desarrollo continúo y como se llevó a cabo con éxito en el caso particular de una empresa aquí se revelan ciertos factores críticos que tiene gran impacto en estos procesos como el enfoque incremental, el esfuerzo de todo el equipo, la elección de la herramienta de prueba y la telemetría. Sin embargo, en el caso de los proyectos en el Departamento de TI de la PUCESE.

Finalmente, en la investigación [37] se hace un análisis de las dificultades que tiene implementar prácticas CI/CD en equipos mencionando que el factor más importante es el consumo de tiempo en las fases de pruebas por lo que el enfoque tomado para la propuesta de implementación en el contexto de la investigación PUCESE las pruebas que se realizan son las adecuadas para evitar que los desarrolladores carguen con pruebas tediosas que demanden grandes esfuerzos de tiempo.

5. CONCLUSIONES

Una vez abordada la presente investigación en su totalidad, se puede establecer las siguientes conclusiones:

Se obtuvieron las distintas formas de crear canales de integración y despliegue continuo entre las que de acuerdo con el contexto de la investigación se limitaron a cinco opciones para su análisis.

Basado en las opciones obtenidas y según los apartados seleccionados de la norma ISO/IEC 25010, se establecieron tablas comparativas que permitieron el análisis de las características.

Se conocieron los requerimientos para proceso el desarrollo de software del Departamento de TI de la PUCSE aplicando los instrumentos de recopilación de datos lo que permitió tener mejor visión del panorama y así se pudo adaptar la solución planteada lo más posible al entorno.

La propuesta otorgada permitió automatizar la integración y despliegue continuo de software, así como asegurar la calidad del código mediante las pruebas establecidas, de modo que brindará beneficios como ahorro de tiempo y reducción de errores, si bien se presenta como una solución a medida para la presente situación esta puede ser extendida y modificada a futuro cuando los requerimientos lo demanden.

6. RECOMENDACIONES

Se recomienda realizar un proceso de mejora continua a la propuesta entregada adicionando nuevas herramientas y nuevos pasos en la canalización con el objetivo seguir mejores prácticas en desarrollo de software y en consecuencia elevar cada vez más los estándares de calidad en el software producido. Así mismo se incita a los desarrolladores a realizar configuraciones adecuadas acordes con el proyecto en sus editores de código o entornos de desarrollo integrado usando formateadores de códigos para que este se organice mejor y plugins que puedan dar aviso del cumpliendo de reglas de sintaxis y demás especificaciones de código a considerar.

También se aconseja explorar y ampliar el abanico de pruebas realizadas sea agregando pruebas de seguridad dentro los canales de integración creados de modo que las aplicaciones creadas sean más robustas y se eviten ciertas vulnerabilidades que se pueden generar en el proceso de desarrollo de software.

Así también otro punto de mejora sería empezar con el uso de metodologías ágiles en el desarrollo de software como Scrum de modo que se puedan organizar en iteraciones los distintos requerimientos solicitados y los usuarios finales puedan obtener de forma más rápida nuevas funcionalidades.

Se recomienda también la migración a entornos de Cloud, con alguno de los proveedores lideres en el mercado de tal manera que estén a disposición mejores y más herramientas de automatización no solo en la integración y despliegue de software si no también en la creación de infraestructura.

Una recomendación para futuras investigaciones que exploren temas similares que incluyan CI/CD generando microservicios a gran escala, es abordar la temática de service mesh o malla de servicios con herramientas como Istio que permitan escalar y aumentar la confiabilidad de este tipo de software.

7. REFERENCIAS

[1] P. Agrawal and N. Rawat, "Devops, A New Approach to Cloud Development Testing," *IEEE International Conference on Issues and Challenges in Intelligent Computing Techniques, ICICT 2019*, 2019, doi: 10.1109/ICICT46931.2019.8977662.

[2] N. Niu, S. Brinkkemper, X. Franch, J. Partanen, and J. Savolainen, "Requirements engineering and continuous deployment," *IEEE Software*, vol. 35, no. 2, pp. 86–90, 2018, doi: 10.1109/MS.2018.1661332.

[3] "Jenkins." https://www.jenkins.io/ (accessed Nov. 15, 2021).

[4] M. Zarour, N. Alhammad, M. Alenezi, and K. Alsarayrah, "A research on DevOps maturity models," *International Journal of Recent Technology and Engineering*, vol. 8, no. 3, pp. 4854–4862, 2019, doi: 10.35940/ijrte.C6888.098319.

[5] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, A. Zaidman, and H. C. Gall, "Context is king: The developer perspective on the usage of static analysis tools," *25th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2018 - Proceedings*, vol. 2018-March, pp. 38–49, 2018, doi: 10.1109/SANER.2018.8330195.

[6] L. de Aguiar Monteiro, "A proposal to systematize introducing DevOps into the software development process," pp. 269–271, 2021, doi: 10.1109/icse-companion52605.2021.00124.

[7] A. Wiedemann, N. Forsgren, M. Wiesche, H. Gewald, and H. Krcmar, "Research for practice: The Devops phenomenon," *Communications of the ACM*, vol. 62, no. 8, pp. 44–49, Aug. 2019, doi: 10.1145/3331138.

[8] Cohn, "Surviving with Agile: Software development using SCRUM," p. 503, 2019.

[9] "Manifesto for Agile Software Development." https://agilemanifesto.org/ (accessed Dec. 01, 2021).

[10] L. E. Lwakatare *et al.*, “DevOps in practice: A multiple case study of five companies,” *Information and Software Technology*, vol. 114, no. April, pp. 217–230, 2019, doi: 10.1016/j.infsof.2019.06.010.

[11] Microsoft, “What is Devops,” *What is Devops*, 2021. https://azure.microsoft.com/es-es/overview/what-is-devops/ (accessed Jun. 28, 2021).

[12] Amazon Web Services, “What is DevOps,” *En que consiste DevOps*, 2021. https://aws.amazon.com/es/devops/what-is-devops/ (accessed Jun. 28, 2021).

[13] Microsoft, “Agile vs DevOps,” *Agile vs DevOps*, 2021. https://azure.microsoft.com/es-es/overview/devops-vs-agile/ (accessed Jun. 28, 2021).

[14] B. Beyer, N. R. Murphy, D. K. Rensin, K. Kawahara, and S. Thorne, “Edited by The Site Reliability Workbook Practical Ways to Implement SRE.”

[15] R. Majumdar, R. Jain, S. Barthwal, and C. Choudhary, “Source code management using version control system,” *2017 6th International Conference on Reliability, Infocom Technologies and Optimization: Trends and Future Directions, ICRITO 2017*, vol. 2018-Janua, pp. 278–281, 2018, doi: 10.1109/ICRITO.2017.8342438.

[16] “About repositories - GitHub Docs.” https://docs.github.com/es/repositories/creating-and-managing-repositories/about-repositories (accessed Dec. 01, 2021).

[17] “About Git - GitHub Docs.” https://docs.github.com/en/get-started/using-git/about-git (accessed Dec. 01, 2021).

[18] “Flujo de trabajo de Git | Atlassian Git Tutorial.” https://www.atlassian.com/es/git/tutorials/comparing-workflows (accessed Dec. 01, 2021).

[19] “A successful Git branching model » nvie.com.” https://nvie.com/posts/a-successful-git-branching-model/ (accessed Dec. 01, 2021).

[20] T. Saha and R. Palit, “Practices of Software Testing Techniques and Tools in Bangladesh Software Industry,” *2019 IEEE Asia-Pacific Conference on Computer Science and Data Engineering, CSDE 2019*, 2019, doi: 10.1109/CSDE48274.2019.9162355.

[21] Y. Wang, M. Pyhajarvi, and M. v. Mantyla, “Test Automation Process Improvement in a DevOps Team: Experience Report,” *Proceedings - 2020 IEEE 13th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2020*, pp. 314–321, 2020, doi: 10.1109/ICSTW50294.2020.00057.

[22] “The Practical Test Pyramid.” https://martinfowler.com/articles/practical-test-pyramid.html (accessed Dec. 01, 2021).

[23] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm, “Continuous deployment at Facebook and OANDA,” *Proceedings - International Conference on Software Engineering*, pp. 21–30, 2016, doi: 10.1145/2889160.2889223.

[24] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 2010.

[25] S. A. I. B. S. Arachchi and I. Perera, “Continuous integration and continuous delivery pipeline automation for agile software project management,” *MERCon 2018 - 4th International Multidisciplinary Moratuwa Engineering Research Conference*, pp. 156–161, 2018, doi: 10.1109/MERCon.2018.8421965.

[26] “DevOps Pipeline | Atlassian.” https://www.atlassian.com/devops/devops-tools/devops-pipeline (accessed Dec. 01, 2021).

[27] B. Fitzgerald and K. J. Stol, “Continuous software engineering: A roadmap and agenda,” *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017, doi: 10.1016/j.jss.2015.06.063.

[28] “Continuous Integration.” https://martinfowler.com/articles/continuousIntegration.html (accessed Dec. 01, 2021).

[29] “About merge methods on GitHub - GitHub Docs.” https://docs.github.com/es/repositories/configuring-branches-and-merges-in-your-repository/configuring-pull-request-merges/about-merge-methods-on-github (accessed Dec. 01, 2021).

[30] “About merge methods on GitHub - GitHub Docs.” https://docs.github.com/es/repositories/configuring-branches-and-merges-in-your-repository/configuring-pull-request-merges/about-merge-methods-on-github (accessed Dec. 01, 2021).

[31] M. Pagels, V. Eloranta, and J. Itkonen, “Focus: release engineering,” 2015.

[32] “Deployment Strategies - Deployments | Developer Guide | OpenShift Container Platform 3.7.” https://docs.openshift.com/container-platform/3.7/dev_guide/deployments/deployment_strategies.html (accessed Dec. 01, 2021).

[33] “Estrategias de implementación y de prueba de aplicaciones.” https://cloud.google.com/architecture/application-deployment-and-testing-strategies (accessed Dec. 01, 2021).

[34] “Intro to Deployment Strategies: Blue-Green, Canary, and More | Harness.” https://harness.io/blog/blue-green-canary-deployment-strategies/ (accessed Dec. 01, 2021).

[35] M. Shahin, M. Ali Babar, and L. Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” *IEEE Access*, vol. 5, no. Ci, pp. 3909–3943, 2017, doi: 10.1109/ACCESS.2017.2685629.

[36] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. di Penta, “How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines,” *IEEE International Working Conference on Mining Software Repositories*, pp. 334–344, 2017, doi: 10.1109/MSR.2017.2.

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

[38] "ISO 25010." https://iso25000.com/index.php/normas-iso-25000/iso-25010 (accessed Dec. 01, 2021).

[39] "Immutable Infrastructure CI/CD Using Hashicorp Terraform and Jenkins - DZone DevOps." https://dzone.com/articles/immutable-infrastructure-cicd-using-hashicorp-terr?utm_source=dzone&utm_medium=article&utm_campaign=CI%2FCD%20content%20cluster (accessed Nov. 24, 2021).

[40] "MLOps: CI/CD for Machine Learning Pipelines & Model Deployment with Kubeflow - Growing Data." https://growingdata.com.au/mlops-ci-cd-for-machine-learning-pipelines-model-deployment-with-kubeflow/ (accessed Dec. 08, 2021).

[41] "Building end-to-end AWS DevSecOps CI/CD pipeline with open source SCA, SAST and DAST tools | AWS DevOps Blog." https://aws.amazon.com/es/blogs/devops/building-end-to-end-aws-devsecops-ci-cd-pipeline-with-open-source-sca-sast-and-dast-tools/ (accessed Dec. 08, 2021).

ANEXOS

Entrevista 1

Desarrollador de Software

¿Cuántas personas trabajan en el desarrollo de software?

Otro compañero hacia el frontend y backend con PHP Yii2, yo hacia backend con Nodejs, Control de Calidad Jefe del Departamento de TI, total 3 personas.

¿Cuántas personas trabajan en la administración u operaciones del software? Administración de Servidores una persona y apoyo otra persona, total de 2 personas.

¿Cómo es la interacción entre Desarrollo y Operaciones?

La infraestructura ya está lista para las aplicaciones, como estamos en un espacio reducido la comunicación fluye, conversamos o nos reunimos y cualquier pedido del área de desarrollo ellos lo ponen inmediatamente a nuestra disposición para que podamos sin problemas desplegar aplicación o probar un servicio.

¿Qué gestor de repositorios se utiliza en el departamento?

Primero usábamos Bitbucket existen algunas aplicaciones ahí. En la actualidad usamos Gitlab y con planes de migrar todo ahí.

¿Qué tecnologías se usan el proceso el desarrollo de software?

Estamos utilizando PHP con YII2, Bases de Datos MariaDB, SQL SERVER, Postgres, Backend Nodejs con Express Framework, Contenedores Docker.

¿Cómo alojan las aplicaciones del departamento?

Dentro de contenedores que están en servidores ubicados unos el Datacenter del departamento los cuales son físicos y también ubicados en la nube de Cedia

¿Cómo es el proceso de control de calidad en el desarrollo de software?

Las pruebas las hacemos en entorno local de forma manual, aun no hay proceso automático, una vez probadas manualmente por el desarrollador, se probaban con los usuarios del sistema y una vez que dan el Ok y alguna retroalimentación se hacían las correcciones y se hacia el despliegue.

¿Qué proceso siguen para poner en producción las aplicaciones?

El desarrollo se hace una rama que puede llamarse Desarrollo y una vez que se termina se pasa a la rama Master mediante un Merge que lo aprueba el Jefe del Departamento TI y una vez que está aprobado es desplegado en producción.

¿Cómo manejan las alertas y comunicación de las aplicaciones?

Hay un sistema que se llama Zabbix y ahí están configuradas las alertas que se muestran en una pantalla para estar atentos.

¿Qué proceso de seguridad se emplea en las aplicaciones?

Certificados SSL para las aplicaciones Web, control de SQL Injection, carga masiva de solicitudes si hay un determinado número de peticiones desde un mismo sitio controlar o impedir realizarlas por un tiempo.

Entrevista 2

Administrador de Redes y Servidores

¿Cuántas personas trabajan en el desarrollo de software?

Directamente en el desarrollo de software en la actualidad hay una persona, (otro puesto vacante).

¿Cuántas personas trabajan en la administración u operaciones del software?

En Administración de Operaciones yo y apoyo como otra persona más, en ocasiones interviene el Jefe del Departamento de TI.

¿Cómo es la interacción entre Desarrollo y Operaciones?

La dinámica consiste en que yo me encargo de colocar el servicio que la mayoría son contenedores y luego viene la personalización, nosotros ponemos la infraestructura y me encargo de que este en producción y sea visible al público.

¿Qué gestor de repositorios se utiliza en el departamento?

Bitbucket y Gitlab.

¿Qué tecnologías se usan el proceso el desarrollo de software?

Contenedores Docker, Yii2, React, PHP y las APIs NodeJS

¿Cómo alojan las aplicaciones del departamento?

Tenemos una máquina que nos provee Cedia que es desde VMWare y administramos por ssh, la mayoría de los servicios están en esa máquina con contenedores y existe un proxy reverso que redirige a cada servicio, tenemos intranet, los repositorios(tesis) Cluster de proxmox en el Datacenter - Actualmente para servicios de internos

¿Cómo es el proceso de control de calidad en el desarrollo de software?

Cuando ya está listo el desarrollo se le hacen pruebas de seguridad, test de estrés y lo ponemos a monitorear de manera constante, una vez que funciona bien, el servicio queda ahí

¿Qué proceso siguen para poner en producción las aplicaciones?

Una vez que está listo generalmente las pruebas las hacemos en máquinas del clúster de proxmox dependiendo del servicio por ejemplo página web se crea un backup y luego se crea el mismo servicio en cedia y se carga el backup, luego de esto se tiene que registrar en él .conf de Nginx el nuevo servicio y luego los registros de DNS y dominios para que se accesible publicamente - en el clúster de proxmox tenemos esta máquina de preproducción y otra máquina de producción

¿Cómo manejan las alertas y comunicación de las aplicaciones?

Zabbix y Uptime Robot conectados con un bot de telegram cuando se cae un servicio nos notifica, en este bot estamos todos para recibir las alertas y resolver de forma inmediata

¿Qué proceso de seguridad se emplea en las aplicaciones?

Estamos pendientes con las vulnerabilidades existentes, y también de la cultura del desarrollador de evitar ataques sql injection, seguridad de firewall, y bloque de peticiones excesivas, y escáneres de puerto también los bloquea, el firewall y otros servicios, limitar el tipo de archivos que se suben, por ejemplo, en intranet se podían subir .exe

¿Cómo se despliegan los cambios en las aplicaciones?

Los cambios se bajan de manera manual al servidor por el desarrollador ingresando al servidor y haciendo git pull