Universidad Internacional de La Rioja
Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Desarrollo y Operaciones (DevOps)

Automatización del ciclo de integración y despliegue continuos de una aplicación móvil desarrollada en Flutter

Trabajo fin de estudio presentado por:	Sayri Cachiguango Maigua
Tipo de trabajo:	Desarrollo práctico
Director/a:	Rafael Merlo Loranca
Fecha:	20 de julio de 2022

Resumen

En el contexto del desarrollo de software tradicional, los equipos de desarrolladores que trabajan sobre el mismo proyecto de forma independiente a menudo reportan problemas a la hora de la integración de cada una de sus partes. Debido a esta independencia y metodología tradicional, los proyectos suelen ser presentados hasta el final del ciclo para su integración y despliegue, lo que a su vez repercute en gastos innecesarios y generación de errores.

Al tratarse de un aplicativo móvil mantenido por un kit de desarrollo de software que construye aplicaciones multiplataforma, la entrega y despliegue continuo depende de la plataforma para la que vaya a ser desplegada y probada, pues este proceso tendría que realizarse por cada plataforma.

La integración y despliegue continuo (CI/CD) surge como una práctica DevOps para resolver los problemas de las metodologías tradicionales con el objetivo de mejorar la calidad y entrega de un producto de software, de la mano de las metodologías ágiles y herramientas DevOps.

Este trabajo de fin de máster presenta la creación de un entorno que automatiza el ciclo de integración y despliegue continuo en el desarrollo de un aplicativo móvil, expone qué es y cómo se consigue este nuevo enfoque de desarrollo de software. También en el cual se explica los conceptos básicos de las tecnologías necesarias para la implementación de un pipeline y posteriormente resume algunas de las herramientas populares de código abierto utilizadas para CI/CD.

Palabras clave: DevOps, integración continua, despliegue continuo, pipeline CI/CD.

Abstract

In the context of traditional software development, where teams of developers worked on the same project independently, often report problems when integrating each of its parts. Due to this independence and traditional methodology, projects are often submitted until the end of the cycle for integration and deployment, it generates unnecessary costs and many errors.

Continuous Integration and Deployment (CI/CD) emerges as a DevOps practice to solve the problems of traditional methodologies to improve the quality and delivery of a software product, hand in hand with agile methodologies and DevOps tools.

This master's thesis presents the creation of an environment that automates the integration cycle and continuous deployment in the development of a mobile application, exposes what it is and how to achieve this innovative approach to software development. It also explains the basic concepts of the technologies required for the implementation of a pipeline and then summarizes some of the popular open-source tools used for CI/CD.

Keywords: DevOps, continuous integration, continuous deployment, CI/CD pipeline

Índice de contenidos

1. Introducción 9
1.1. Justificación del trabajo 10
1.2. Planteamiento del problema 10
1.3. Estructura de la memoria 12
2. Contexto y estado del arte 13
2.1. Contextualización y antecedentes 13
2.1.1. Ciclo de vida de desarrollo de sistemas 13
2.1.2. Metodología ágil y prácticas DevOps 14
2.1.3. Planificación y colaboración 17
2.1.4. Implementación de Software 17
2.1.5. Control de versiones con Git 18
2.1.6. Integración continua 18
2.1.7. Integración continua como un proceso de desarrollo 19
2.1.8. Entrega continua 21
2.1.9. Despliegue continuo 22
2.1.10. Entrega continua vs Despliegue continuo 22
2.1.11. Pipeline de CI/CD automatizado 23
2.1.12. Ventajas de la automatización de un pipeline de CI/CD 23
2.1.13. CI/CD en aplicación móviles 24
2.1.14. Criterios para evaluar herramientas CI/CD para aplicaciones móviles 25
2.1.15. Aplicaciones Móviles 26
2.2. Trabajos relacionados 28
2.3. Conclusiones del estado del arte 34
3. Objetivos y metodología de trabajo 35

3.1. Objetivo general....35
3.2. Objetivos específicos....35
3.3. Metodología del trabajo....35
4. Desarrollo específico de la contribución....39
4.1. Planificación / Análisis / Requisitos....39
4.1.1. Introducción Tecnologías de uso....39
4.1.2. Esquema del proceso....40
4.1.3. Arquitectura....41
4.2. Descripción del sistema desarrollado / Implementación....43
4.2.1. Entorno de desarrollo....43
4.2.2. Repositorio Git y GitHub....43
4.2.3. Configuración inicial del servicio de CodeMagic....44
4.2.4. Entorno de producción y almacenamiento de artefactos del pipeline....46
4.2.5. Configuración del servicio de CodeMagic....49
4.3. Evaluación....60
4.3.1. Prueba práctica de pipeline de CodeMagic....60
4.3.2. Acceso remoto a máquina virtual de compilación para prueba manual de la aplicación en dispositivo iOS....66
5. Conclusiones y trabajo futuro....70
5.1. Conclusiones....70
5.2. Líneas de trabajo futuro....71
Referencias bibliográficas....73
Anexo A. Código fuente de pruebas unitarias....76
Anexo B. Código fuente de pruebas de integración....77
Anexo C. Código fuente de archivo main.dart....78

Índice de figuras

Figura 1. *Planificación del trabajo de fin de máster.* 11

Figura 2. *Proceso típico de desarrollo de software bajo el marco de trabajo ágil.* 15

Figura 3. *Documento extraído de 2020 DevOps Trends Survey by Atlassian & CITE Research.* 16

Figura 4. *Representación simple de la integración continua en la práctica.* 19

Figura 5. *Ejecución de pruebas y resolución de conflictos antes de la fusión.* 20

Figura 6. *Un típico proceso de entrega continua.* 21

Figura 7. *Representación del proceso de Despliegue Continuo.* 22

Figura 8. *Pipeline CI/CD de aplicaciones móviles automatizada.* 24

Figura 9. *Representación del flujo de trabajo de Scrum.* 36

Figura 10. *Estimación de product backlog según tipo de historia de usuario.* 37

Figura 11. *Arquitectura de la implementación del CI/CD.* 42

Figura 12. *Autorización de permisos de cuenta de GitHub para uso de CodeMagic.* 44

Figura 13. *Instalación y autorización de uso de repositorios para CodeMagic CI/CD.* 45

Figura 14. *Instalación, selección de repositorio y tipo de proyecto.* 45

Figura 15. *Página de inicio de sesión de AWS (Izq.) y Consola de administración de AWS (der.)* 46

Figura 16. *Formulario de creación de bucket S3.* 47

Figura 17. *Formulario de creación de distribución.* 48

Figura 18. *Nombre de dominio de distribución generado por defecto.* 49

Figura 19. *Distribución habilitada y lista para mostrar la aplicación web.* 49

Figura 20. *Página principal de aplicaciones para uso del pipeline.* 50

Figura 21. *Workflow Editor de aplicación.* 50

Figura 22. *Selección de plataforma de despliegue, y máquina de compilación.* 51

Figura 23. *Comprobación de Webhooks en CodeMagic y repositorio de GitHub.* 52

Figura 24. *Configuración de desencadenadores de compilación.* 53

Figura 25. *Habilitación de análisis estático de código.*.....54

Figura 26. *Habilitación de pruebas unitarias y de integración.*.....55

Figura 27. *Configuración del proceso de compilación.* 56

Figura 28. *Acceso a Credenciales de seguridad.* 57

Figura 29. *Claves de acceso de cuenta raíz de AWS.* 57

Figura 30. *Configuración de distribución a bucket S3 de AWS.* 58

Figura 31. *Alternativas en el caso de distribución a tiendas de aplicaciones.*.....58

Figura 32. *Configuración de notificaciones de estado de compilación y artefactos.* 59

Figura 33. *Estado inicial de la aplicación web desarrollada en Flutter.* 61

Figura 34. *Generando la ejecución del pipeline al hacer un push que provocará una pull request.* 62

Figura 35. *Pull request y comparación de código notificados para revisión.* 62

Figura 36. *Comparando y creando el pull request.*.....63

Figura 37. *El pull request genera la ejecución del pipeline.*.....63

Figura 38. *Construcción finalizada correctamente.*.....64

Figura 39. *Cambios y nuevas funcionalidades generadas por el pipeline automatizado.*.....65

Figura 40. *Correo electrónico de notificación de estado final de construcción.* 65

Figura 41. *Credenciales de ingreso a máquina virtual vía SSH o cliente VNC.* 66

Figura 42. *Configuración de conexión a máquina virtual.*.....67

Figura 43. *Pantalla al acceder a la máquina virtual vía VNC Viewer.* 68

Figura 44. *Ejecución de comandos para depurar el aplicativo en el simulador.* 69

Figura 45. *Depuración de aplicativo en simulador (iPhone 13 – iOS 15.4).*.....69

Índice de tablas

Tabla 1. *Comparación de tecnologías móviles multiplataforma.* 29

Tabla 2. *Comparación de herramientas CI/CD.* 32

1. Introducción

Existen varios procesos involucrados entre el editor de texto del desarrollador donde el código es escrito y la máquina del usuario final donde el software es ejecutado. El proceso de hacer que un software esté listo para el uso de los usuarios finales, testers y otras partes interesadas, se denomina implementación o entrega. Estos procesos forman parte del ciclo de vida de desarrollo de un sistema (SDLC). El ciclo de vida de desarrollo de software con sus siglas en inglés SDLC consiste en la descripción de un procedimiento para planear, diseñar, crear, testear y desplegar software o una aplicación que garantiza la calidad y precisión del software generado (GoodFirms, 2020). Varios escenarios en el SDLC exigen que se realicen tareas relevantes en esas etapas, puesto que administrar esos escenarios sin un estándar o una práctica de la industria a menudo es extremadamente difícil para la mayoría de los proyectos, en realidad existen diferentes prácticas para administrar con éxito el ciclo antes mencionado. Estas prácticas se denominan metodologías, y dos de las metodologías más utilizadas son en cascada y agiles. Este trabajo se centrará también en DevOps, que irá de la mano con otras metodologías ágiles, pues las metodologías de desarrollo de software han ido evolucionando constantemente ante las nuevas necesidades en el proceso de desarrollo, por ello marcos de trabajo como DevOps y metodologías agiles son importantes hoy en día (Domínguez Acosta, 2021). Cabe recalcar que implementar DevOps con métodos tradicionales es una tarea realmente difícil, puesto que uno de los principales objetivos de DevOps es ofrecer nuevas funciones de software con frecuencia, rapidez y calidad (Jabbari, bin Ali, Petersen, & Tanveer, 2016).

La práctica de integración y entrega continuas (y despliegue continuo) es perfectamente compatible con las metodologías ágiles en proyectos de desarrollo de software. Estas son prácticas que simplifican el proceso de ejecutar el desarrollo de tareas de ciclo de vida del software y se enfoca en el aseguramiento de la calidad y lanzamientos exitosos del producto. Estas prácticas de CI/CD introducen el concepto ágil DevOps de flujo de trabajo denominado pipeline de CI/CD, un conjunto de tareas bien definidas que se automatizan para garantizar la calidad del código.

El objetivo principal de adoptar pipelines de CI/CD en la escena industrial es integrar cambio en el código efectivamente, realizar pruebas sin esfuerzo, y reducir el tiempo de entrega del

software a producción, para que las nuevas características estén disponibles para los usuarios finales de manera más temprana. Para los desarrolladores de software, la adopción de estas prácticas, los libera de tareas innecesarias y repetitivas y les proporciona más tiempo para enforcarse en el desarrollo de nuevas funciones, y al mismo tiempo reduce el riesgo de introducción de futuros errores y desconfiguración de la infraestructura.

1.1. Justificación del trabajo

La importancia de este trabajo es mostrar las oportunidades de la automatización del ciclo CI/CD y las prácticas DevOps a desarrolladores de software y equipos pequeños de desarrollo. Las prácticas DevOps, pipelines CI/CD e implementaciones en la nube se han vuelto una tendencia en los últimos tiempos, pues no sólo las grandes organizaciones deben beneficiarse de su uso, sino también los desarrolladores autónomos y pequeñas organizaciones. Adoptar las prácticas y aprovechar la tecnología para la producción de software de alta calidad debe convertirse en una prioridad dentro de las empresas, para ello una vez que la mentalidad de DevOps se encuentre instaurada, configurar los pipelines y usarlas puede ser realmente simple y de fácil integración con el flujo de trabajo actual.

En el TFM analizaremos los antecedentes a nivel teórico y se mostrará una implementación de un pipeline CI/CD completamente automatizado, capaz de integrar, entregar e implementar un aplicativo móvil. Con el objetivo de demostrar que cualquier desarrollador de software o equipo pequeño puede implementar fácilmente dicha tecnología, utilizando una cantidad mínima de herramientas y configurándolas de una manera que no incurra en un costo adicional con una rápida puesta en marcha. Por ello, la intención detrás de la implementación del caso práctico es demostrar cuan fácil, rápido y rentable se puede crear e integrar un pipeline CI/CD en un flujo de trabajo habitual, sea este un desarrollador autónomo o equipo pequeño.

1.2. Planteamiento del problema

Muchas son las causas que han logrado que los frameworks de desarrollo de aplicaciones nativas multiplataforma (Android, iOS, Web) se popularicen tanto entre los desarrolladores, pues al contar con bajo presupuesto, tiempos de entrega casi inmediatos, equipos pequeños de desarrollo, han hecho que el uso de estas nuevas tecnologías de desarrollo de software

vayan en constante incremento, pues de cierta manera son una opción de desarrollo fácil y rápida, además de evitar un costo elevado en el desarrollo de aplicaciones Android, iOS o Web.

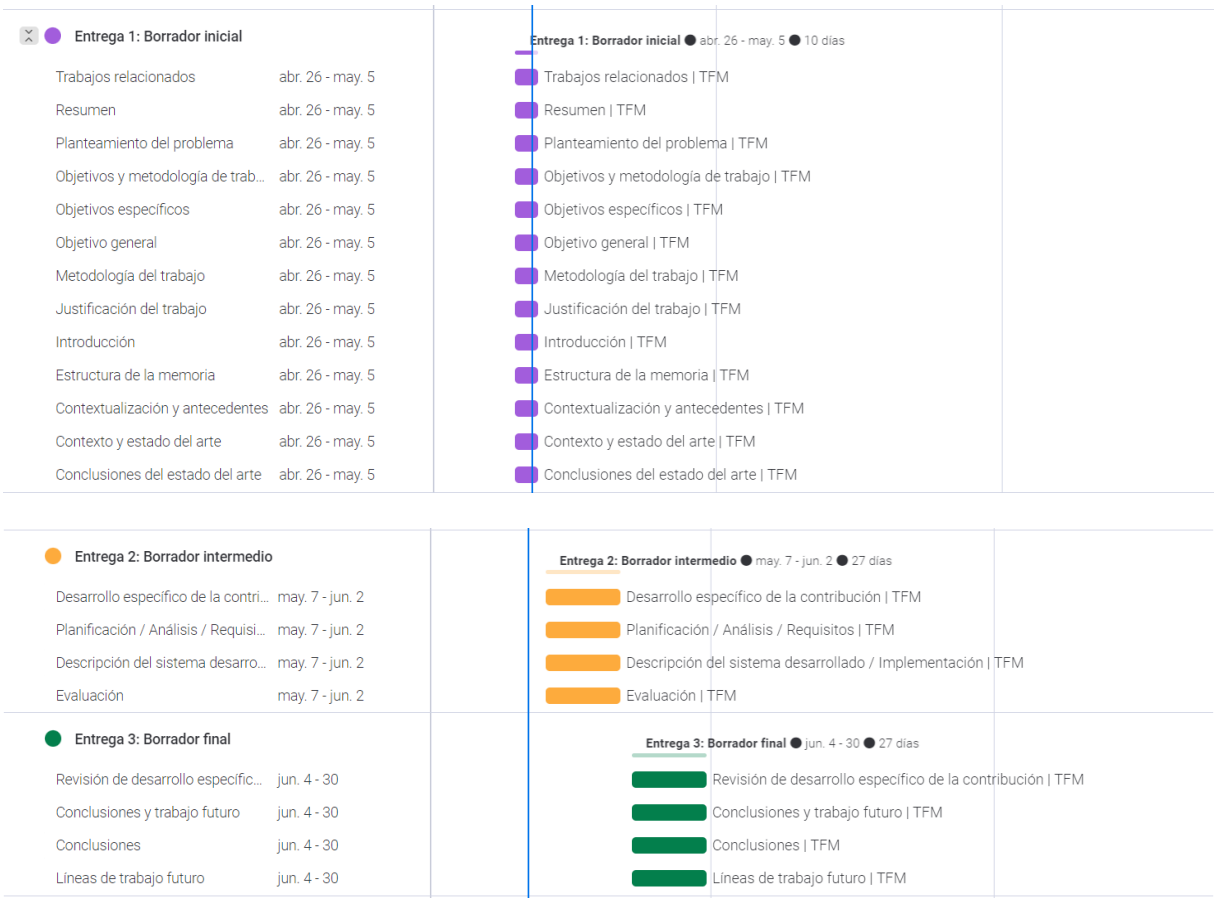
El desarrollo de aplicaciones nativas brinda una principal ventaja, la cual proporciona un mejor rendimiento de las aplicaciones en los dispositivos, además de brindar una mejor experiencia de usuario. Las aplicaciones desarrolladas nativamente cuentan con soporte completo del hardware y del sistema operativo lo que repercute en un alto nivel de rendimiento (Dharmwan, 2021). Fomentar el uso de un framework para desarrollar aplicaciones nativas multiplataforma, provee una solución con múltiples resultados, utilizando el mismo código para el despliegue de aplicaciones en plataformas como iOS o Android.

Para el despliegue de una solución nativa multiplataforma, se debe tomar en cuenta que esta debe compilarse desde entornos de desarrollo compatibles, en el caso de iOS su compilación debe correr sobre un sistema operativo MacOS, lo que conlleva a una serie de limitantes al estar desarrollando desde Windows o Linux.

Este trabajo pretende promover el uso de las mejores herramientas para crear un pipeline CI/CD que resuelva las problemáticas antes mencionadas, ya que automatizar la integración y el despliegue de la aplicación desde un ambiente centralizado solventaría este principal cuello de botella.

Con el fin de dar seguimiento al trabajo se ha construido una planificación.

Figura 1. *Planificación del trabajo de fin de máster.*



Fuente: Propia

1.3. Estructura de la memoria

Los antecedentes teóricos es lo primero que se encuentra establecido en la siguiente sección, seguida de otras secciones donde se investiga la conceptualización de la aplicación, objetivo, desarrollo de la aplicación, conclusión y trabajo futuro. La siguiente sección, Contexto y estado del arte, ofrece una breve presentación de la tecnología y sienta las bases para la implementación de la teoría que se presentará en las secciones siguientes.

El objetivo general, objetivos específicos y metodología de trabajo serán descritos en la sección 3. En el apartado cuarto se describe el paso a paso del desarrollo de la contribución, ejemplos de código fuente, diagramas de diseño, la utilidad del software, y la arquitectura del diseño del software, que lastimosamente se encuentra fuera del alcance de este trabajo. En el apartado quinto se desarrollarán las conclusiones del trabajo y las posibles mejoras a realizar a partir de lo expuesto en este TFM.

2. Contexto y estado del arte

Antes de embarcarnos en el ciclo de integración, y despliegue continuo, debemos familiarizarnos con la tecnología que rodea esta área. El presente capítulo proporcionará una breve introducción a los conceptos de DevOps, integración continua, despliegue continuo, pipeline de CI/CD, framework de desarrollo de aplicaciones nativas multiplataforma. La explicación a detalle del desarrollo de una aplicación nativa se encuentra más allá del alcance de este trabajo, por lo que se utilizará un enfoque pragmático.

2.1. Contextualización y antecedentes

Un conjunto de herramientas se deriva de los conceptos mencionados en el párrafo anterior. Con el objetivo de cumplir con los principios DevOps y las metodologías ágiles, a continuación, se listarán las herramientas que necesita el proyecto:

- **Herramienta de planificación de trabajo**, para que los desarrolladores e interesados puedan debatir sobre él y realizar un seguimiento.
- Un **administrador de control de versiones**, para mantener un único repositorio de código fuente y garantizar que cada commit se construya en el repositorio de integración.
- Un **servidor** para el flujo de trabajo de un pipeline de CI/CD para compilar, probar e implementar código, y llevar a cabo despliegues continuos.

El panorama de la tecnología de la información ha cambiado exponencialmente al introducir tendencias y tecnologías contemporáneas cada año. Sin embargo, todavía tenemos que escribir código, probar código, implantar el código y entregar el software. Esta tarea era muy diferente de cómo es hoy durante la era pre-ágil y DevOps del desarrollo de software.

2.1.1. Ciclo de vida de desarrollo de sistemas

Al igual que la construcción de objetos del mundo real, los sistemas de información se definen, planifican, diseñan, desarrollan, prueban, implementan y mantienen. Este proceso es indispensable en todo sistema de información y solo varía en las etapas internas y la diversidad de herramientas, tecnología y marcos que allí se aplican. Este proceso es comúnmente denominado como Ciclo de vida de desarrollo de sistemas.

En los escenarios más comunes del ciclo de vida del desarrollo de sistemas en las que se basan la mayoría de los proyectos son: análisis de requerimientos, diseño, desarrollo, pruebas, implementación y mantenimiento. En el marco ágil, estas etapas ocurren con frecuencia y se aplican de forma repetitiva en pequeños fragmentos de trabajo llamados Scrum y en pequeños períodos llamados Sprint en lugar de aplicar a todo el proyecto. DevOps generalmente usa esta forma ágil de trabajar, sin embargo, DevOps presenta muchos otros beneficios, tales como:

1. Incremento de la velocidad de entregas y despliegues en un ambiente de producción.
2. Disminución de errores en el ciclo de desarrollo a través de la automatización.
3. Mejorar el control del ciclo de desarrollo de un producto.
4. Mejor la confiabilidad del producto entregado.
5. Mejora la comunicación de las diferentes áreas involucradas. (Pragma, 2021)

Estos beneficios sin duda alteran positivamente el ciclo de vida del software y los cuales serán aplicados en la práctica de la integración y entrega continuas la cual es descrita más adelante.

2.1.2. Metodología ágil y prácticas DevOps

Las metodologías ágiles introducen una gran cantidad de prácticas y principios tales como la colaboración, el desarrollo incremental, pruebas repetitivas, flexibilidad, retroalimentación, enfoque en el cliente, etc. Estás prácticas promueven una organización más efectiva en el desarrollo de software, lo que al mismo tiempo provoca la necesidad de lanzamientos, pruebas e implementaciones más frecuentes.

Figura 2. *Proceso típico de desarrollo de software bajo el marco de trabajo ágil.*

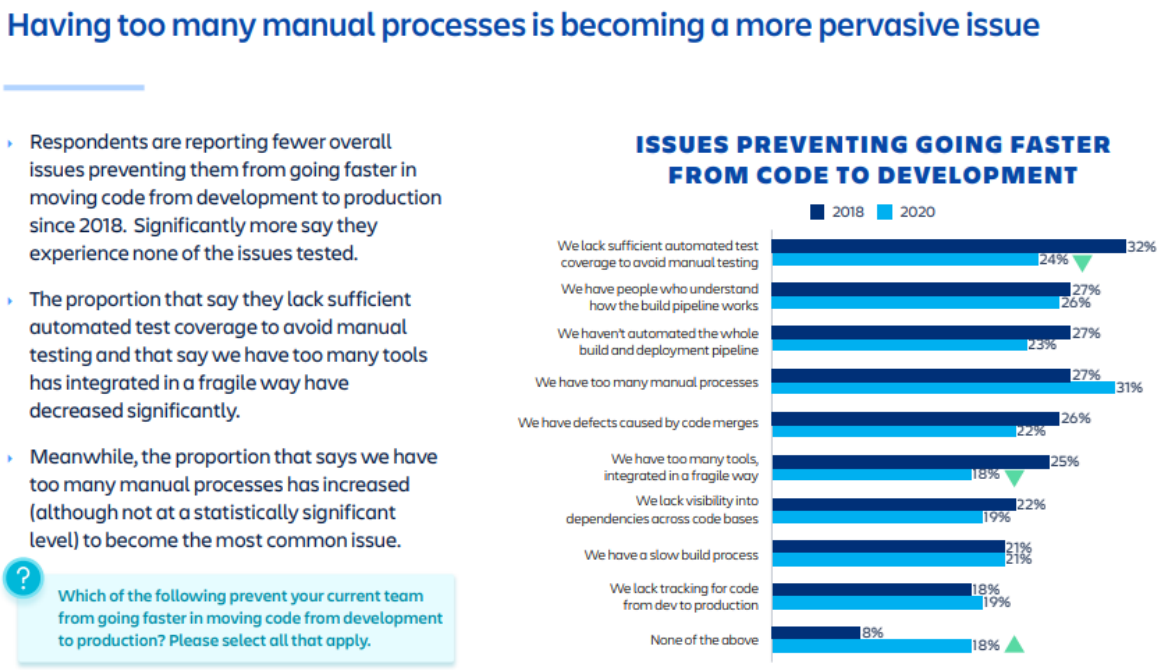
Fuente: Propia

La Figura 2 representa un marco de trabajo ágil aplicado a un proyecto de desarrollo de software con las seis etapas más comunes que se repiten en el ciclo. Los lanzamientos pueden ser diferentes versiones del software, nuevas funcionalidades, o corrección de errores. Es sólo una representación simplificada del proceso de desarrollo de software actual, en algunos casos puede incluir más etapas intermedias.

El cambio tecnológico y las nuevas formas de trabajo, tales como la adopción de prácticas DevOps, exigen la automatización de tareas repetitivas para una gestión de proyectos eficaz y eficiente. Por lo que cada vez más equipos han ido desarrollando y adoptando la automatización de pipelines de CI/CD.

Como se representa en la siguiente figura, uno de los problemas más comunes que impiden pasar más rápido del código a producción, es la no automatización del pipeline de compilación e implementación. También se evidencia que la adopción de la automatización de pipelines ha incrementado levemente desde el año 2018, disminuyendo en un 4% los problemas por no adopción de un pipeline automatizado.

Figura 3. *Documento extraído de 2020 DevOps Trends Survey by Atlassian & CITE Research.*



Fuente: Atlassian & CITE Research

Los pipelines de CI/CD están siendo cada vez más comunes en las empresas que desean automatizar el proceso de desarrollo de software. En una práctica muy común de DevOps, los pipelines son respaldadas por técnicos dedicados. El objetivo principal de DevOps es incorporar el proceso de desarrollo y operaciones de tal manera que se reduce las barreras entre estos dos equipos. Las prácticas relacionadas con DevOps, incluido el diseño y mantenimiento de pipelines de CI/CD, son habilidades importantes que se deben adquirir como desarrollador, ya que es posible que empresas pequeñas no tengan un administrador de sistemas dedicado o personal de DevOps capacitado para crear y respaldar los pipelines de integración continua y de implementación continua.

“La innovación inherente en DevOps es la conexión entre dos cohortes organizacionales, es decir, Desarrollo y Operaciones. La desconexión tradicional entre estas funciones condujo a silos dañinos que resultaron en grandes ineficiencias en la entrega de software. Sin embargo, conectar las funciones de Desarrollo y Operaciones ha llevado a la creación de nuevos patrones de colaboración entre roles que incluyen Desarrolladores, Arquitectos, Scrum Masters, Product Owners, Testers” (Aymeric, 2020)

Según la encuesta (The 2021 State of Database DevOps) realizada por RedGate Software en el año 2021, una empresa dedicada a brindar soluciones de base de datos ha documentado lo siguiente: Casi tres cuartas partes de las organizaciones, 74%, han adoptado DevOps de alguna forma en su enforque de desarrollo, comparado con un 47% del año 2020. Según los autores de la encuesta, ha surgido una clara correlación entre la adopción de DevOps y el rendimiento de la entrega de software, liberando cambios en aplicaciones más rápido, con más frecuencia y con menos errores.

2.1.3. Planificación y colaboración

La planificación y colaboración juegan un papel importante dentro del ciclo de desarrollo de software basado en DevOps y marcos de trabajo ágiles. Dentro del contexto existen varias herramientas que ayudan a los lideres y equipo de desarrolladores a planificar y dar seguimiento a sus tareas.

Una herramienta de gestión de proyectos es un software que sirve para la planificación, organización y gestión de un equipo involucrado en un trabajo, de manera que todos tengan claro las tareas que deben desarrollar, cuando realizar la entrega y quien debe ejecutarlas (Edix España, 2021).

Las herramientas de colaboración en línea permiten un flujo de trabajo más coordinado al proporcionar una plataforma nos permite interactuar en equipo ya sea para discutir sobre un tema, intercambiar archivos, interacción en tiempo real. Estas herramientas tienen que ver con sincronizar a las personas (The Digital Project Manager, 2022).

2.1.4. Implementación de Software

Varias etapas del ciclo de vida como, la construcción del código, control de versiones, integración continua, entrega continua, y despliegue continuo conforman un pipeline de CI/CD. El pipeline es usado para desplegar software en producción o pruebas. La integración continua es la primera etapa del pipeline el cual ayuda a los desarrolladores a integrar su código en el repositorio. La entrega continua por otra parte, como su nombre lo indica, es el proceso de asegurar el código para su despliegue a producción y el despliegue continuo se basa en automatizar el despliegue del código listo. Todo el pipeline puede ser activado dependiendo de un conjunto de reglas predefinidas y los desarrolladores pueden trabajar en el código y su integración, la entrega y el despliegue pueden suceder cuando sean requeridos.

2.1.5. Control de versiones con Git

EL control de versiones es una tecnología que permite rastrear y manejar los cambios del código, lo cual permite al desarrollador colaborar en un código base en común, suministrando detalles entorno a los cambios en el código y haciendo posible regresar a un estado anterior fácilmente.

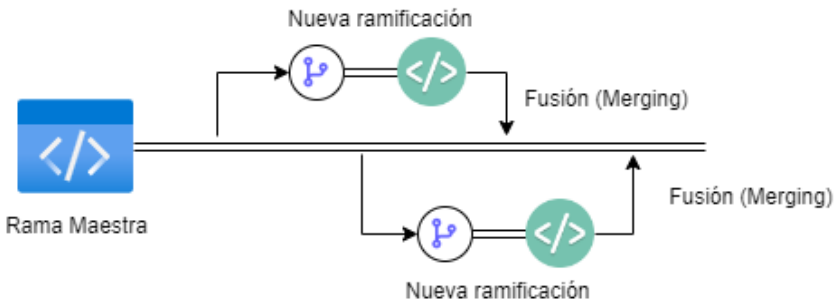
Según (Tsitoara) el control de versiones no es más que la administración de múltiples versiones de un proyecto. Para manejar una versión, cada cambio (incorporación, edición, o remoción) en los archivos de un proyecto deben ser rastreables. Cada cambio de una archivo o grupo de archivos registrado por el control de versiones ofrece una forma de deshacer o revertir cada cambio.

Hoy en día es realmente impensable para los desarrolladores no hacer uso de la tecnología de control de versiones. Todo equipo de desarrolladores trabaja con un sistema de control de versiones. Hay muchos tipos de sistemas de control de versiones, pero Git es uno de los más populares. Según una encuesta realizada a desarrolladores por StackOverflow, el 90% de desarrolladores usan Git como su sistema de control de versiones, lo que sugiere que es una herramienta fundamental para ser un desarrollador (Stack Overflow, 2021).

2.1.6. Integración continua

La integración continua es una práctica de una *continua integración* de código en un repositorio. Un repositorio de software es donde un todo el código que ha sido escrito por cada desarrollador es almacenado y mantenido. Un repositorio muy a menudo contiene muchos sub-repositorios temporales lo cuales son denominados ramas. Las ramas pueden ser formadas basadas en el desarrollador, donde cada desarrollador trabaja en una rama dedicada, o en características donde cada una de ella posee su propia rama, o acorde a la cultura y convención adoptada por el equipo de desarrollo. El repositorio principal de donde se realiza esta ramificación se denomina rama maestra. La práctica de integración continua quiere decir que las ramas son *fusionadas* con la rama maestra, esto puede ocurrir algunas veces al día.

Figura 4. *Representación simple de la integración continua en la práctica.*



Fuente: Propia

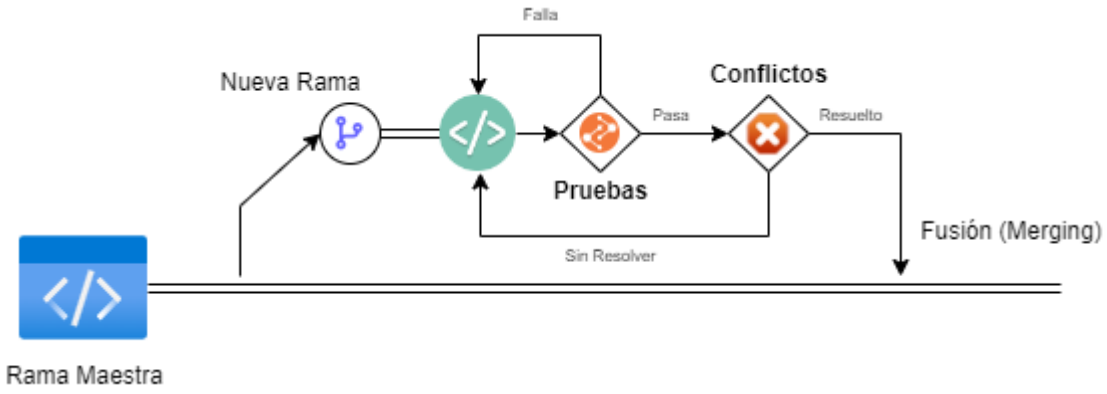
Cuando los desarrolladores realizan integración a menudo, o cuando la integración sucede frecuentemente, el costo de la integración es reducido. Un pequeño bloque de código es fácil de integrar, probar, y depurar que una base grande de código. Al hacerlo, lo conflictos de código, los comportamientos inesperados, los efectos secundario no deseados, etc. de la integración del código puede ser fácilmente detectada y resuelta de manera más temprana.

La integración continua termina con una versión de la aplicación (ej. un archivo binario, container, jar, etc.) llamado artefacto. La integración continua o CI puede automáticamente generar documentación o construir un instalador.

2.1.7. Integración continua como un proceso de desarrollo

En la Figura 4 se representa un típico proceso de integración continua con una forma simplificada de ejecución de pruebas, resolución de conflictos, etc. no son incluidas por esa misma simplicidad. La rama maestra suele ser también denominada rama principal. En un escenario típico práctico, el código es probado con pruebas definidas, comprobado contra conflictos con código existente en la rama maestra y pull requests se crean antes de fusionar con la rama maestra. La Figura 5 a continuación demuestra las medidas de control como ejecución de pruebas y resolución de conflictos antes de la fusión.

Figura 5. *Ejecución de pruebas y resolución de conflictos antes de la fusión.*



Fuente: Propia

Los equipos de desarrollo pueden integrar la revisión de código y verificar las medidas de control de calidad en el proceso de integración continua. Los detalles sobre las medidas de control de calidad, los tipos de pruebas y las prácticas en repositorios, etc. no están en el alcance de este trabajo.

En el libro (Continuous Integration: Improving Software Quality and Reducing Risk) Paul M. Duvall, Steve Matyas y Andrew Glover explican el proceso típico de integración continua como:

1. *Un desarrollador envía y confirma el código al repositorio de control de versiones. Mientras el servidor de CI en la maquina donde se integra y construye, pregunta al repositorio por cambios (ejemplo, cada cierto tiempo).*
2. *El servidor de CI detecta que los cambios han ocurrido en el repositorio de control de versiones, entonces el servidor de CI obtiene la última copia del código desde el repositorio, y luego ejecuta un script de construcción el cual integra el software.*
3. *El servidor de CI genera retroalimentación por correo electrónico de los resultados de la construcción a miembros específicos del proyecto.*
4. *El servidor de CI continúa a la espera de cambios en el repositorio de control de versiones.*

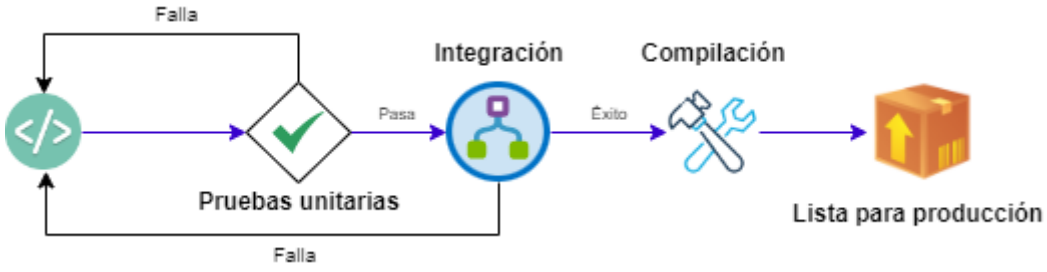
Por lo tanto, la integración continua inicia con la iniciativa de los desarrolladores y es fácilmente integrado con su flujo de trabajo existente. La adopción de la integración continua no solo hace desarrolladores más eficientes, también incrementa la calidad del código y ayuda en la gestión general del proyecto.

2.1.8. Entrega continua

La integración continua se extiende mediante el proceso de entrega continua el cual incluye la automatización del proceso de entrega del software. Esto quiere decir que el software está listo para el despliegue en producción a cualquier momento. Algunas compañías prefieren un despliegue manual de software en producción, sin embargo, el próximo escenario en el pipeline es el despliegue continuo el cual habilita desplegar el software automáticamente si decidimos hacerlo. La entrega continua beneficia a los desarrolladores reduciendo el riesgo de fallos al momento del despliegue, conseguimos agilizar la retroalimentación por parte del cliente lo que consecuentemente incrementa la calidad en el software.

Según (Red Hat, 2018) la entrega continua se refiere a que los cambios que implementa un desarrollador en una aplicación se someten a pruebas automáticas de errores y se cargan en un repositorio (como GitHub) para que luego el equipo de operaciones pueda implementarlos en un entorno de producción con el mínimo esfuerzo.

Figura 6. *Un típico proceso de entrega continua.*



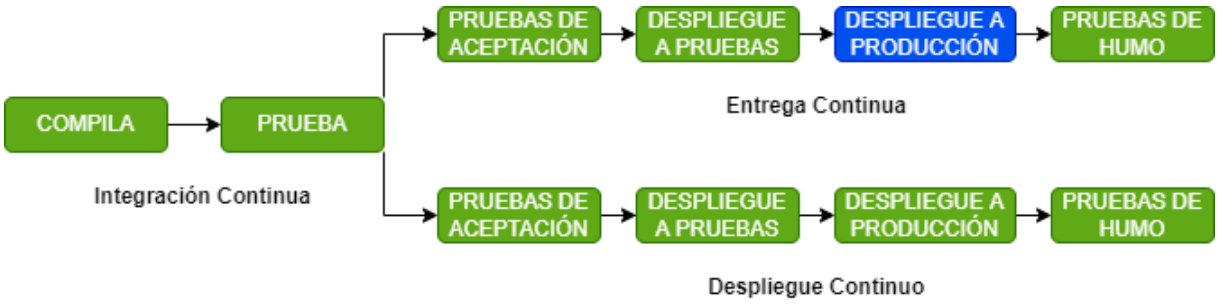
Fuente: Propia

El proceso de **Entrega Continua** replica un entorno idéntico al de producción y compila el software. Como la integración continua, esto sucede en pequeños intervalos repetidamente a todo momento. Esto permite a los desarrolladores de software estar siempre seguros de que su código funcionará en el entorno de producción. La entrega continua comúnmente contiene un conjunto de pruebas que son ejecutadas antes de los procesos de construcción y despliegue. La Figura 6 representa este proceso.

2.1.9. Despliegue continuo

El despliegue continuo es el proceso de desplegar automáticamente a un ambiente de producción cuando el software es compilado exitosamente por el proceso de entrega continua y todas las pruebas aprobadas. Ciertas organizaciones pueden decidir no usar el despliegue continuo porque prefieren desplegar manualmente basándose en su planificación. Sin embargo, hay un gran beneficio en adoptar el proceso de despliegue continuo ya que permite a las organizaciones llegar a los usuarios, reparar errores, lanzar nuevas versiones más rápido.

Figura 7. *Representación del proceso de Despliegue Continuo.*



Fuente: Propia

2.1.10. Entrega continua vs Despliegue continuo

La Figura 7 describe el proceso de **Despliegue Continúo** siendo activado después de una compilación exitosa desde el escenario de **Entrega Continua** del pipeline. El despliegue continuo, si es implementado, también sucede repetidamente en pequeños intervalos. En un escenario típico de despliegue, el proceso de despliegue continuo es activado tan pronto el proceso de entrega continua se completa, y el software está listo para el despliegue. Este proceso se encarga de mover el código al ambiente de producción, comúnmente suele ser a un servidor con infraestructura en la nube en organizaciones más modernas, haciendo que el nuevo software se encuentre disponible para los usuarios.

Una diferencia muy común entre la entrega y el despliegue continuos, casi siempre se trata de cómo el software se abre camino en producción. Esto significa que no todos los pasos del pipeline están automatizados y algunos especialmente el despliegue a producción es manejado manualmente. Dependiendo de la situación esto puede dar más control a los desarrolladores, pero al mismo tiempo agregar cierta sobrecarga e interrupción al proceso completo.

A menudo la entrega y despliegue continuos se confunden a pesar de ser dos procesos separados. A veces, estos dos procesos se combinan en uno solo como parte de un pipeline automatizado y la distinción podría no existir. Sin embargo, ambos procesos son importantes y comúnmente completan el pipeline CI/CD.

2.1.11. Pipeline de CI/CD automatizado

Un pipeline de CI/CD automatizado se forma mediante la creación un solo sistema el cual combina la Integración Continua, Entrega Continua y Despliegue Continuo en uno y es completamente automatizado, pues de esta manera no se requiere de mucha intervención humana cuando es ejecutado. Aprovechar la adopción de un pipeline automatizado puede reducir los costos y eliminar errores humanos. También puede proporcionar una importante retroalimentación sobre el código escrito, llegar rápidamente a los usuarios finales, y la oportunidad para que los desarrolladores se concentren en escribir un mejor software.

Según (Humble & Farley, 2010) argumentan que el despliegue manual de software es un anti-patrón y debe ser evitado. Señalan que esto puede dar lugar a muchos errores humanos, pero también reconocen que es muy común en todas las organizaciones. Además, continúan y enfatizan que la implementación del software debe ser un proceso automatizado, puesto que sólo debería haber dos tareas que deben ser ejecutadas por un humano para desplegar software en diferentes ambientes (desarrollo, pruebas o producción), seleccionar la versión, el ambiente y presionar el botón de “desplegar”.

2.1.12. Ventajas de la automatización de un pipeline de CI/CD

La adopción de un pipeline automatizado de CI/CD cambia el proceso de desarrollo, pruebas y despliegue completamente. El flujo de trabajo es altamente optimizado, y los errores son significativamente reducidos, esto es posible porque el código debe ir a través de etapas definidas por en el pipeline sujetas a pruebas automatizadas las cuales aseguran que los errores son descubiertos y restaurados antes de que lleguen a producción.

A continuación, las principales ventajas de adoptar un pipeline:

- Desarrollo más eficiente
- Despliegues más frecuentes
- Pruebas rápidas y eficientes
- Menos errores

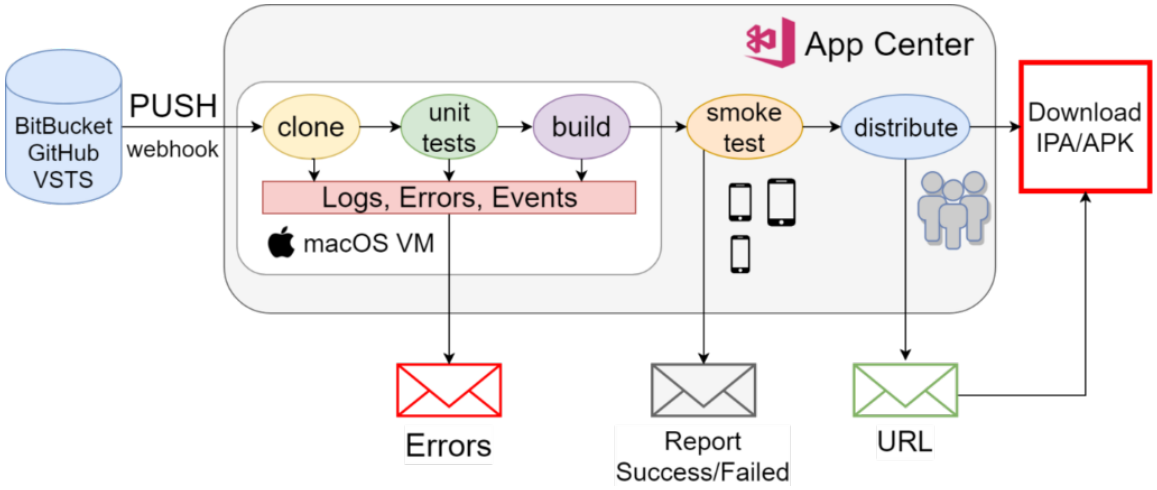
- Mejor experiencia de desarrollo
- Reducción de costos
- Procesos estandarizados
- Mejor calidad de software

2.1.13. CI/CD en aplicación móviles

Según (Niemand, s/f) el pipeline de integración continua y el despliegue continuo ayuda a los desarrolladores a entregar software más rápido gracias a la automatización de escenarios como compilación, pruebas y despliegue del ciclo de vida de software. Pues suele ser complicado hacer esto en el desarrollo de aplicaciones móviles. Esto ocurre debido al concepto de múltiples ambientes o entornos, por ejemplo, los ambientes de desarrollo, pruebas, producción, en realidad no existen. La gente encargada de pruebas normalmente solo posee un dispositivo físico, sin la habilidad de correr múltiples versiones de una aplicación junto a otra. Sin embargo, con nuevas herramientas ahora es posible.

Los kits de herramientas para crear y publicar paquetes de instalación existen desde hace mucho tiempo. Por lo general, son un conjunto de scripts que se usa en la máquina de compilación en el equipo de desarrollo móvil. La muestra el proceso general de un pipeline CI/CD de aplicaciones móviles. Recientemente los servicios en la nube han comenzado a ganar popularidad para la implementación de CI/CD, ahora es muy fácil optar por servicios como Visual Studio App Center, Code Magic, Bitrise.

Figura 8. *Pipeline CI/CD de aplicaciones móviles automatizada.*



Fuente: Mobile CI/CD 101 – App Center Blog

Las herramientas de CI/CD puede ser de dos tipos, alojadas en el mismo equipo o basadas en la nube.

Las soluciones CI/CD **alojadas en el mismo equipo** requiere poseer hardware, como por ejemplo máquinas con MacOS o servidores Linux. El proceso de configuración de los pipelines CI/CD puede tomar más tiempo y el mantenimiento de hardware y paquetes de software siempre es necesario para las herramientas CI/CD. Algunos ejemplos pueden ser Jenkins, TeamCity, BuildKite, etc.

Las herramientas CI/CD **basadas en la nube** son fáciles de configurar y todo el hardware y paquetes de software son gestionados en la nube, lo que requiere menos mantenimiento y de cierta manera los desarrolladores pueden enfocarse en seguir construyendo software. Algunos ejemplos son Bitrise, Travis, Circle, etc.

2.1.14. Criterios para evaluar herramientas CI/CD para aplicaciones móviles

Para la evaluación de herramientas CI/CD para el desarrollo móvil, existen algunos criterios claves importantes a tomar en cuenta.

Plataformas móviles soportadas y ambientes

La herramienta CI/CD idealmente tendría que soportar diferentes tipos de aplicaciones a compilar como iOS, Android y más si es posible. También buscar por un soporte robusto para múltiples ambientes o entornos, sistemas operativos, e incluso configuración de máquinas virtuales (Windows, Mac y Linux) si es necesario compilar otro tipo de aplicaciones no necesariamente móviles.

Integración

Una buena relación de integraciones puede realmente elevar el pipeline. Se debe mirar por soporte para integraciones con proveedores de repositorios de control de versiones, plataformas de distribución, herramientas de gestión de proyectos, servicios de análisis y monitoreo. La integración con herramientas de comunicación puede mejorar la visibilidad entorno al pipeline CI/CD: por ejemplo, en Slack se podría notificar al equipo sobre el estado de la compilación entre otros cambios de estado.

Diferenciar características

Algunas características CI/CD no son esenciales, pero pueden mejorar la experiencia de desarrollo. La habilidad de correr múltiples compilaciones simultáneamente, opciones de planificación para pruebas y compilaciones, pruebas de interfaz de usuario mediante el uso de simulación de hardware, y automatización de despliegue a varias tiendas de aplicaciones. Dependiendo de cuanto se desea mejorar el proceso de automatización del pipeline CI/CD para aplicaciones móviles, algunas de las alternativas antes mencionadas pueden ser grandes ahorradores de tiempo.

Usabilidad

La instalación, configuración y uso de la herramienta debe ser simple e intuitiva. Hay que recordar que estas herramientas serán usadas por equipos diversos y con diferentes roles, pues significaría una curva de aprendizaje elevada y podría decepcionar a los miembros del equipo y partes interesadas. Eso complicaría la interacción con la herramienta y la ayuda en su mantenimiento.

Soporte

Siempre es bueno saber el nivel de soporte que ofrece la compañía. Los representantes de soporte expertos, guías de implementación, los tutoriales detallados, y la extensa documentación son buenos indicadores de un programa de soporte de calidad.

Costo

Muchas de las herramientas ofrecen una capa gratuita, a más aplicaciones generadas y llevadas a producción más costo que abra que solventar. Muchas herramientas basan su precio en el número de compilaciones y usuarios, en el caso de equipos pequeños y pocas aplicaciones suele ser más que suficiente una capa gratuita. Sea cual sea la elección el precio se basa en el uso, para esto se debe tomar en cuanto y evaluar la necesidad de volumen de uso del equipo.

2.1.15. Aplicaciones Móviles

El uso de aplicaciones móviles ha tenido un incremento cada vez más popular en todos los aspectos de la vida, tales como trabajo, entretenimiento, educación y salud (Alrumayh, Lehman, & Tan, 2021).

Lo que nos hace pensar que el desarrollo de estas es cada vez más importante dentro de cualquier industria, por lo que la elección del enfoque de desarrollo adecuado será sustancial si se desea construir una aplicación funcional y eficiente.

Aplicaciones Móviles Nativas

El desarrollo de aplicaciones nativas se centra en la creación de aplicaciones para Android, iOS, o Windows. Ciertos aspectos y funcionalidades hacen que el desarrollo de aplicaciones móviles nativas sea diferente al de otros enfoques como es el caso de aplicaciones web o híbridas. A continuación, se menciona algunas de sus características.

- Desarrollo específico de cada plataforma.
- Descargas en tiendas oficiales.
- Mayor estabilidad.
- Mayor rendimiento.
- Diseño UX superior.
- Uso de características y funcionalidades exclusivas del dispositivo (Cámara, GPS, micrófono, etc.)

Aplicaciones Web

Son un tipo de aplicaciones basadas en la web las cuales no hacen uso de los recursos del dispositivo, pues es una aplicación que se almacena en un servidor remoto y se entrega a través de la interfaz del navegador (TechTarget, 2019). Las aplicaciones web tienen usos diferentes, y potenciales beneficios tales como:

- Permite el acceso de múltiples usuarios a una misma versión del aplicativo.
- No necesitan ser instaladas.
- Puede ser accesible desde múltiples plataformas tales como computadores de escritorio, laptops o teléfonos móviles.
- Acceso a través de múltiples navegadores.

Aplicaciones Móviles Hibridas

Son aplicaciones móviles diseñadas en un lenguaje de programación web, ya sea HTML5, CSS o JavaScript, junto con un framework que permite adaptar la vista web a cualquier vista de diferentes dispositivos móviles (Bellido, 2021). A continuación, se menciona las características de este tipo de aplicaciones.

- Desarrollo unificado.
- Desarrollo más rápido.
- Rendimiento mucho más lento.
- Diseño UX inferior.

Aplicaciones Nativas Multiplataforma

El propósito del desarrollo de aplicaciones nativas multiplataforma es solventar los principales errores que las aplicaciones hibridas basadas en web presentan, además de optimizar la relación costo/beneficio, lo cual se logra a través de una misma base de código que al crearse bajo un mismo lenguaje de programación, facilita su exportación e implementación en las diferentes plataformas.

La creación de aplicaciones móviles nativas multiplataforma se puede realizar con herramientas de rendering nativo. Herramientas como Flutter o React Native son frameworks que generan código nativo para cada sistema operativo. Esto hace que la experiencia de usuario sea igual que una aplicación nativa (Fernández, 2021).

Claro está que las aplicaciones nativas multiplataforma presentan un sin número de beneficios con respecto a las completamente nativas, sin embargo, en ocasiones se ven condicionadas con ciertas limitantes con respecto al uso de los recursos del dispositivo de manera directa, que pueden ser solventados con el uso de librerías de terceros, pero su rendimiento se verá limitado.

2.2. Trabajos relacionados

El objetivo primordial en la construcción de aplicaciones móviles es tratar de llegar a la mayor cantidad de plataformas, por lo que se considera usar una tecnología multiplataforma que sea de ayuda para obtener ese resultado. Por lo que se ha considerado algunas soluciones modernas que son mencionadas a continuación.

Ionic

Es un kit de desarrollo de software (SDK) de código abierto para la creación de aplicaciones móviles y de escritorio de alta calidad y rendimiento que utiliza tecnologías web (HTML, CSS y JavaScript) con integraciones para marcos populares como Angular, React y Vue (Ionicframework.com, s.f.). Se ejecuta como una mezcla de

código nativo y web, una aplicación de Ionic, es, por lo tanto, una página web ejecutándose como una capa de aplicación nativa.

React Native

Es un framework de código abierto para el desarrollo de aplicaciones móviles creado por Facebook, combina las mejores partes desarrollo nativo con React, una librería JavaScript para crear interfaces de usuario (Reactnative.dev, s.f.). React Native es famoso por ofrecer aplicaciones de Facebook, Instagram y Skype para plataformas iOS y Android.

Flutter

Es un framework de código abierto creado por Google para la construcción de hermosas aplicaciones, nativamente compiladas, y multiplataforma desde un único código base (Flutter.dev, s.f.). Es una tecnología relativamente nueva a comparación de las dos antes mencionadas, gracias a eso, es la solución más innovadora.

A continuación, se presenta un cuadro comparativo de las tecnologías antes descritas.

Tabla 1. *Comparación de tecnologías móviles multiplataforma.*

Categoría	Ionic	React Native	Flutter	Desarrollo Nativo
Reusabilidad de Código	10 – Sólo una aplicación	5 – Estilo personalizado requerido	7 – Muchos controles por defecto	1 – Cada plataforma necesita su estilo dedicado
Lenguaje / Curva de aprendizaje	7 – Código JavaScript compartido / problemas de plataforma bien conocidos con plugins	8 - Código JavaScript compartido / plataforma conocida	5 – Lenguaje poco conocido (Dart)	1 – Dos plataformas, dos lenguajes y marcos de trabajo distintos.
UI / Biblioteca de componentes	7 – La mayor parte de componentes de interfaz de usuario son proveídos.	3 – Requiere ajustes de estilo en muchos componentes	8 – La mayor parte de componentes de interfaz de usuario son proveídos.	10 – Nativo, todo está ahí

Ecosistema / Bibliotecas de terceros	8 – Muy bueno, todas las dependencias están disponibles en la plataforma web	7 – Muy bueno, todas las bibliotecas principales están disponibles	7 – Bueno, todas las bibliotecas principales están disponibles	10 – Completo, todos los plugins necesarios están ahí
Popularidad	5 – Usualmente considerada con código base existente, y no muy popular como una plataforma móvil dedicada	7 – La solución multiplataforma más popular	7 – La plataforma más joven no tan popular aún, pero está subiendo rápidamente	10 – El enfoque más popular
Rendimiento	4 – Promedio, especialmente con un diseño complejo	7 – Bueno, cercano a lo nativo con pequeños problemas con componentes personalizados	9 – Mu alto, casi nativo a 60fps.	10 - Nativo
Funciones nativas de dispositivo	3 – Faltan algunas dependencias nativas o requieren plugins comerciales	7 – La mayor parte de plugins están disponibles	6 - La mayor parte de plugins están disponibles	10 - Todas
Uso en el mundo real	5 – No muy popular en aplicaciones móviles dedicadas.	8 – Muy popular (Facebook, Instagram, Airbnb)	7- Ganando recientemente fuerte popularidad	10 – Muy popular
Proceso de desarrollo	5 – Bueno, desarrollo rápido	8 – Muy bueno, iteración de desarrollo rápido	9 – Muy bueno, iteración de desarrollo rápido	7 - Bueno
Desarrollo de interfaz de usuario personalizada (Animaciones)	5 – Bajo control sobre apariencia de componentes de interfaz de usuario personalizados	6 – Toma mucho trabajo crear animaciones enriquecidas	10 – Animaciones enriquecidas sin restricción de apariencia de interfaz de usuario	5 – Crear animaciones interesantes no es difícil, el problema es cuando Android y iOS deben verse igual
Legado	10 – Sin legado	10 – Sin legado	10 – Sin legado	1 – Mucho legado
Disponibilidad de equipo de desarrollo	4 – Un pequeño número de desarrolladores	8 – Muchos freelancers y pocas compañías	7 - Muchos freelancers y pocas compañías	10 – La mayor parte de compañías que

			ofrecen móvil	desarrollo
SUMATORIA	**73**	**84**	**92**	**87**

Fuente: Propia, traducción de https://allbright.io/blog/ionic-reactnative-flutter-comparison

Al final de la Tabla 1 se encuentran marcada una sumatoria de todos los puntos, en cada categoría pudiendo obtenerse diez puntos para un total de ciento veinte puntos, en el cual Flutter ha obtenido una calificación de 92 puntos, seguido de Desarrollo Nativo con 87 puntos. Los puntos fuertes según la categoría en las que es superior Flutter son en el desarrollo de la interfaz de usuario, iteración de desarrollo rápido, reusabilidad de código y muy buen rendimiento comparándose casi con uno nativo.

Flutter al ser una nueva tecnología y al contar con un lenguaje de programación no tan popular llamado Dart podría implicar cierta incertidumbre en la curva de aprendizaje del desarrollador, pero al ser un lenguaje similar a Java y JavaScript, se convierte en un lenguaje simple, moderno y altamente eficiente para trabajar con él, el aprendizaje en si es usualmente muy intuitivo. Además, se prevé que la comunidad de Flutter ganará más popularidad en los próximos años y debido a esta rápida expansión y crecimiento de esta tecnología la comunidad probablemente deje atrás a React Native.

La selección de Flutter como tecnología de desarrollo de aplicaciones multiplataforma en este trabajo es un aporte directo hacia la comunidad que este representa, además, Flutter al requerir de menos tiempo y esfuerzo para construir una aplicación es considerado ideal como producto mínimo viable encajando con la metodología de trabajo expuesta en el siguiente capítulo.

En el mercado existen varias herramientas DevOps para la gestión y automatización de un pipeline CI/CD para el desarrollo, integración, entrega y despliegue de aplicaciones móviles nativas. Una de las características a tomarse en cuenta será el soporte con el framework de desarrollo de software que se ha comparado en párrafos anteriores.

Bitrise

Es una plataforma como servicio (PaaS) de integración y entrega continua (CI/CD) con un enfoque principal en el desarrollo de aplicaciones móviles (iOS, Android, React Native, Flutter, etc.). En si es una colección de herramientas y servicios que ayudan con

el desarrollo y automatización de proyectos de software (Welcome to Bitrise documentation!, 2022).

Jenkins

Es un servidor líder de automatización de código abierto, el cual provee cientos de plugins para soportar compilaciones, despliegues y automatización de cualquier proyecto. Al ser un servidor de automatización extensible, Jenkins puede usarse como servidor de CI simple o convertirse en el centro de entrega continua de cualquier proyecto (Jenkins, 2022).

CodeMagic

Es la plataforma CI/CD para desarrollo móvil más rápido que existe con flujos de trabajo personalizables, la cual permite conectar e integrar herramientas y servicio para automatizar el pipeline. Con CodeMagic se puede configurar un entorno CI/CD para aplicaciones nativas Android o iOS, Flutter, ReactNative, Cordova y Ionic (Codemagic - CI/CD for android, iOS, flutter and react native projects, 2022).

A continuación, se presenta un cuadro comparativo de las tecnologías antes descritas.

Tabla 2. *Comparación de herramientas CI/CD.*

Categoría	Bitrise	Jenkins	CodeMagic
Planes	9 - Capa gratuita con 300 créditos al mes.	10 - Servidor de código abierto	9 - Capa gratuita con 500 minutos por mes de construcción.
Tipos de cómputo para compilación	8 - Standard VMs en capa gratuita	6 - Depende del servidor y agentes donde serán desplegados	8 - macOS standard VMs en capa gratuita
Número de construcciones de pipeline simultaneas	8 - 5 en capa gratuita	10 - Ilimitadas	6 - 1 en capa gratuita
Compatibilidad con Flutter	7 - Si	5 - Si, pero la construcción a distintas plataformas se ve limitada por el S.O. del agente de despliegue.	10 - Si, CodeMagic en sus inicios comenzó como una plataforma de integración

			continúa especializada en Flutter.
Permite la creación de pipelines complejas	8 - Si	7 - Si	6 - Si, pero con algunas limitaciones
Integración directa con Google Play y App Store Connect	6 - Si, la configuración tiende a ser más compleja	5 - Si, la configuración tiende a ser más compleja	10 - Si, fácil integración
Dificultad para crear un pipeline básico	8 – Relativamente sencillo	5 – Necesita la configuración de muchos plugins	10 – Rápido y sencillo
SUMATORIA	**54**	**48**	**59**

Fuente: Propia

Al final de la Tabla 2 se encuentran marcada una sumatoria de todos los puntos, en cada categoría pudiendo obtenerse diez puntos para un total de setenta puntos, en el cual CodeMagic ha obtenido una calificación de 59 puntos, seguido de Bitrise con 54 puntos. Los puntos fuertes según la categoría en las que es superior CodeMagic son en la compatibilidad que tiene con el framework de desarrollo seleccionado, la integración directa y de fácil configuración para la distribución simultánea en Google Play y App Store Connect, y la dificultad para montar un pipeline básico es relativamente rápido y sencillo que es una de las cualidades que busca el objetivo de este trabajo.

Jenkins posee varias desventajas ante estas dos plataformas puesto que, para empezar a construir el pipeline, Jenkins debe ser instalado, configurado y mantenido, lo que se convierte en tiempo que podríamos utilizar para montar el pipeline CI/CD, que a comparación de CodeMagic y Bitrise con el solo hecho de iniciar sesión ya estarían listas para empezar con la configuración del pipeline. Otra de las desventajas parte de una de las problemáticas establecidas en capítulos iniciales de este trabajo, ya que, al momento de la compilación, si se desea compilar para Android y iOS, al menos se debería configurar el agente de Jenkins sobre MacOS, pues al tener un agente sobre Linux o Windows sólo se podría compilar para Android. Nuevamente CodeMagic y Bitrise tienen la ventaja al proveer máquinas virtuales, mismas que contienen el sistema operativo MacOS, lo que solucionaría el inconveniente de compilación y pruebas para Android y en especial iOS, sin la necesidad de invertir en un equipo Apple.

2.3. Conclusiones del estado del arte

Existe una gran cantidad de conceptos y términos asociados al proceso del ciclo de vida del desarrollo de software, metodologías ágiles, prácticas DevOps, integración continua, entrega continua, despliegue continuo, y desarrollo de aplicaciones móviles nativas multiplataforma, y nuevas tendencias tecnológicas que son parte de la investigación de este trabajo, que servirá como marco referencial para la elaboración del desarrollo específico de esta contribución.

El enfoque de CI/CD de aplicaciones móviles permite simplificar y acelerar el proceso de creación de un producto de calidad. El pipeline de CI/CD de aplicaciones móviles en la nube permite "configurar y olvidar" un entorno local para compilar, probar y distribuir software de forma manual. Estas nuevas tecnologías sirven de ayuda puesto que, como plataformas se encuentran listas para la configuración y ejecución de nuevos flujos de trabajo de pipelines CI/CD.

En este capítulo se ha explicado el proceso de desarrollo de software móvil y no móvil, y las ventajas de optar por la automatización de un pipeline CI/CD.

3. Objetivos y metodología de trabajo

A continuación, se describen el objetivo general, objetivos específicos y la metodología de trabajo que se implementará en este proyecto.

3.1. Objetivo general

EL objetivo del trabajo consiste en optimizar la integración, entrega y despliegue en el desarrollo de una aplicación móvil multiplataforma con marcos de trabajo o kit de desarrollo de software de código abierto mediante la automatización de cada uno de sus procesos asociados.

3.2. Objetivos específicos

- Reducir costes y tiempos en el desarrollo de un aplicativo móvil nativo.
- Conocer las diferentes herramientas para producir aplicaciones móviles nativas.
- Indicar la correcta aplicación y flujo de trabajo de un pipeline de CI/CD.
- Comparar el uso de herramientas DevOps en el despliegue y testeo de aplicaciones móviles nativas.
- Desarrollar una aplicación móvil nativa enmarcada en las prácticas DevOps determinando el mejor marco de trabajo ágil.

3.3. Metodología del trabajo

Para el desarrollo y ejecución del proyecto se utilizará prácticas DevOps de la mano de metodologías ágiles. Se determinará un marco de trabajo que cumpla con las expectativas del proceso de implementación del aplicativo móvil nativo tomando las mejores prácticas del marco de trabajo Scrum, y haciendo uso de un tablero Kanban para una mejor organización y seguimiento de las tareas que se llevaran a cabo, de esta manera obtendremos lo mejor de ambos. Cabe recalcar que ninguna de estas herramientas es completa o perfecta, pues no mencionan todo lo que se debe hacer, sólo proporcionan las directrices necesarias para manejar nuestro proyecto.

Por una parte, SCRUM introduce los siguientes roles dentro de este proyecto:

- **Product Owner (Dueño del producto):** es el responsable del financiamiento del proyecto durante su ciclo de vida y pone en marcha los requerimientos y objetivos del proyecto. (Sverrisdottir, Ingason, & Jonasson, 2014)
- **Scrum Master:** es el encargado de velar por el cumplimiento y correcta ejecución de la metodología y de ser el líder del equipo. (Sverrisdottir, Ingason, & Jonasson, 2014)
- **Development Team (Equipo de desarrollo):** está conformado por el equipo de desarrolladores, los que se encargará de generar los productos de software.

Una vez definidos los roles, Scrum parte de un primer conjunto o lista de requerimientos los cuales son denominados Product Backlog. La intención del Product Backlog será la de evolucionar durante toda la vida del proyecto/producto, adaptándose tanto a las nuevas necesidades, como a todo lo que el equipo Scrum y los interesados vayan aprendiendo Sprint tras Sprint. (Herranz, 2016)

Figura 9. *Representación del flujo de trabajo de Scrum.*

Fuente: Propia

Para este proyecto se define el siguiente flujo de trabajo basado en la Figura 9:

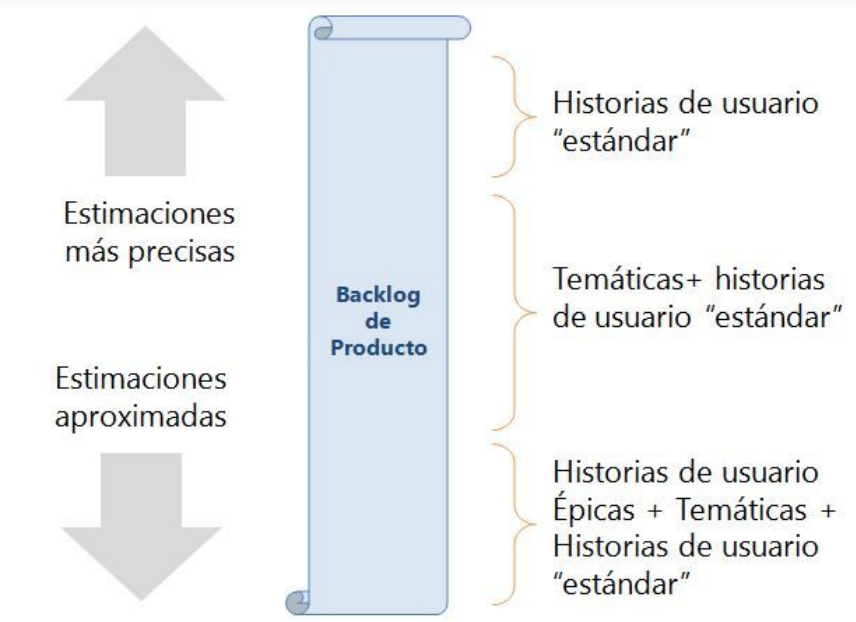
- Duración de 1 semana para cada sprint.
- Reunión diaria de 15 minutos antes de iniciar actividades.
- Reunión de planificación de Sprint de 4 horas.
- Reunión de retrospectiva de Sprint de 4 horas.

- Revisión de Sprint de 1 hora.

Cabe recalcar que el flujo de trabajo antes mencionado es frecuentemente reflejado en un equipo multidisciplinar, en este trabajo se adaptará a un marco de trabajo de un desarrollador autónomo.

Para la recopilación de requisitos se hará uso de historias de usuario, pues el paradigma ágil asocia la historia de usuario a requisito funcional. Pueden existir diferentes tipos de historias de usuario, mientras más historias de usuario estándar se recopilen en el product backlog se conseguirá una estimación de proyecto más aproximada, tal y como muestra la Figura 10. Para la elaboración del product backlog se hace uso de historias de usuario épicas, y conforme vayamos organizando las funciones están van a descomponerse en historias de usuario estándar, las cuales serán manejadas como tareas en cada sprint y en función de su importancia.

Figura 10. *Estimación de product backlog según tipo de historia de usuario.*



Fuente: Proagilist.es

Por un lado, se utilizará Scrum como marco de trabajo ágil y también se hará énfasis en la metodología de Kanban, de las cuales podemos destacar sus principales características y las que se tomará en cuenta para este trabajo:

Scrum:

- Valores ágiles.

- Colaboración entre miembros del equipo.
- Autoorganización, y autogestión de los equipos.
- Product backlog adaptativo.
- Iteraciones de duración fija.
- Retroalimentación continua.

Kanban:

- Tablero
- Adaptabilidad ante cambios.
- Flujo de trabajo limitado por el trabajo en curso.
- Principio “Menos es más”.
- Limitación de tareas por cada columna del tablero.
- Tablero con columnas personalizables.

4. Desarrollo específico de la contribución

Este trabajo implementa el marco de trabajo teórico establecido en los anteriores capítulos.

4.1. Planificación / Análisis / Requisitos

La implementación contiene un ejemplo de aplicativo móvil, un pipeline CI/CD totalmente automatizado, una instancia con ambientes pruebas y producción replicado en la nube. Al ser este trabajo enfocado en una aplicación móvil, la mayoría de las herramientas utilizadas son basadas en la nube a excepción del editor de código, y el ambiente local de desarrollo para la escritura del código. Existe una gran variedad de herramientas y tecnología, las que son nombradas a continuación han sido seleccionadas considerando la facilidad de integración con las herramientas existentes para desarrolladores, los costos, la dificultad de instalación y configuración:

- Flutter SDK
- Ambiente local:
  - Computador con Windows 10.
  - Editor de código Visual Studio Code.
  - Git como sistema de control de versiones.
- Repositorio de software en la nube
  - GitHub (URL: https://github.com)
- CI/CD Pipeline
  - CodeMagic (URL: https://codemagic.io)
- Simulación de pruebas y producción
  - CodeMagic Static Pages, Amazon S3.

4.1.1. Introducción Tecnologías de uso.

Flutter, es una herramienta de desarrollo de aplicaciones móviles de código abierto creado por Google. En esta implementación, Flutter es usado para desarrollar interfaces de usuario para aplicaciones en Android, iOS, y Web. Esta implementación puede ser modificada para ser utilizada por cualquier otro framework y lenguaje de programación. Cabe mencionar que la implementación es completamente independiente al sistema operativo que se utilice, pues Windows 10 se usa únicamente por la disponibilidad y familiaridad.

Visual Studio Code, es una herramienta gratuita y de código abierto, que nos ayudará como editor de código fuente, control integrado de Git y que además presenta como gran utilidad la descarga y gestión de extensiones con las que se puede personalizar y potenciar la herramienta.

Git será utilizado como sistema de control de versiones en esta implementación. Git es un software gratuito, de código abierto y uno de los sistemas más populares de control de versiones. Existen otro tipo de sistemas de control de versiones, sin embargo, no son muy utilizadas en estos días, tales como Subversion (SVN), Mercurial.

GitHub, es una plataforma en la nube utilizada como repositorio y almacén de proyectos utilizando el sistema de control de versiones Git. Además, GitHub es reconocida como la plataforma más importante de colaboración para proyectos de código abierto. La implementación está basada en GitHub, por su fácil integración con el pipeline CI/CD de CodeMagic.

CodeMagic, es un servicio basado en la nube que presenta herramientas de integración y entrega continuas para la construcción de aplicaciones móviles. El servicio ofrece soporte para los frameworks más populares de desarrollo móvil, como Android, iOS, Flutter, ReactNative, Cordova, o Ionic. Este servicio proporciona una solución de pipeline completo, integración con el repositorio de código, herramientas de integración continua, herramientas de entrega, pruebas y análisis de código, publicación automática, máquina virtual MacOS para compilación.

AWS, es un servicio de nube que provee servicios empresariales en la nube, incluido instancias de servidores, almacenamiento, servicios de red, etc. Para el propósito de la implementación se hará uso de Amazon S3 para publicar y probar la aplicación, debido a que Flutter posee la característica de crear una solución web con base al mismo código que se utiliza para generar las aplicaciones para Android y iOS.

Otro tipo de herramientas y tecnología usada será explicado según aparezca en el contexto a continuación.

4.1.2. Esquema del proceso

Se optó por los siguientes pasos con el objetivo de crear los componentes necesarios de esta implementación:

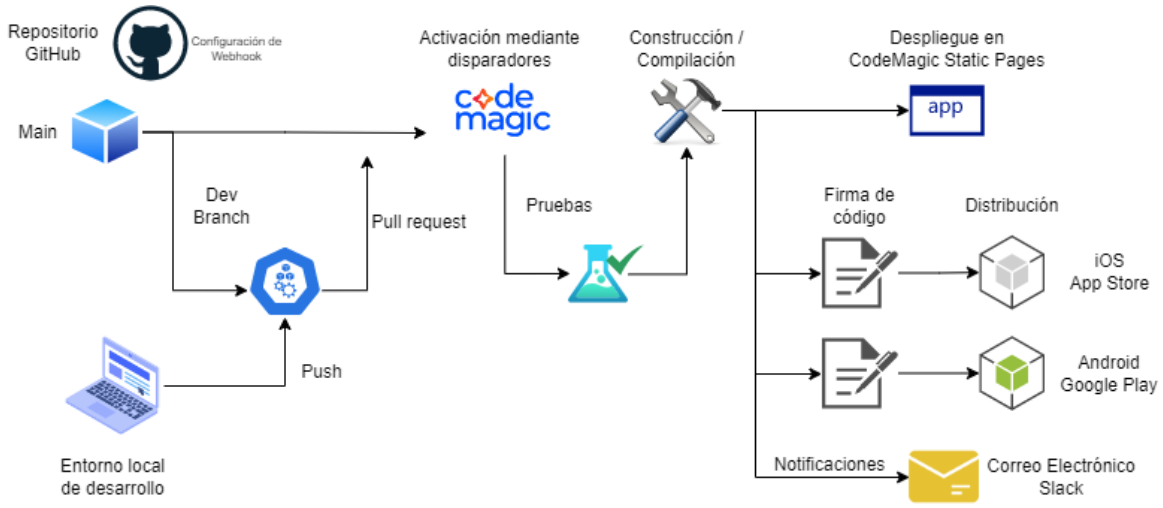
- Creación de un proyecto simple de Flutter en el computador local.
- Prueba y selección de un repositorio de software en la nube para el proyecto: como selección GitHub.
- Creación de una versión inicial del pipeline CI/CD conectando el código local, el repositorio y CodeMagic.
- Selección de un proveedor de servicio de nube y almacenamiento para el despliegue de la aplicación web: Amazon S3 y CloudFront de la nube de AWS.
- Escritura de pruebas unitarias, widget, y de integración.
- Configuración de máquina virtual de compilación para Web, iOS y Android en CodeMagic.
- Configuración necesaria del servicio de nube para el despliegue del pipeline. Amazon S3, Amazon CloudFront y CodeMagic.
- Configuración de firma de código en CodeMagic.
- Configuración de publicaciones y distribución automática a tiendas de aplicaciones (AppStore y Google Play) y Amazon S3 en CodeMagic.
- Configuración de envío de notificaciones.
- Usando el pipeline para correr pruebas automáticas en la nube de CodeMagic.
- Prueba de la aplicación como usuario final.

La siguiente arquitectura es elaborada implementando estos pasos, cabe recalcar que los detalles de la implementación se explican en las siguientes secciones.

4.1.3. Arquitectura

El siguiente diagrama ilustra la arquitectura del pipeline en este trabajo:

Figura 11. *Arquitectura de la implementación del CI/CD.*



Fuente: Propia

Tal y como muestra la Figura 11, el punto de entrada del pipeline es el entorno local de desarrollo, el cual está conectado al repositorio de GitHub. El repositorio contiene una rama **main** y otras ramas. La rama de desarrollo será creada con el objetivo de realizar los cambios necesarios en el código que posteriormente generará una pull request luego de efectuar un push a la rama main.

La construcción y configuración de los activadores del pipeline de CodeMagic funcionaran al momento de que un pull request sea aprobado, para lo cual se debe también configurar los webhooks de conexión de GitHub con las APIs de CodeMagic. Cuando el pipeline se active, un conjunto de pruebas correrá y las pruebas deberían culminar satisfactoriamente. Si se pasan las pruebas se iniciará el proceso de compilación o construcción, para también correr una prueba de integración según sea el caso, una vez culminado iniciará el proceso de firma de código si se ha optado por construir una aplicación móvil para proceder con la distribución del nuevo lanzamiento en las tiendas de aplicaciones, o en el caso de optar por una construcción web se efectuará un despliegue sobre la misma nube de CodeMagic publicándolo sobre su apartado de static pages. Al finalizar el proceso, el pipeline ofrece un procedimiento de notificaciones ya sea por correo electrónico o notificaciones en Slack al equipo de trabajo.

En el proceso de la construcción o compilación podrá usarse distintos modos, el CD con CodeMagic nos garantizará modos de depuración, lanzamiento o de perfil. En el caso de ejecutarse pruebas de integración se podrá especificar qué tipo de dispositivo utilizar, entre los cuales se puede optar por emulador, simulador o un dispositivo físico (AWS Device Farm).

4.2. Descripción del sistema desarrollado / Implementación

Como se describe en el bosquejo del proceso anterior, la implementación se lleva a cabo de la siguiente manera:

4.2.1. Entorno de desarrollo

No existe una regla estricta o guías sobre como un entorno de desarrollo debe ser configurado, pues existen diversas posibilidades. En este trabajo, la simpleza y la facilidad será una prioridad antes que establecer o seguir convenciones.

Primero se crea un directorio denominado CICD, el directorio CICD será donde se almacenará el proyecto de Flutter y para su generación habrá que descargar y configurar el Flutter SDK https://storage.googleapis.com/flutter_infra_release/releases/stable/windows/flutter_windows_3.0.2-stable.zip que a la fecha en la que se realiza este trabajo se encuentra en la versión 3.0.2.

Una vez configurada la variable de entorno para el uso del SDK de Flutter y demás propiedades, haremos uso de la línea de comandos sobre la cual ejecutaremos la siguiente línea ***$ flutter create flutter_app***, el cual creará un nuevo proyecto denominado ***flutter_app***, que incluye el soporte web además del soporte móvil.

Finalmente, ya generado el proyecto se inicializa el repositorio local de git, ejecutando el comando:

$ git init

Este comando inicializa el directorio completo bajo un sistema git de control de versiones.

4.2.2. Repositorio Git y GitHub

GitHub ha sido seleccionado como el repositorio git del proyecto, un nuevo proyecto llamado **flutter_app** ha sido creado en el entorno local de desarrollo para luego subirlo en el repositorio creado en GitHub. El proyecto puede ser nombrado de cualquier manera, no es necesario llamarlo flutter_app, pero es llevar las cosas con simpleza y orden.

Para subir el primer código base a GitHub será necesario crear un nuevo repositorio en la web de GitHub y obtener la URL del repositorio de GitHub (mostrada como $GITHUB_URL_REPOSITORIO más adelante) para establecer la conexión con el depósito local de git, usando el siguiente comando:

$ git remote add origin $GITHUB_URL_REPOSITORIO

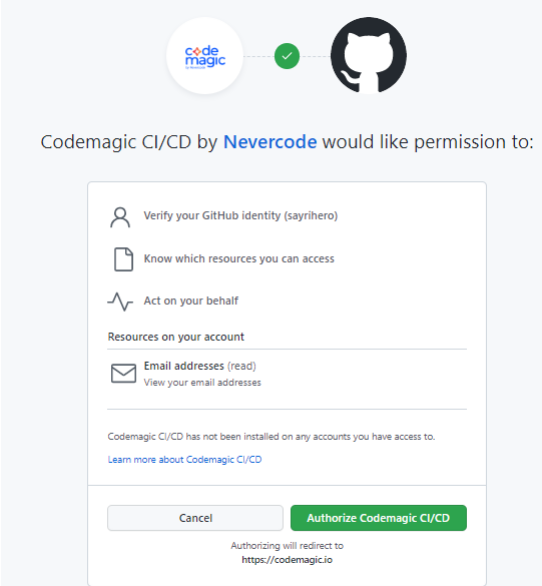
Finalmente, el contenido local del proyecto **flutter_app** es enviado al repositorio remoto de GitHub mediante un push realizando los siguientes comandos de git:

- Establecer nuestra rama como main con ***$ git Branch -M main***
- Añadir todos los archivos al rastreador de git con ***$ git add .***
- Realizando el respectivo commit con ***$ git commit -m "Proyecto base"***
- Subir los archivos mediante ***$ git push -u origin main***

4.2.3. Configuración inicial del servicio de CodeMagic

El servicio en la nube de CodeMagic debe ser configurado con el objetivo de automatizar el pipeline CI/CD del aplicativo en desarrollo, aprovechando todas y cada una de las características que este servicio ofrece en la nube. Una de las grandes ventajas de CodeMagic es la integración directa con GitHub, puesto que para iniciar sesión bastará con otorgar los permisos necesarios de una cuenta de GitHub, tal y como la Figura 12.

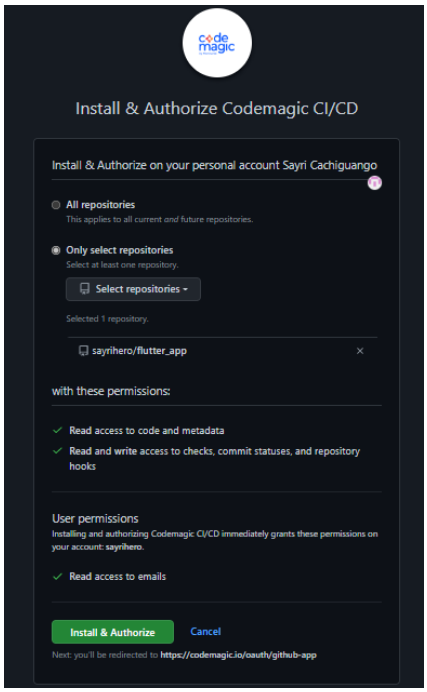
Figura 12. *Autorización de permisos de cuenta de GitHub para uso de CodeMagic.*



Fuente: Propia, GitHub.com

Ya en la interfaz principal de CodeMagic, el primer movimiento a realizar será adicionar en CodeMagic el proyecto que ya cuenta con un repositorio en GitHub denominado ***flutter_app***, para lo cual conectaremos el repositorio de GitHub con CodeMagic, instalando y autorizando el repositorio para su uso en CodeMagic CI/CD, como se ve a continuación en la Figura 13.

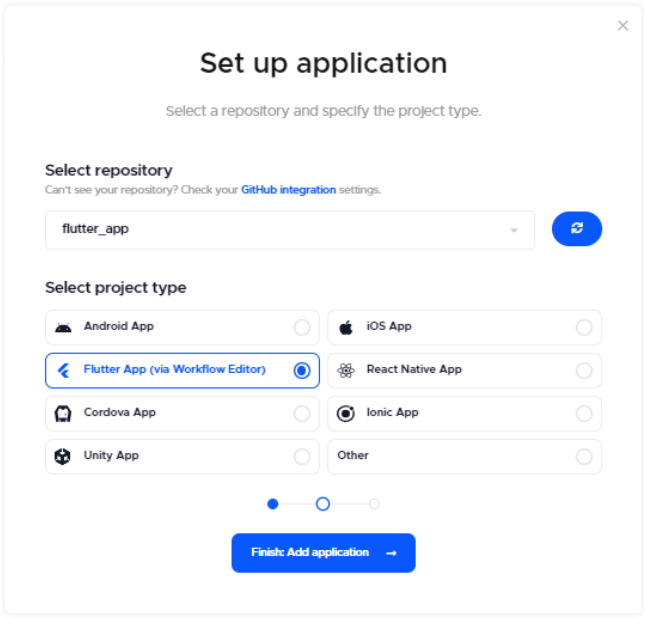
Figura 13. *Instalación y autorización de uso de repositorios para CodeMagic CI/CD.*



Fuente: Propia, GitHub.com

Para configurar el proyecto se debe seleccionar el repositorio que ha autorizado y el tipo de proyecto, tal y como se muestra en Figura 14, que en este caso es una aplicación de Flutter. De esta manera la configuración inicial del pipeline quedará generado, y listo para su configuración.

Figura 14. *Instalación, selección de repositorio y tipo de proyecto.*



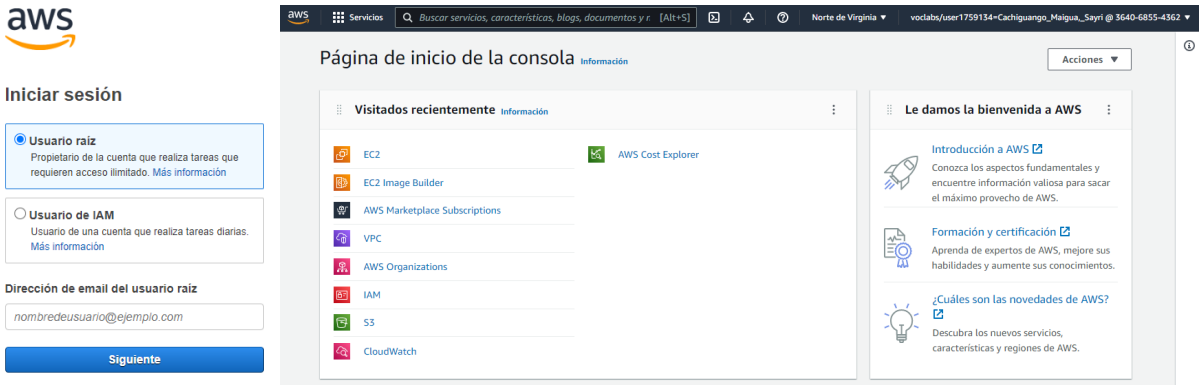
Fuente: Propia, CodeMagic.io

4.2.4. Entorno de producción y almacenamiento de artefactos del pipeline

El entorno de producción probablemente sea una pieza fundamental del trabajo. Los pasos anteriores se encuentran más o menos relacionados con el conocimiento de los desarrolladores incluso si nunca han utilizado un pipeline CI/CD, ya que la tecnología de control de versiones y la configuración de diferentes entornos son prácticas comunes dentro del desarrollo de software.

La configuración inicia desde la activación de una instancia de almacenamiento S3 en la nube de AWS (Amazon Web Services). Para aquello una cuenta de AWS debe ser creada e iniciada sesión en la consola de AWS usando el usuario raíz y la contraseña correspondiente. Una consola de AWS o AWS Management Console es la interfaz de usuario que permite activar el uso de diferentes servicios ofertados por la nube de AWS.

Figura 15. *Página de inicio de sesión de AWS (Izq.) y Consola de administración de AWS (der.)*



Fuente: Propia, aws.amazon.com

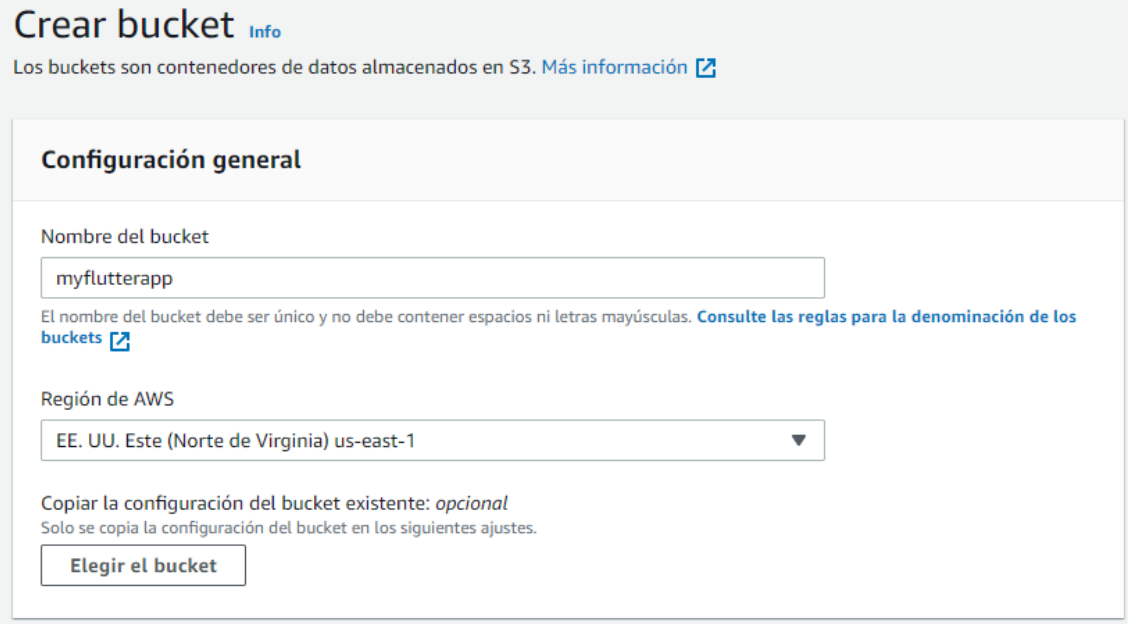
Algunos recursos de AWS se crean para simular diferentes ambientes, tal es el caso del servicio de almacenamiento Amazon S3 para el hospedaje de la aplicación de página única (fluter_app) en un bucket (contenedor de datos almacenados en S3) como ambiente de producción, y además otro bucket que será usado como almacenamiento de los artefactos generados tras la ejecución del pipeline.

4.2.4.1. Creación de Almacenamiento Amazon S3 para sitio web

Se empezará a configurar el primer almacenamiento de Amazon S3, con el propósito de ser utilizado como hospedaje de la aplicación de página única de producción que será desplegada desde Code Magic. Durante la creación del almacenamiento S3 es muy importante crearlo en la región recomendada del área en la que se encuentre.

Para crear un bucket se debe dirigir al dashboard de administración de S3, y en la sección de **Buckets**, crear el bucket, con las siguientes características:

Figura 16. *Formulario de creación de bucket S3.*



Fuente: Propia, aws.amazon.com.

Como se muestra en la Figura 16, el formulario de creación del bucket solicita un nombre, la región de AWS que para esta implementación se ha seleccionado el área de EE.UU. Este (Norte de Virginia) us-east-1, y los demás valores irán por defecto.

4.2.4.2. Configuración de Amazon CloudFront

La habilitación del servicio de red de Amazon CloudFront se lo hace con el objetivo de entregar el contenido, en este caso la aplicación web, mediante una CDN. Una de las ventajas de utilizar CloudFront es que cachea y funciona como servidor de contenido estático y dinámico de un origen que para este caso en particular es el bucket de S3.

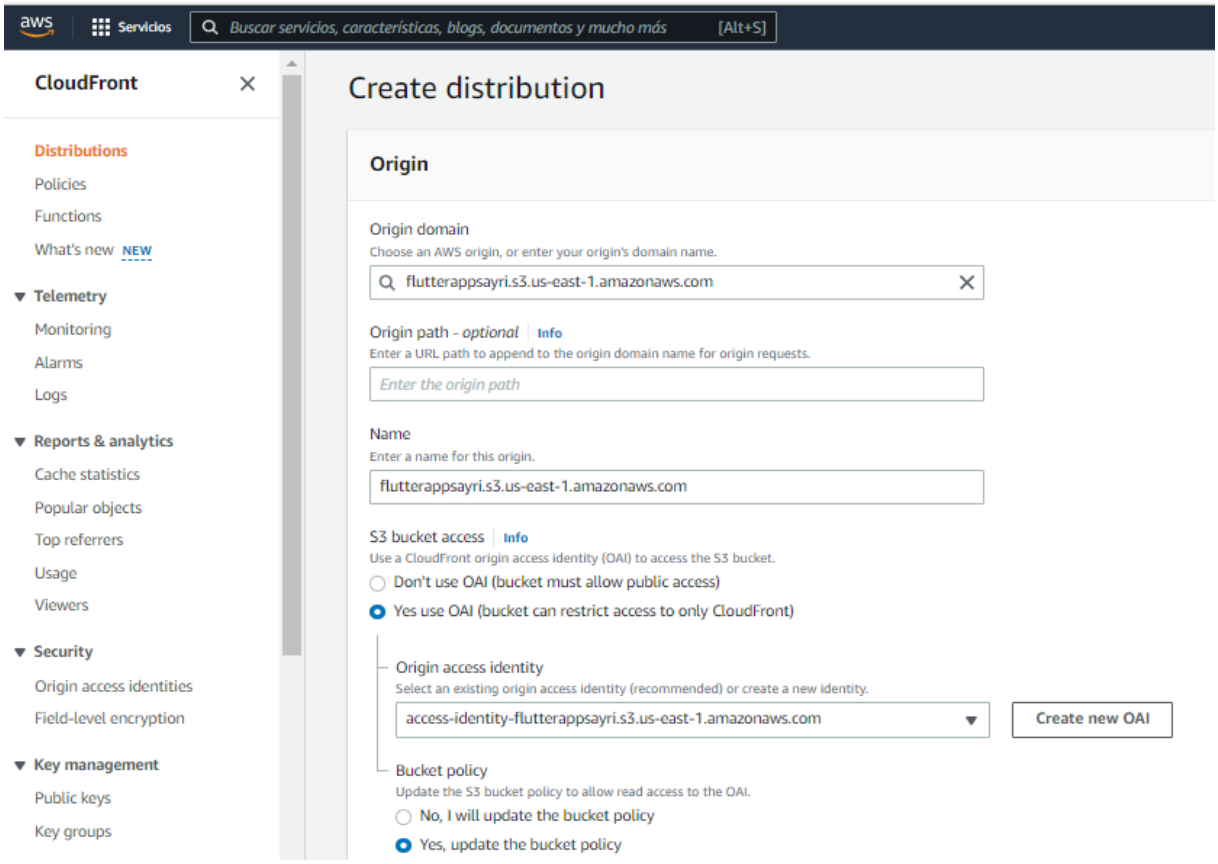
Para configurar se debe dirigir al dashboard de administración de **CloudFront**, y en la sección de **Distribuciones** se creará una nueva con las siguientes características:

- Primero se selecciona el origen, se utilizará el bucket S3 creado en la sección anterior.
- Se configurará el acceso al bucket S3 mediante **OAI (Origin Access Identity)**, esta configuración permite a CloudFront tener acceso al bucket para ser el canal de presentación de los objetos de este, la aplicación web. Además, se crea un nuevo OAI y se habilita la opción de actualizar la política del bucket, lo que garantiza los permisos

necesarios para la perfecta comunicación entre CloudFront y el bucket S3. Tal y como se muestra en la Figura 17.

- Para la **Política del protocolo de visor** se selecciona la opción **Redirección HTTP to HTTPS**.
- En la opción de **Default root object** se agrega el index.html, para que sea la página de inicio.

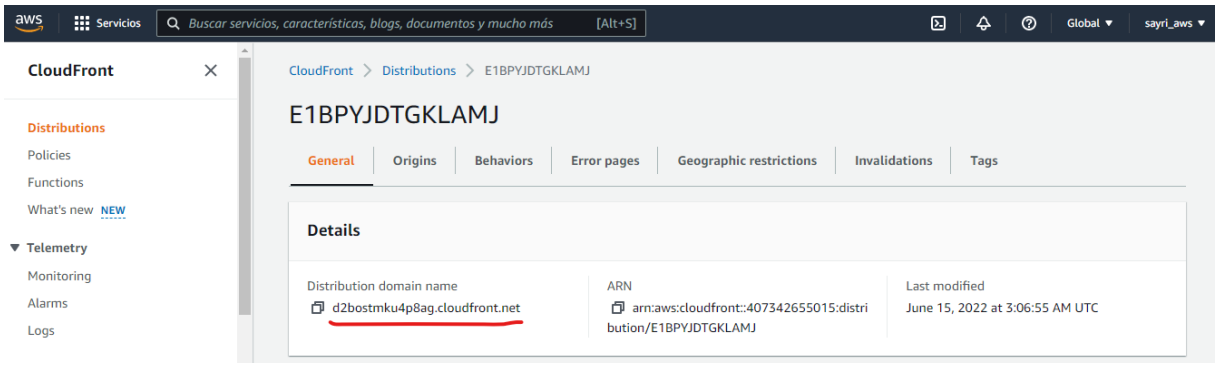
Figura 17. *Formulario de creación de distribución.*



Fuente: Propia, aws.amazon.com

Y con las demás opciones por defecto se procede a crear la distribución, la misma que tardará un par de minutos en desplegarse por completo. La distribución generará un nombre de dominio señalado en la Figura 18, el cual se utilizará para ingresar al aplicativo web en el ambiente de producción una vez que este haya sido desplegado desde CodeMagic.

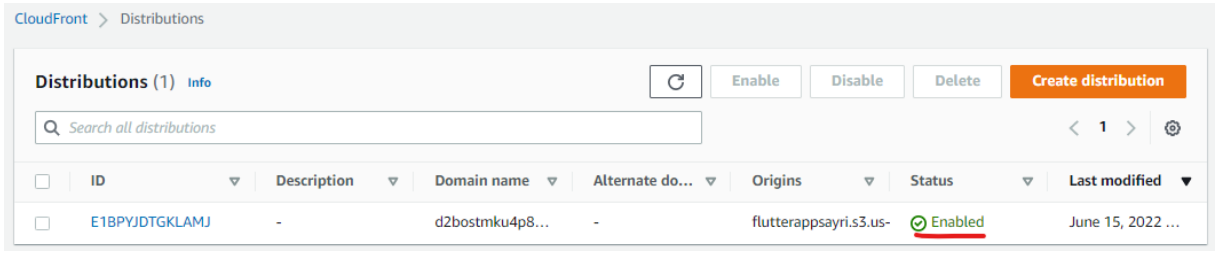
Figura 18. *Nombre de dominio de distribución generado por defecto.*



Fuente: Propia, aws.amazon.com

Hay que tomar en cuenta que la distribución debe estar habilitada tal y como se muestra en la Figura 19, esto es un indicador de que se puede acceder a la aplicación web.

Figura 19. *Distribución habilitada y lista para mostrar la aplicación web.*

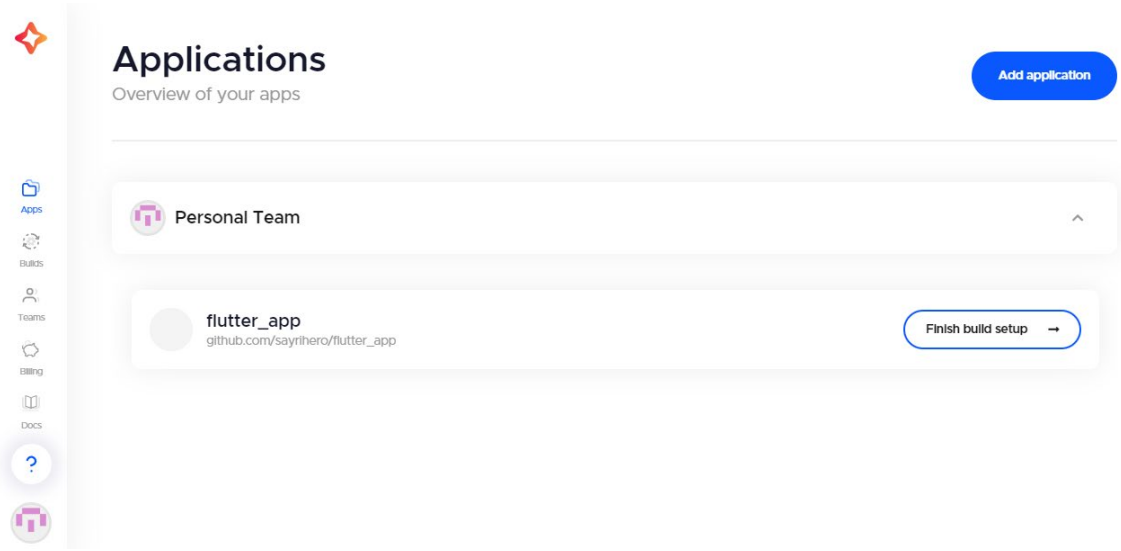


Fuente: Propia, aws.amazon.com

4.2.5. Configuración del servicio de CodeMagic

En el apartado 4.2.3. se generó la configuración inicial del pipeline en el servicio de CodeMagic, a partir de este momento continuaremos con la configuración del flujo de trabajo que manejará el pipeline.

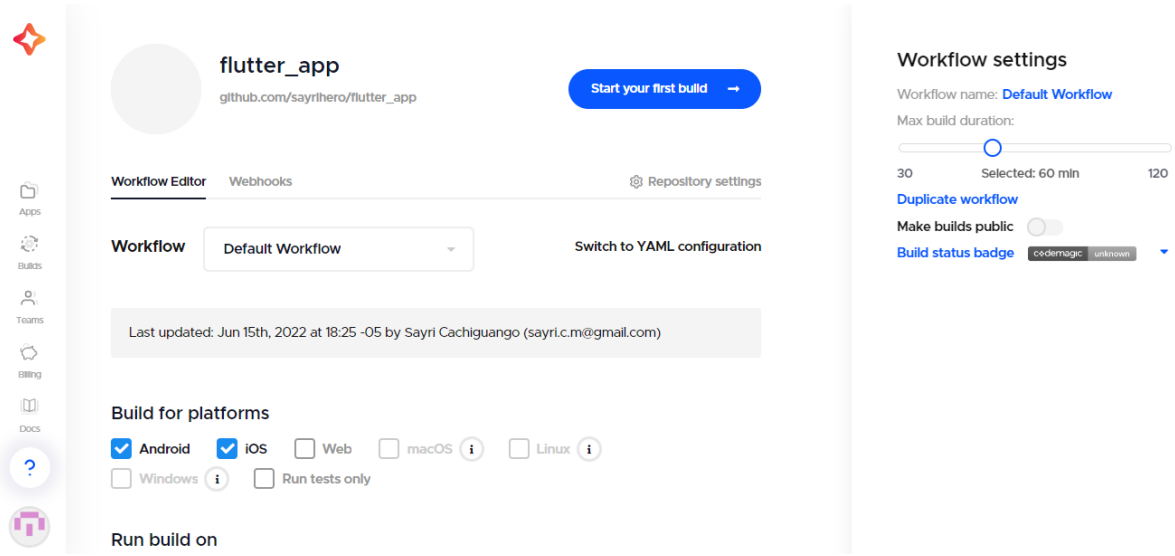
Figura 20. *Página principal de aplicaciones para uso del pipeline.*



Fuente: Propia, codemagic.io

Dentro de CodeMagic se ingresa al menú de **Apps** el cual se presenta tal y como se muestra en la Figura 20, y como se puede observar aún falta completar la configuración. Una vez dentro de la configuración del flujo de trabajo en el Workflow Editor (Figura 21) se procede a modificar cada una de las secciones que un pipeline CI/CD puede presentar (Integración, Pruebas, Compilación, Distribución).

Figura 21. *Workflow Editor de aplicación.*



Fuente: Propia, codemagic.io

4.2.5.1. Flujo de trabajo (Workflow)

CodeMagic brinda la posibilidad de crear múltiples flujos de trabajo para la construcción de un proyecto, es decir, el flujo completo del pipeline desde sus activadores hasta su publicación. Por ejemplo, en el caso de querer tener flujos de trabajo separados para desarrollo, pruebas, o producción de la aplicación, construcción de diferentes ramas del proyecto, construcciones separadas de depuración o lanzamiento.

Un nuevo flujo de trabajo puede ser creado duplicando uno existente. Se navega a la Barra Lateral Derecha > Workflow settings y clic en Duplicate workflow. Para el propósito de este trabajo se construirá un flujo que simule el pipeline de salida a producción, por lo que renombraremos el flujo de trabajo a **CI/CD Web Producción**.

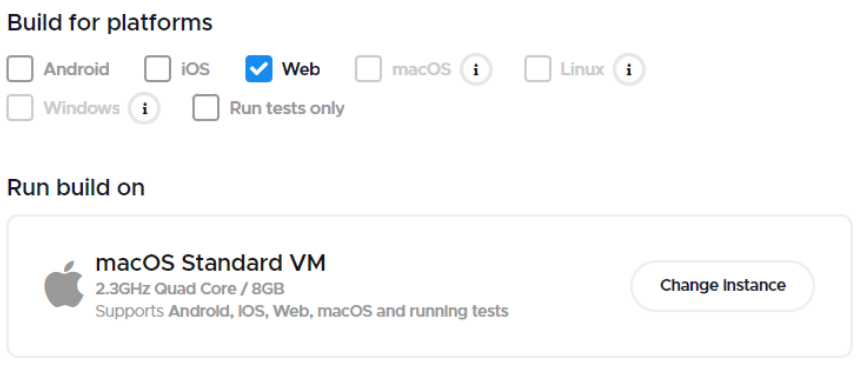
El flujo de trabajo puede ser configurado a través del **Workflow Editor** o haciendo uso de la configuración mediante un **Archivo YAML**, en este caso se usará Workflow Editor para una mejor experiencia de usuario.

4.2.5.2. Compilación para plataformas y Ejecutar compilación (Build for platforms & Run build on)

Este proyecto al estar basado en principios ágiles se ha visto necesario elaborar un producto mínimo viable el cuál presentará la aplicación en un entorno web en sus primeras demostraciones. Además, se optó por el despliegue sobre esta plataforma debido al bajo presupuesto y la incertidumbre de adquirir cuentas de desarrollo de Google y Apple para la distribución en las diferentes tiendas de aplicaciones (Google Play y App Store).

Ante este antecedente se realiza la siguiente configuración como se muestra en la Figura 22.

Figura 22. *Selección de plataforma de despliegue, y máquina de compilación.*



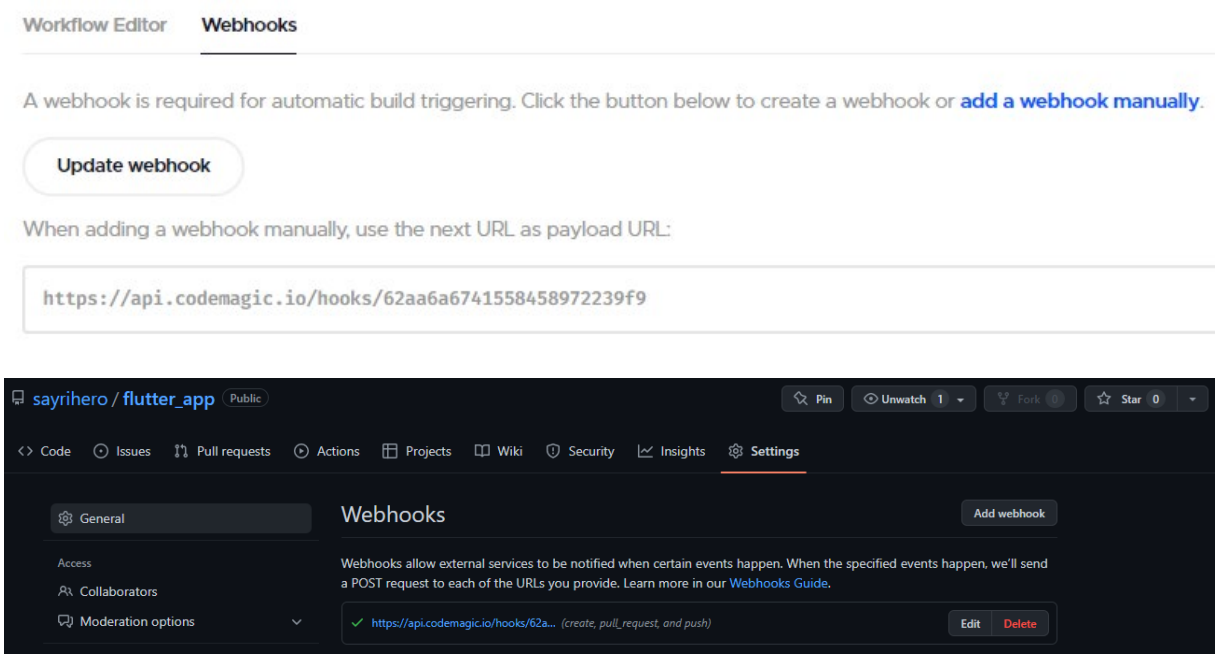
Fuente: Propia, codemagic.io

La selección de la plataforma a compilarse es Web, y la máquina virtual de compilación es la seleccionada por defecto en esta capa gratuita de CodeMagic, una instancia macOS con características más que suficientes para compilar para Web, Android y iOS en el caso de ser requerido.

4.2.5.3. Desencadenadores de compilación (Build triggers)

Al agregar un proyecto a CodeMagic, gracias a la integración con GitHub, los Webhooks son configurados automáticamente. Los **Webhooks** son necesarios para poder activar el ciclo del pipeline automáticamente en respuesta a eventos que suceden en el repositorio, los eventos habilitados por defecto para su elección son desencadenador en Push, desencadenador en actualización de Pull Request, desencadenador en Creación de Tag. Para comprobar esta configuración se abre la sección de Webhooks y verificamos que la URL de la API de CodeMagic sea idéntica a la que se encuentra configurada en Ajustes > Webhooks del repositorio de GitHub según lo expuesto en la Figura 23.

Figura 23. *Comprobación de Webhooks en CodeMagic y repositorio de GitHub.*

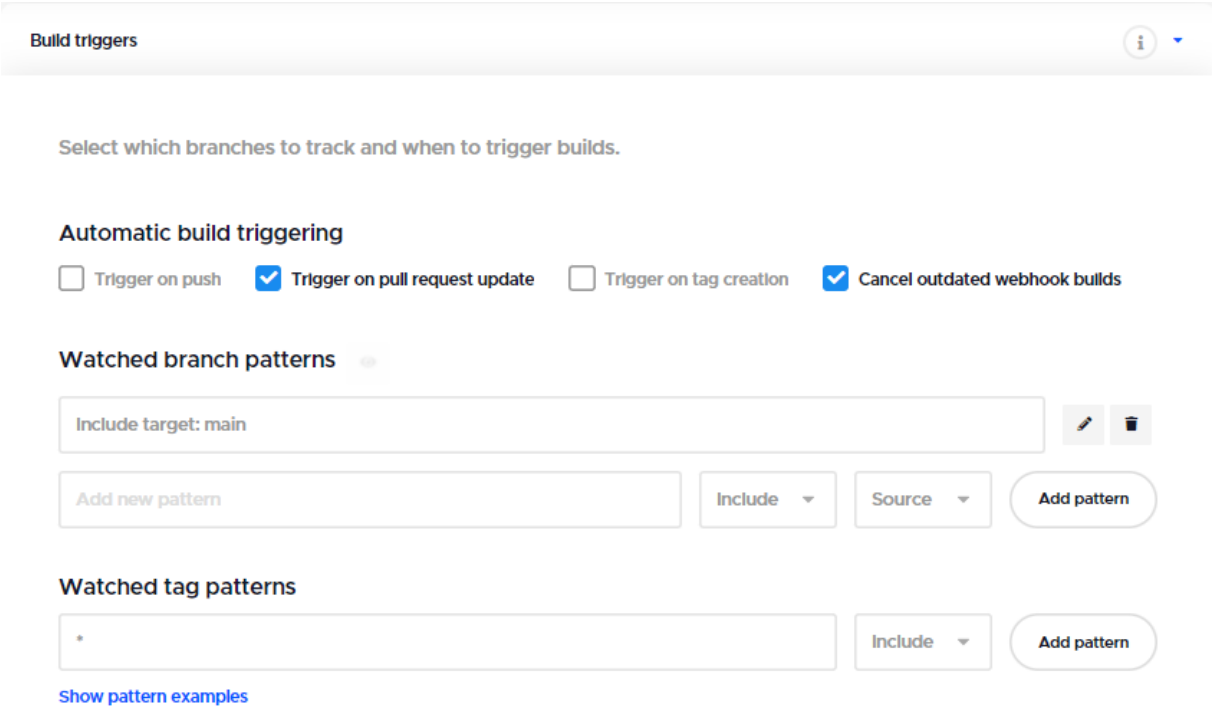


Fuente: Propia, codemagic.io

Ya comprobada la configuración de Webhooks se podría seleccionar el desencadenador en Push, pero no sería una buena práctica puesto que obligaría a los desarrolladores a probar el software localmente antes de sincronizarlo con el repositorio remoto. En lugar de optar por el desencadenador en Push, se seleccionará el desencadenador en actualización de Pull

Request, pues de esta manera se esperaría la comprobación y aceptación del administrador del repositorio para su paso a producción. Además, se recomienda habilitar la opción **Cancel outdated webhook builds** para que CodeMagic pueda cancelar automáticamente todas las compilaciones en curso y en cola cuando se haya activado una compilación más reciente en la misma rama.

Figura 24. *Configuración de desencadenadores de compilación.*



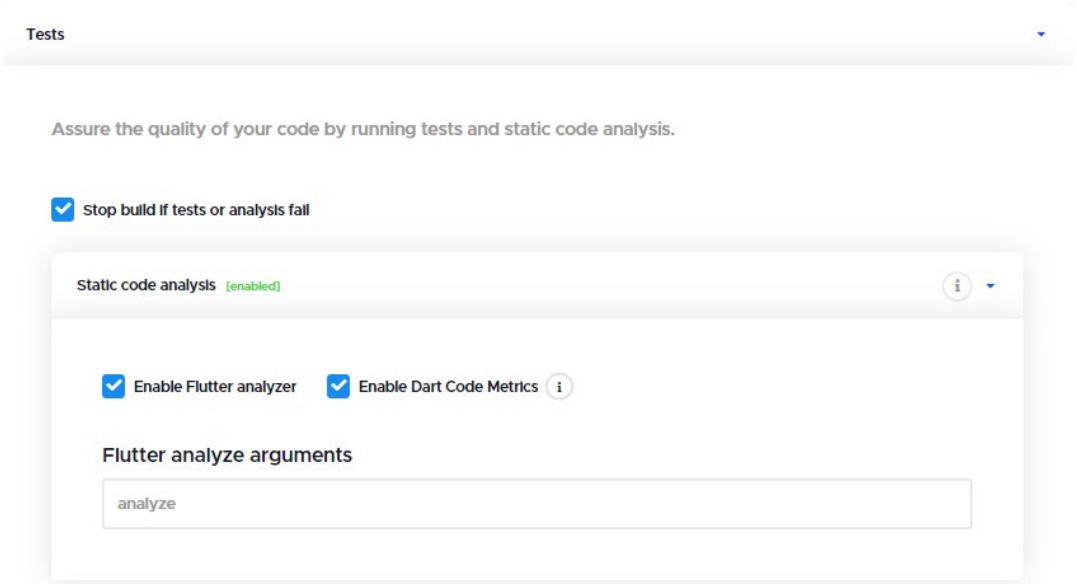
Fuente: Propia, codemagic.io

Se agregará un patrón de rama que se establecerá como objetivo a la rama main pues al recibir un pull request sólo se activará si la rama main es modificada. La configuración se muestra en la Figura 24.

4.2.5.4. Pruebas (Tests)

Para el apartado de pruebas CodeMagic nos permite habilitar el análisis estático de código mediante Flutter Analyze, el proyecto debe tener agregada la dependencia dart_code_metrics como dependencia de desarrollo y el archivo ***analysis_options.yml*** del proyecto debe estar parametrizado con reglas según nuestras necesidades. Dart Code Metrics se encarga de revisar anti-patrones y reportar métricas de código que ayudarán a monitorizar la calidad de nuestro código para así perfeccionarlo conforme suceden las iteraciones, en la Figura 25 se muestras las siguientes configuraciones.

Figura 25. *Habilitación de análisis estático de código.*



Fuente: Propia, codemagic.io

Dentro de las pruebas también está la posibilidad de habilitar Flutter test para pruebas unitarias, widgets, api, etc. así como también Flutter drive que son las pruebas de interfaz de usuario y de integración ya que simula y automatiza un flujo de proceso de la aplicación, como por ejemplo un inicio de sesión donde se ingresa el campo de usuario y contraseña para finalmente tocar el botón de inicio de sesión, todo esto se programa en la prueba. Para habilitar este tipo de prueba es necesario agregar la dependencia integration_test en la sección de dependencias de desarrollo del archivo ***pubsec.yaml*** y objetivamente crear la prueba de este tipo.

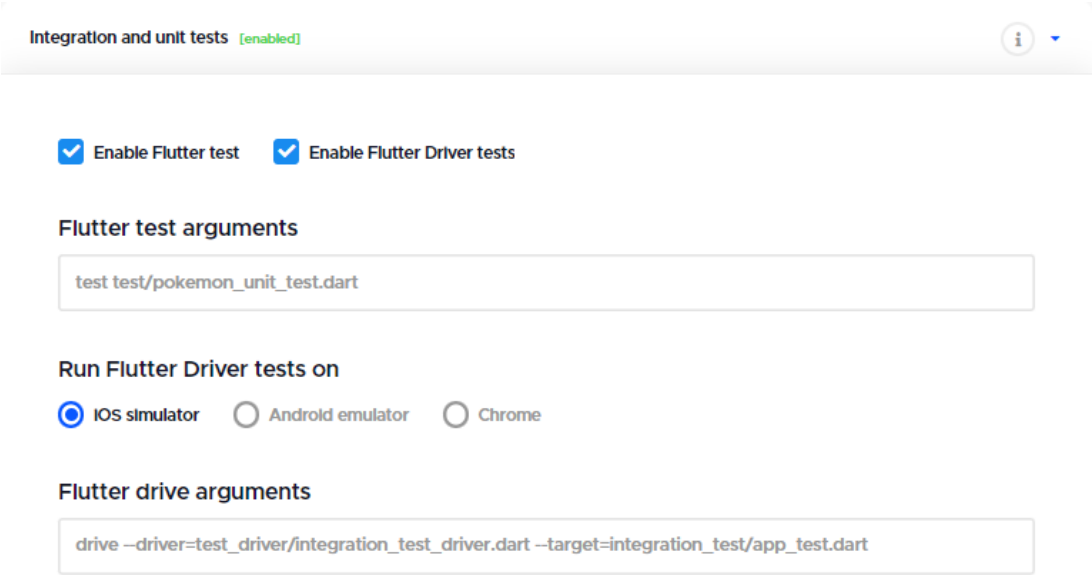
CodeMagic permite arrancar la prueba de Flutter drive sobre un ambiente simulado de iOS, emulador de Android o Chrome. En vista de que en el entorno de desarrollo no se puede realizar una prueba directa sobre un dispositivo iOS se configura la prueba sobre una simulación de iOS.

El argumento de Flutter test es el siguiente ***test test/pokemon_unit_test.dart*** y como objetivo de este trabajo se ejecutará las pruebas de este tipo, claro está que puede ejecutarse toda la carpeta de pruebas, pero en este caso ejecutaremos un solo archivo de prueba unitaria.

El argumento de Flutter drive será el siguiente ***drive --driver=test_driver/integration_test_driver.dart --target=integration_test/app_test.dart*** el cual identifica el archivo driver y el archivo objetivo de pruebas, estos dos archivos han sido

desarrollados para ejecutar la prueba de integración de interfaz de usuario, tal y como se especifica en la Figura 26.

Figura 26. *Habilitación de pruebas unitarias y de integración.*



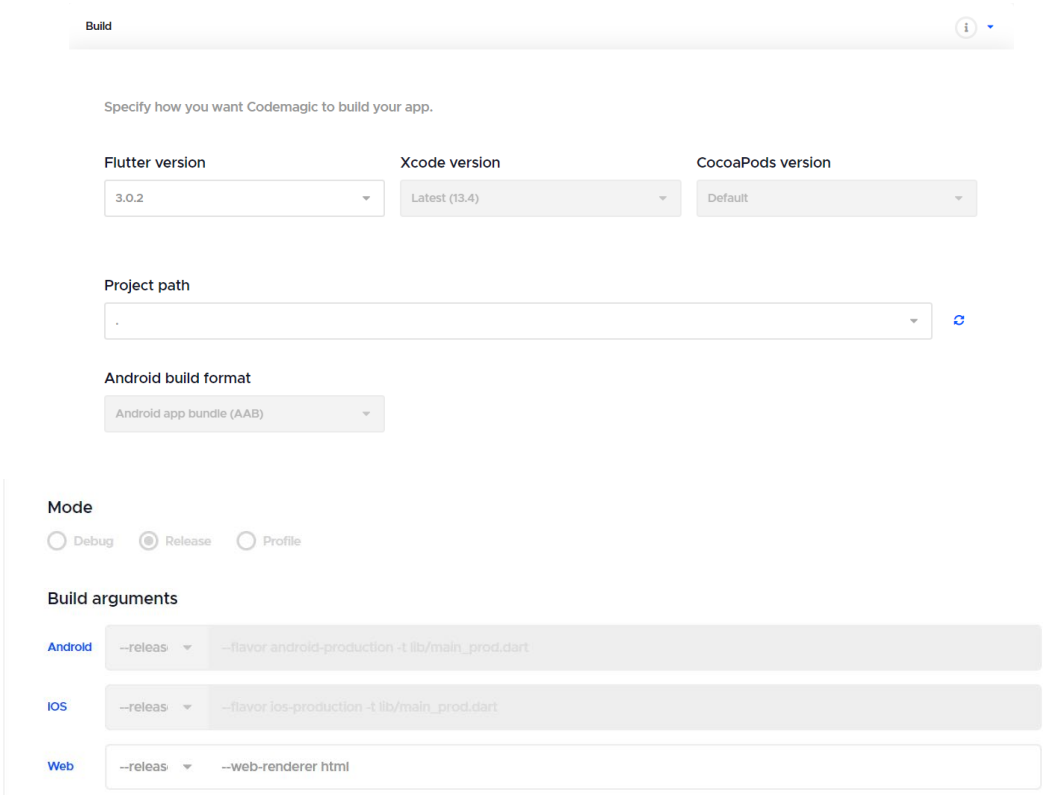
Fuente: Propia, codemagic.io

EL proceso de pruebas ha sido configurado con la opción de **parar** el pipeline si las pruebas fallan.

4.2.5.5. Compilación (Build)

Dependiendo de la o las plataformas objetivo a las que se va a compilar, las cuales se seleccionó en un punto anterior (únicamente Web), el proceso de **Compilación** habilitará las opciones a configurar, dentro de las cuales se selecciona la versión de Flutter con la que el proyecto será compilado, en este caso la versión de Flutter 3.0.2 y también los argumentos de la compilación que al tratarse de un despliegue a producción se selecciona una compilación de tipo --release con la opción de renderizado web como HTML, para que el aplicativo web no presente errores en su ejecución. La configuración de este proceso se puede apreciar en la Figura 27.

Figura 27. *Configuración del proceso de compilación.*



Fuente: Propia, codemagic.io

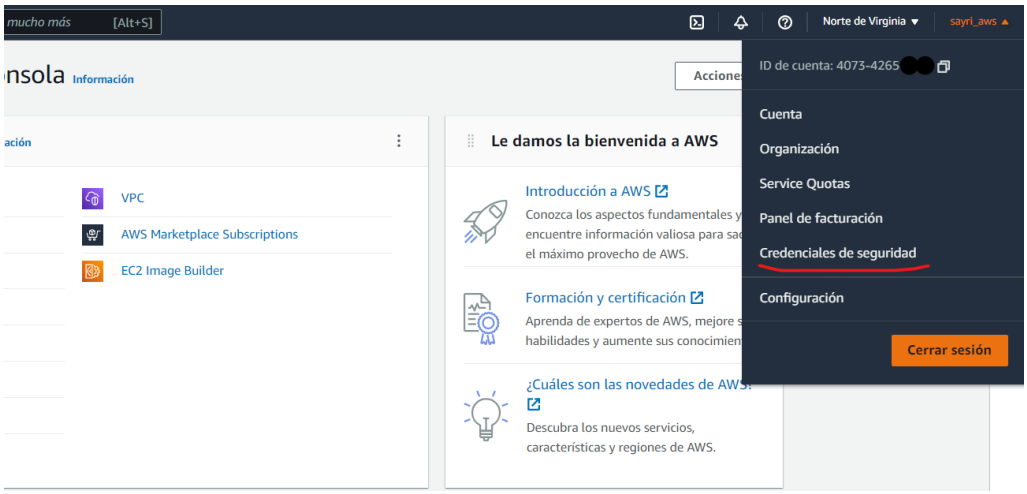
Opciones como la versión de Xcode y CocoaPods son habilitadas sólo si se seleccionó la plataforma iOS para compilación y de igual forma la selección del formato de compilación de Android en el caso de haber selecciona la plataforma Android. Los argumentos de compilación también se encuentran bloqueados ya que no será posible compilar para estas plataformas, a no ser que se agregue iOS y Android como plataformas objetivo de compilación.

4.2.5.6. Distribución (Distribution)

La configuración de este proceso se centra en establecer los puntos de distribución y despliegue de la aplicación, pues al tratarse de una distribución web, se presentará las opciones de CodeMagic Static Pages y AWS S3 Bucket. Se podría configurar el despliegue hacia los dos servicios, pero como objetivo del trabajo se desplegará sobre un bucket S3 de AWS, el cual se configuró en el punto 4.2.4.1. y 4.2.4.2.

Para habilitar el despliegue hacía el bucket es necesario obtener las claves de acceso de la cuenta de AWS, para lo cual se ingresa al usuario raíz de AWS y se abre la consola de administración. Ya en la consola se dirige a la opción de menú Credenciales de seguridad.

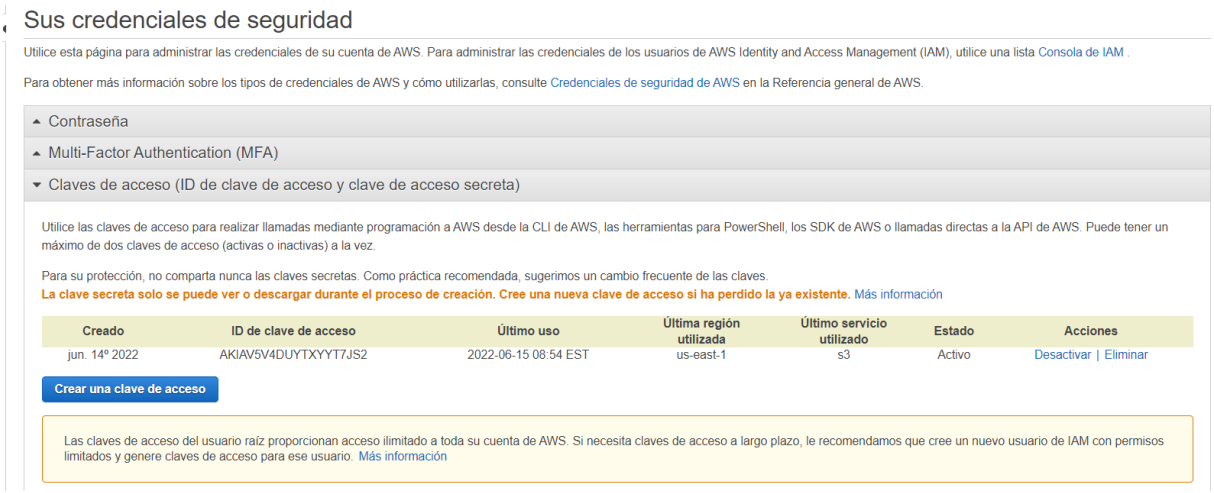
Figura 28. *Acceso a Credenciales de seguridad.*



Fuente: Propia, aws.amazon.com

Dentro de credenciales de seguridad se dirige al apartado de claves de acceso para obtener el ID de clave de acceso y la clave de acceso secreta, si no se tiene aún se debería crear unas claves nuevas.

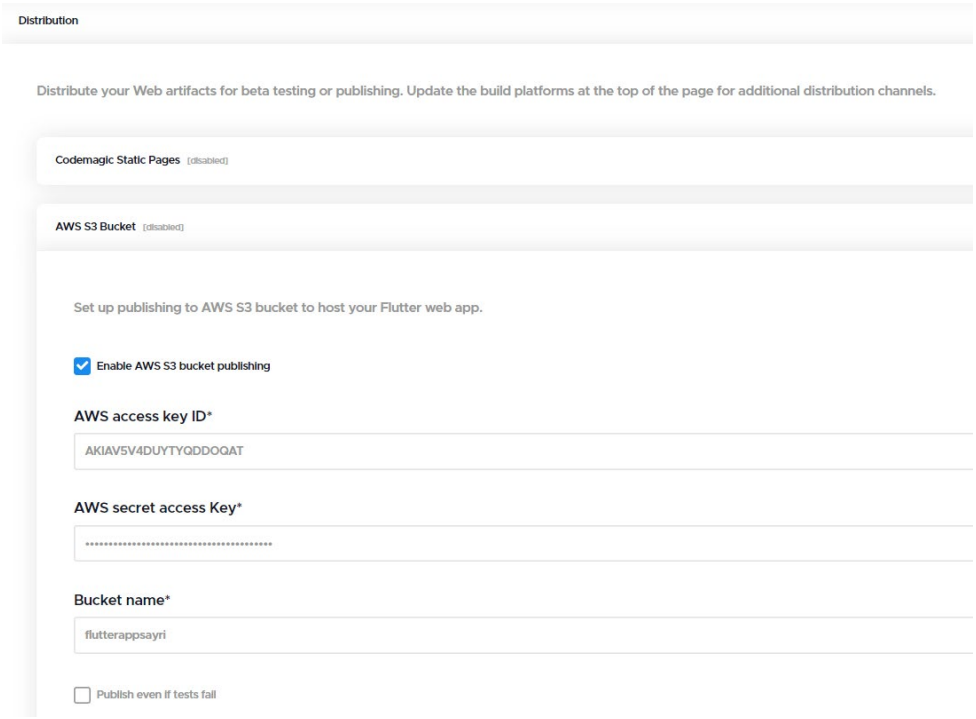
Figura 29. *Claves de acceso de cuenta raíz de AWS.*



Fuente: Propia, aws.amazon.com

Una vez que se haya generado las claves de acceso procedemos a habilitar el despliegue hacia el bucket S3 de AWS, registrando las claves de acceso, colocando el nombre del bucket y deshabilitando la opción de publicación aún si el proceso de pruebas falla, siguiendo la configuración de la Figura 30.

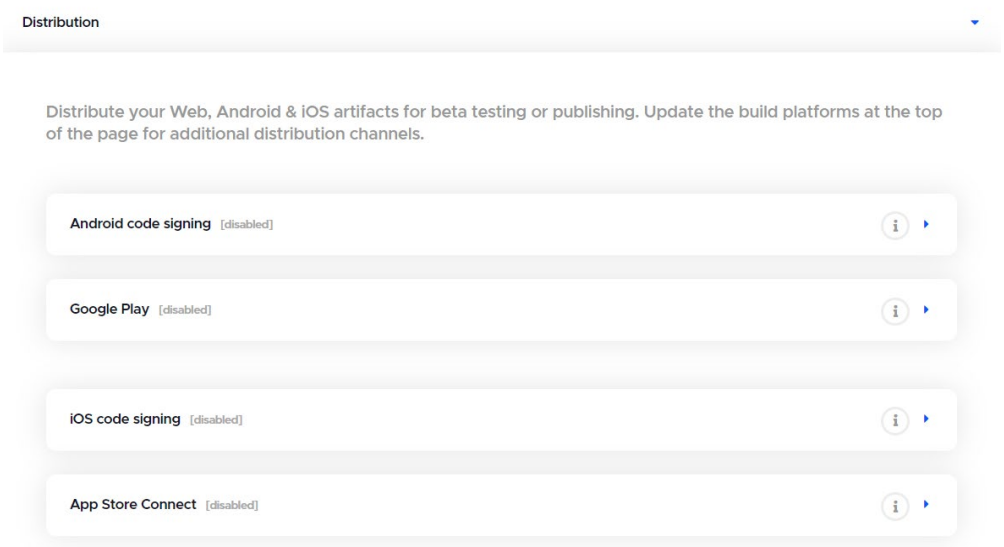
Figura 30. *Configuración de distribución a bucket S3 de AWS.*



Fuente: Propia, codemagic.io

El proceso de distribución también presenta cambios dependiendo la plataforma objetivo de compilación, pues si se selecciona iOS y Android, aparecerán las opciones de firmas de código y la configuración de conexión con las respectivas tiendas de aplicaciones para su distribución, sea para una distribución interna, alpha, beta, o producción.

Figura 31. *Alternativas en el caso de distribución a tiendas de aplicaciones.*



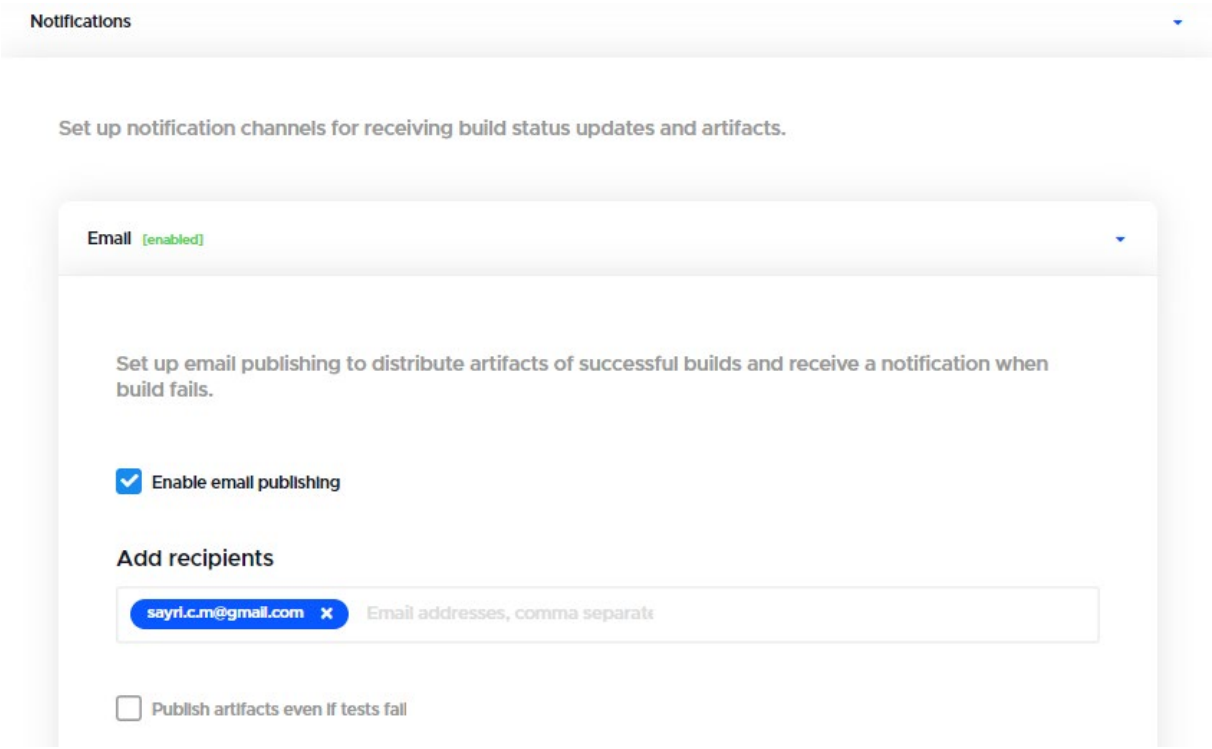
Fuente: Propia, codemagic.io

Cabe recalcar que para proceder con la configuración de distribución hacia tiendas de aplicaciones es necesario obtener cuentas con una suscripción de desarrollador en cada una de las plataformas sea Google para Android o Apple para iOS.

4.2.5.7. Notificaciones (Notifications)

La configuración del proceso de notificaciones contempla la creación de un canal para envío de actualizaciones de estado de compilación y artefactos. Las notificaciones pueden ser configuradas para que puedan ser enviadas vía correo electrónico o slack una herramienta de comunicación muy utilizada entre equipos de desarrollo. El alcance de este trabajo contemplará las notificaciones mediante correo electrónico, cabe recalcar que esta configuración (Figura 32) viene activada por defecto con el correo perteneciente a la cuenta de GitHub, en el caso de ser necesario se ingresará más correos de destino adicionales que deben ser separados por coma.

Figura 32. *Configuración de notificaciones de estado de compilación y artefactos.*



Fuente: Propia, codemagic.io

De la mano de CodeMagic se puede ejecutar un pipeline CI/CD para construir, probar y publicar aplicaciones de Flutter con una configuración sencilla y con la opción de ejecutar construcciones en entornos controlados utilizando flujos de trabajo personalizados.

Una vez finalizados los cambios se recomienda que la primera ejecución del pipeline se ejecute manualmente presionando el botón **Start your first build** ya que de esta manera se comprobará que el pipeline funciona desde su primera ejecución, para continuar con la evaluación se dará la recomendación por hecha y se continuará con lo expuesto anteriormente para demostrar la automatización del pipeline.

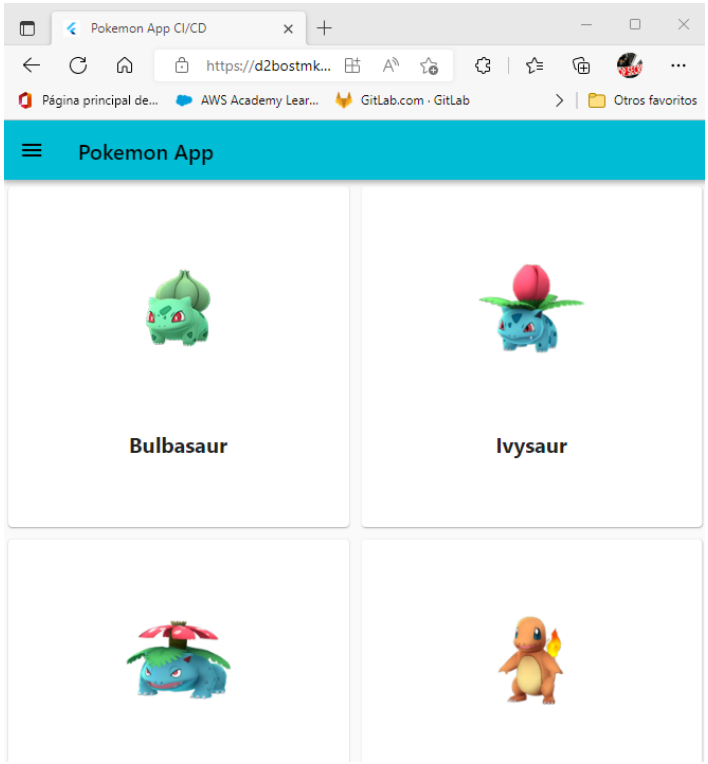
4.3. Evaluación

4.3.1. Prueba práctica de pipeline de CodeMagic

Para evaluar si el pipeline se ejecuta satisfactoriamente con las configuraciones que se han expuesto se realizará un pequeño cambio en el fichero ***main.dart*** que forma parte del directorio **flutter_app/lib** el cuál es un directorio principal donde el código fuente de un aplicativo de Flutter es agregado. El pipeline está configurado para que se ejecute cada vez que se realiza un push desde una rama de desarrollo local hacia el repositorio remoto de GitHub, generando la misma rama en el repositorio remoto y provocando un pull request que al ser aprobado ejecutaría el pipeline de CodeMagic a través de los disparadores y webhooks configurados.

Antes de iniciar con la evaluación exponemos en la Figura 33 el estado inicial de la aplicación web antes de la ejecución del pipeline automatizado.

Figura 33. *Estado inicial de la aplicación web desarrollada en Flutter.*



Fuente: Propia

Por lo general para provocar un pull request se debería seguir los siguientes pasos:

- Realizar un fork del repositorio antes de iniciar los cambios o incidencias.
- Clonar el repositorio que se ha hecho fork.
- Crear una nueva rama para el desarrollo de cambios o incidencias.
- Realizar el push de la nueva rama con los nuevos cambios.

Pero en este caso al usarse la cuenta administradora del repositorio de GitHub realizaremos los siguientes pasos para provocar el pull request:

- Realizar un pull para obtener los últimos cambios del aplicativo.
- Crear una nueva rama para el desarrollo de cambios o incidencias.
- Realizar el push de la nueva rama con los nuevos cambios.

Una vez que se hayan hecho los cambios necesarios en la nueva rama la cual se denomina dev-issue-1 procedemos a realizar el push de la nueva rama en el origen remoto, los pasos se encuentran en la Figura 34.

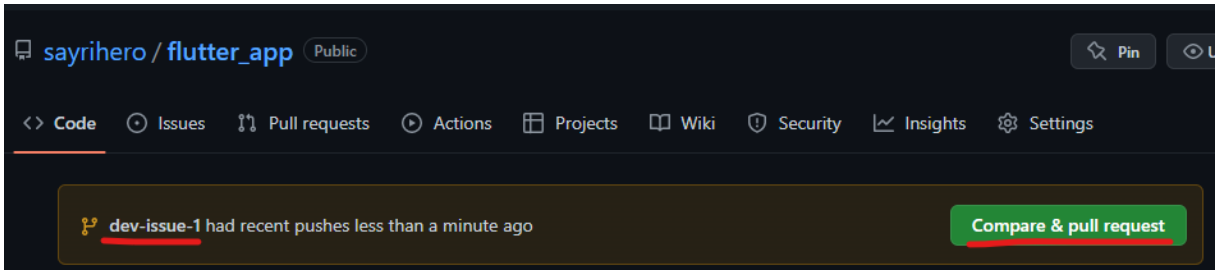
Figura 34. *Generando la ejecución del pipeline al hacer un push que provocará una pull request.*

```
TERMINAL   JUPYTER   DEBUG CONSOLE   PROBLEMS   OUTPUT

PS F:\CICD\flutter_app> git checkout -b "dev-issue-1"
Switched to a new branch 'dev-issue-1'
PS F:\CICD\flutter_app> git status
On branch dev-issue-1
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   lib/main.dart
        modified:   lib/pokemondetalle.dart

no changes added to commit (use "git add" and/or "git commit -a")
PS F:\CICD\flutter_app> git add ..
fatal: ..: '..' is outside repository at 'F:/CICD/flutter_app'
PS F:\CICD\flutter_app> git add .
PS F:\CICD\flutter_app> git commit -m "Commit de rama dev-issue-1"
[dev-issue-1 72c16fa] Commit de rama dev-issue-1
 2 files changed, 11 insertions(+), 11 deletions(-)
PS F:\CICD\flutter_app> git push origin "dev-issue-1"
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 12 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 547 bytes | 547.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
remote:
remote: Create a pull request for 'dev-issue-1' on GitHub by visiting:
remote:      https://github.com/sayrihero/flutter_app/pull/new/dev-issue-1
remote:
To https://github.com/sayrihero/flutter_app.git
 * [new branch]      dev-issue-1 -> dev-issue-1
PS F:\CICD\flutter_app>
```

Fuente: Propia

En la Figura 35 a continuación se puede observar que en el repositorio de GitHub ya se encuentra habilitado el botón para comparar y realizar el pull request.

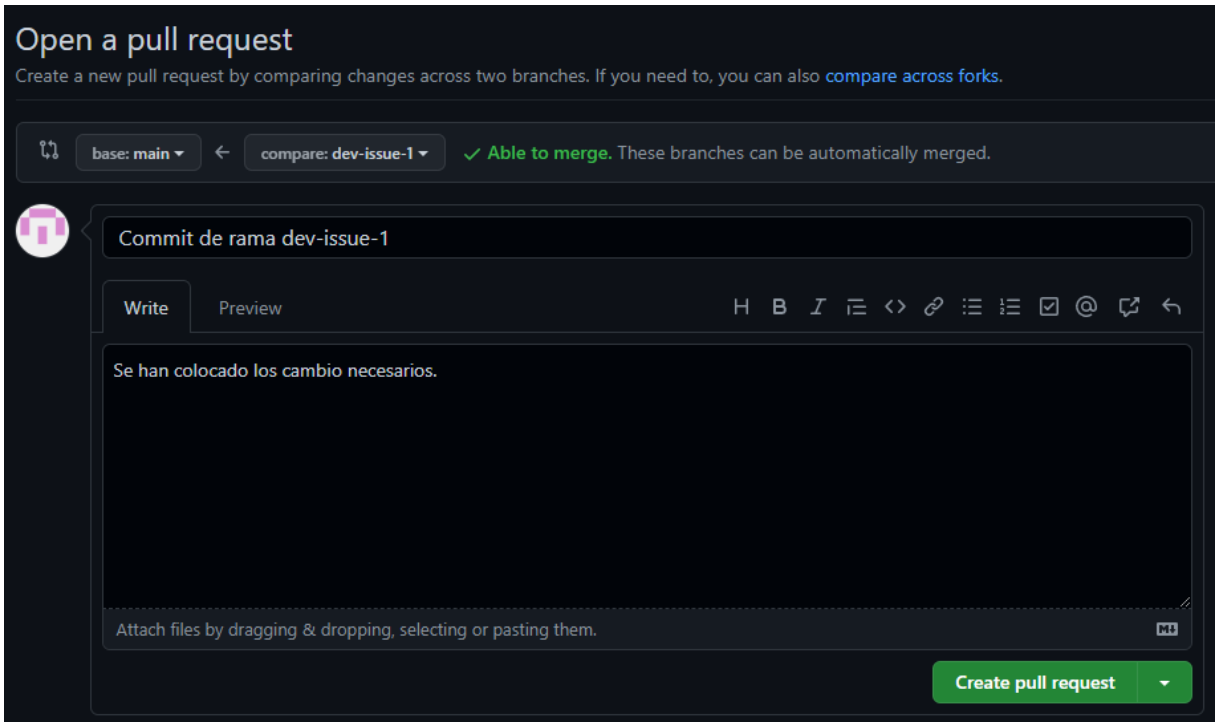
Figura 35. *Pull request y comparación de código notificados para revisión.*



Fuente: Propia

Al dar clic en el botón nos redirige a la ventana que se muestra en la Figura 36, en la cual se comentará y se creará el pull request que fusionará la rama **dev-issue-1** con **main**.

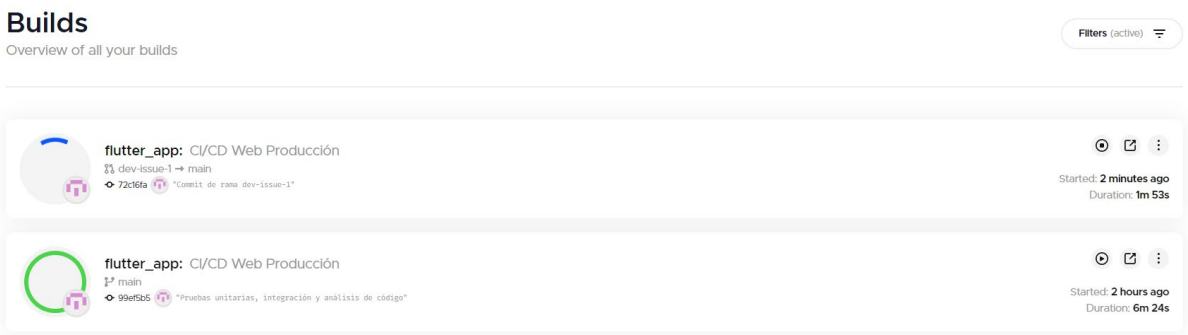
Figura 36. *Comparando y creando el pull request.*



Fuente: Propia

Al crear el pull request y fusionar la rama dev-issue-1 con la rama main automáticamente el pipeline iniciará con la construcción del pipeline pasando por cada uno de los escenarios configurados.

Figura 37. *El pull request genera la ejecución del pipeline.*

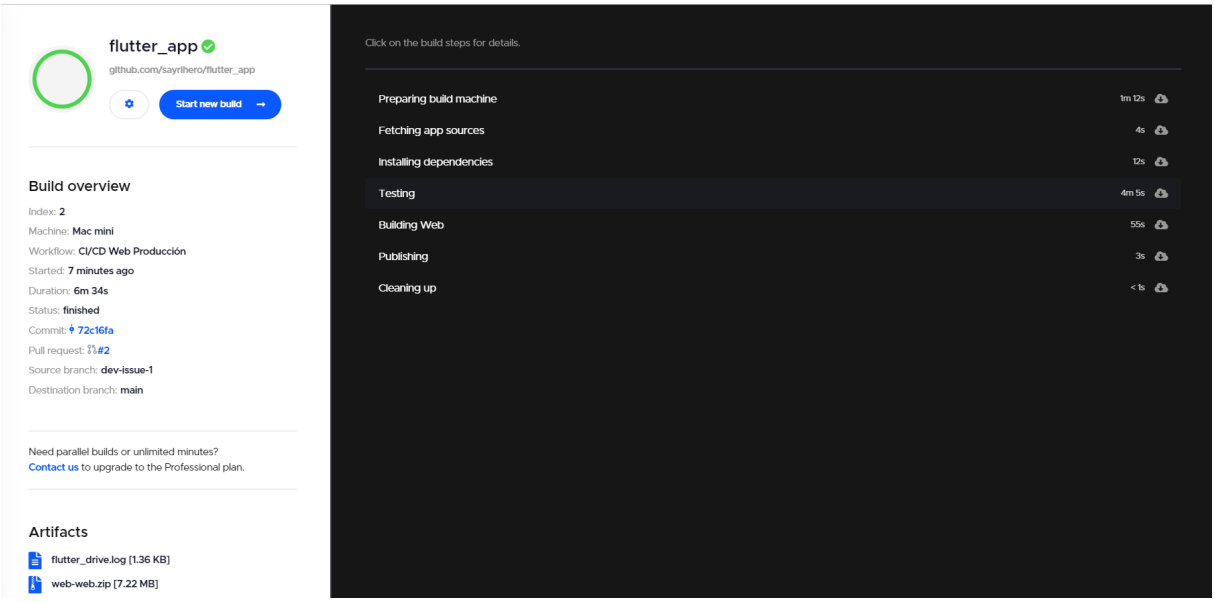


Fuente: Propia

En la Figura 38 se puede observar el detalle de cada escenario realizado para la construcción de la aplicación, iniciando por la preparación de la máquina virtual que compilará el código, obtiene el código del proyecto e instala las dependencias, realiza el análisis del código fuente,

ejecuta las pruebas (unitarias, integración), compila para la web si pasa las pruebas, publica la aplicación en el bucket de Amazon S3, notifica por correo electrónico y realiza una limpieza.

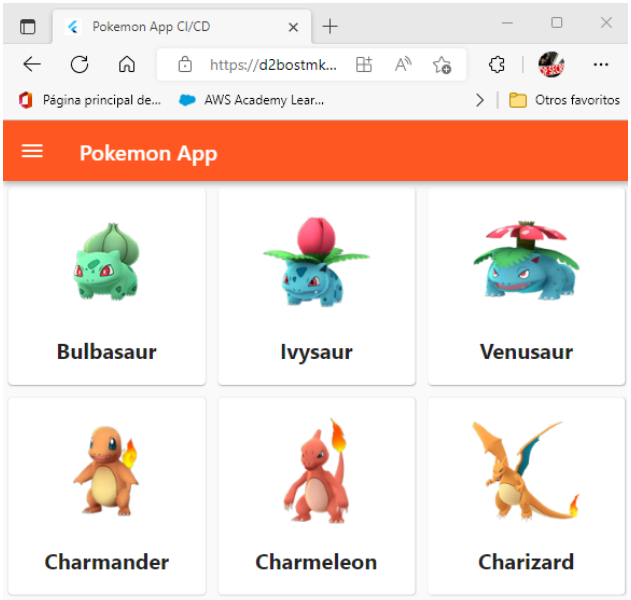
Figura 38. *Construcción finalizada correctamente.*



Fuente: Propia

Además, existe la posibilidad de descargar los artefactos (compilado del proyecto en carpeta comprimida, y resultado de pruebas de integración) que se muestran debajo de la sección Artifacts, como también cada uno de los logs generados en cada escenario. Para comprobar que los cambios se han efectuado se ingresa nuevamente a la página web del aplicativo, y como se muestra en la Figura 39 los cambio han sido aplicados y se encuentran en producción.

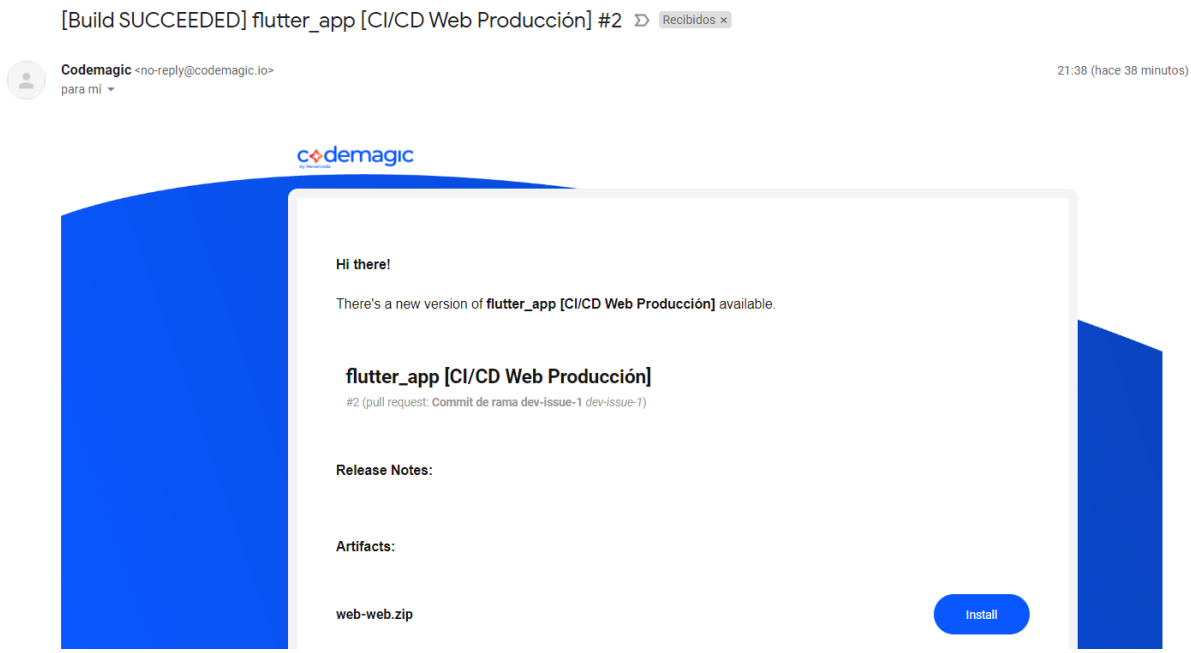
Figura 39. *Cambios y nuevas funcionalidades generadas por el pipeline automatizado.*



Fuente: Propia

Finalmente, un correo electrónico es enviado por CodeMagic con el resultado del estado final de la construcción del pipeline, sea que haya finalizado con éxito o haya fallado.

Figura 40. *Correo electrónico de notificación de estado final de construcción.*



Fuente: Propia

La ejecución del pipeline ha finalizado con éxito, y ahora ya se encuentra configurado y funcionando. Se podría seguir usando la configuración del pipeline tal cuál, pero el despliegue

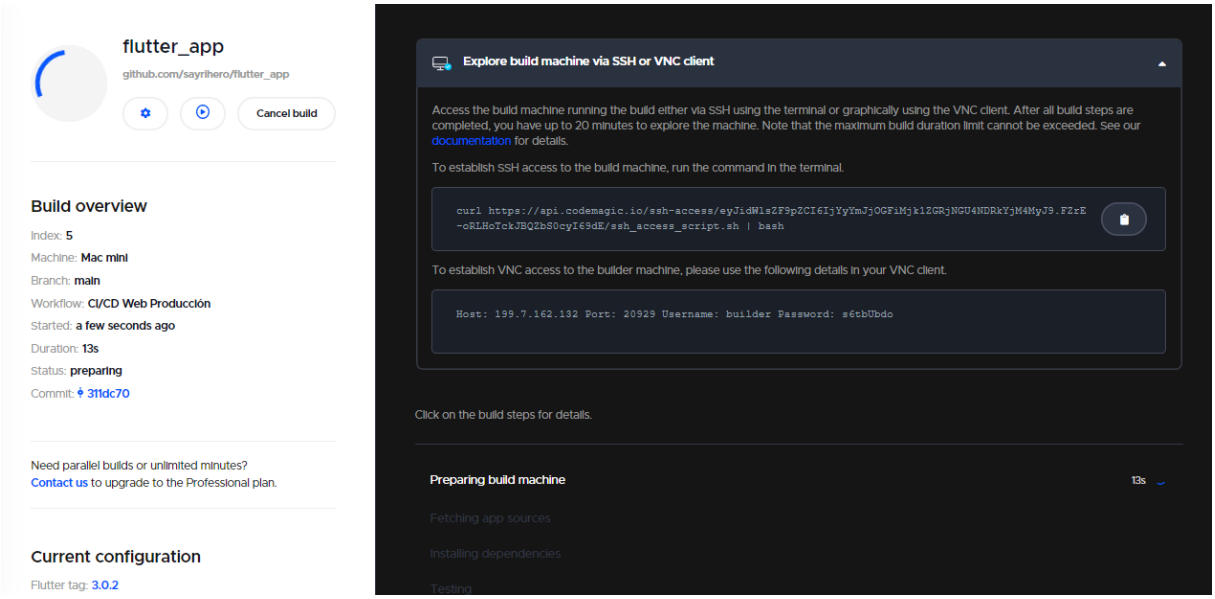
de la aplicación está limitado a un ambiente de producción, para que sea un pipeline más robusto se podría crear un flujo de trabajo para el despliegue a un ambiente previo de staging y una vez pase por la revisión y aprobación de ese ambiente podría desplegarse a producción. Otra de las mejoras a implementar podría ser la construcción de más pruebas unitarias, de widgets y de integración que cubran el 80% del código generado.

Esto concluye que la implementación de un pipeline CI/CD automatizado puede soportar la distribución simultanea de software, ejecutar pruebas automatizadas, ejecutar compilaciones en simultaneo para diferentes plataformas, proporcionar artefactos, y tener varios flujos de trabajo para separar los despliegues a los diferentes ambientes.

4.3.2. Acceso remoto a máquina virtual de compilación para prueba manual de la aplicación en dispositivo iOS

Al momento de la construcción y ejecución del pipeline se tiene la opción para crear una conexión con la maquina virtual de compilación mediante una conexión VNC (Virtual Network Computing) por lo que se recomienda instalar en el equipo el programa VNC Viewer https://www.realvnc.com/es/connect/download/viewer/ el cual nos permitirá establecer una conexión remota a la máquina virtual que CodeMagic nos otorga. En la Figura 41 podemos observar como en la sesión del pipeline que se está ejecutando se encuentran las credenciales de acceso vía VNC, cabe recalcar que dichas credenciales varían en cada sesión o ejecución del pipeline.

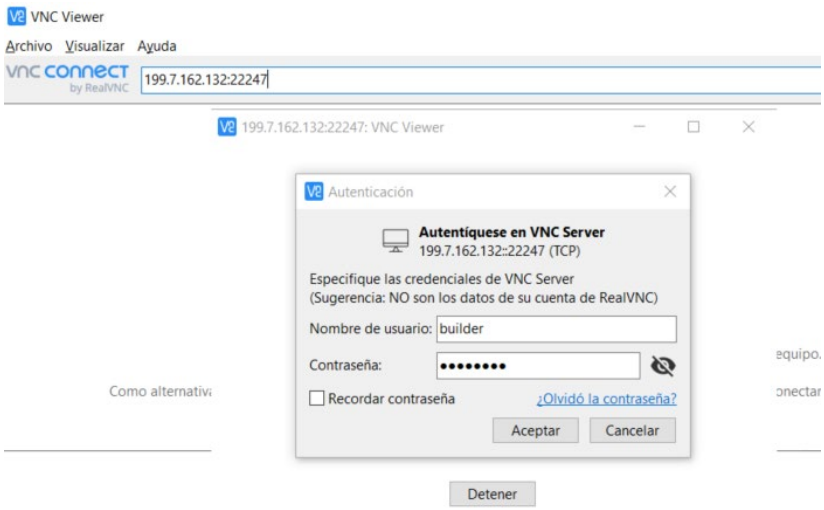
Figura 41. *Credenciales de ingreso a máquina virtual vía SSH o cliente VNC.*



Fuente: Propia

Las credenciales que se indican son la ip del host, el puerto de conexión, el usuario y la contraseña de acceso, mismas que se deben ingresar en el programa VNC Viewer como se muestra a continuación.

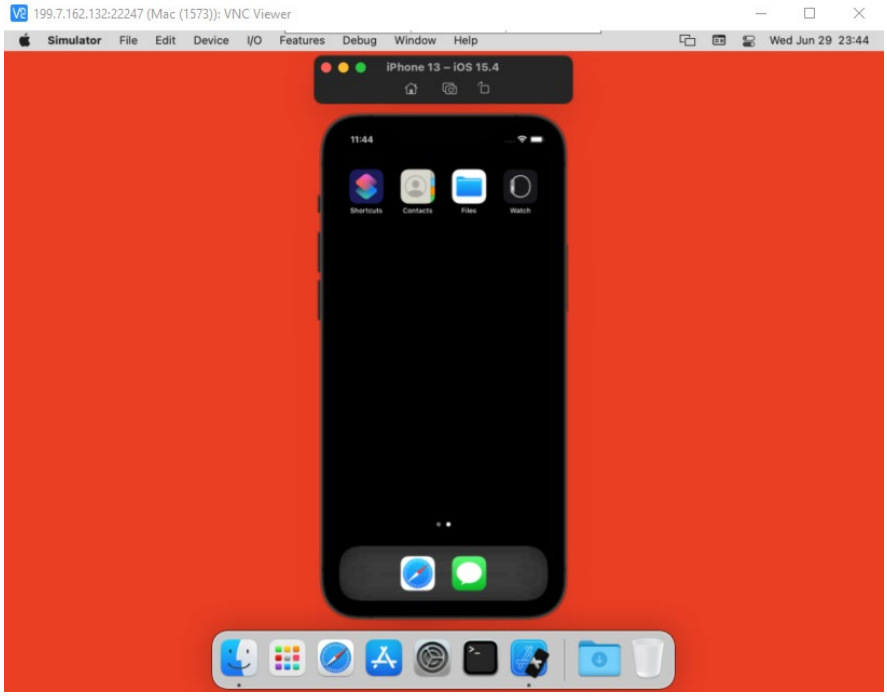
Figura 42. *Configuración de conexión a máquina virtual.*



Fuente: Propia

Una vez que se haya accedido a la máquina virtual aparecerá el escritorio de macOS como en la Figura 43, si se ingresa antes de que se haya ejecutado las pruebas se tendrá la oportunidad de visualizar la prueba automatizada de integración en el simulador que se muestra en pantalla.

Figura 43. *Pantalla al acceder a la máquina virtual vía VNC Viewer.*



Fuente: Propia

Una vez finalizada la prueba de integración el simulador se cerrará, no habrá que preocuparse por la ejecución del pipeline puesto que se ejecuta en 2do plano, pues bien, con el propósito de realizar una prueba manual del aplicativo sobre un dispositivo simulado, en este caso un iPhone 13 con iOS 15.4 que viene por defecto, abriremos el terminal en macOS para correr los siguientes comandos:

Habrá que dirigirse a la carpeta clone ya que es ahí donde CodeMagic clona e instala las dependencias del proyecto de Flutter.

$ cd clone

Se debe abrir el simulador del dispositivo.

$ open -a simulator

Una vez que el simulador se haya cargado completamente se ejecutará la depuración del proyecto, mismo que seleccionará para su ejecución el simulador que se abrió en el paso previo.

$ flutter run

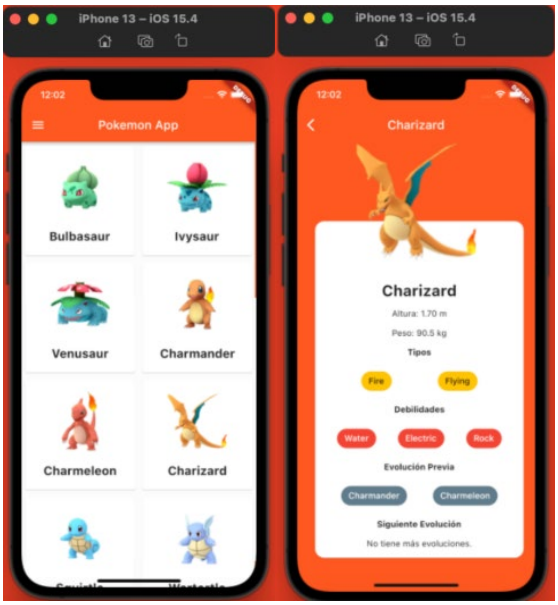
En la Figura 44 se muestra la ejecución de cada uno de los comandos.

Figura 44. *Ejecución de comandos para depurar el aplicativo en el simulador.*

Fuente: Propia, terminal de máquina virtual de compilación

La depuración del aplicativo tal y como se muestra en la Figura 45, representa una gran ventaja para el desarrollador puesto que en muchas de las ocasiones no se cuenta con un dispositivo de Apple al alcance de la mano o presupuesto para realizar este tipo de pruebas.

Figura 45. *Depuración de aplicativo en simulador (iPhone 13 – iOS 15.4).*



Fuente: Propia, simulador en máquina virtual de compilación

Una vez que se cierre la conexión a la máquina virtual también terminará la ejecución del pipeline que se mantuvo ejecutando durante el tiempo que se mantuvo abierta la conexión a esta.

5. Conclusiones y trabajo futuro

5.1. Conclusiones

El concepto de este trabajo de fin de máster se originó a partir de una inquietud con respecto al automatismo en el ciclo de desarrollo de software gracias al uso de herramientas DevOps que podrían aplicarse en el lugar de trabajo. En estos últimos años que se ha visto una tendencia creciente de las prácticas de DevOps y el uso de la integración y despliegues continuos, mismos que ayudan a construir software de calidad a un menor tiempo. Además, dentro de la comunidad de desarrolladores de software se puede apreciar que existe el miedo a lo desconocido que afecta a dar el salto a una mentalidad DevOps. Dando como resultado que una práctica como la adopción de la automatización de pipelines CI/CD, no se le dé el uso que debería o que ni si quiera se lo ponga en práctica. Uno de los objetivos de este trabajo, junto con otros explicados alrededor de capítulos anteriores era el de demostrar que esta adopción no es tan difícil de dar.

La implementación explicada en los capítulos previos es un pipeline CI/CD completo el cual requiere de una configuración mínima, que se integra bien con las herramientas y tecnología existentes de las prácticas de desarrollo de software más comunes, soportando construcción de software con múltiples componentes, ejecutando pruebas automáticas antes de la construcción, guardando diferentes versiones en un almacenamiento de artefactos y desplegando software en un ambiente de producción definido.

Con base a lo expuesto en el antecedente teórico y la implementación práctica de la idea, ahora está bastante claro que cualquier desarrollador de software o un equipo de desarrollo puede simplemente crear una integración, entrega y despliegue continuos minimalista, rentable y eficiente en el contexto del desarrollo de aplicaciones móviles nativos multiplataforma. Como se demuestra en la práctica la creación e implementación del pipeline CI/CD no están difícil y brinda muchos beneficios. La práctica de configurar y utilizar dichos pipelines automatizados es compatible con la forma de trabajo ágil y la mentalidad DevOps del desarrollo de software.

Sin embargo, dado que todo trabajo de desarrollo de software es diferente y existen diferencias en la preferencia de herramientas y tecnología utilizadas por los propios desarrolladores, la arquitectura de un posible pipeline CI/CD debe establecerse con cautela.

Un pipeline CI/CD puede ser simple, si el proyecto no requiere todos los aspectos de un pipeline o ser más complejo si el proyecto requiere de muchos pasos o involucra una gran cantidad de procesos intermedios. Una configuración simple de CI/CD puede funcionar como punto de partida para el comienzo del proyecto de desarrollo de software y a medida que el equipo o trabajo de desarrollo avanza, se puede agregar fácilmente más componentes.

Las herramientas utilizadas en esta implementación pueden tener varias alternativas, y en algunos casos dichas alternativas pueden brindarnos ciertos beneficios adicionales. Específicamente GitHub podría ser fácilmente reemplazado por Bitbucket o GitLab como proveedor de repositorio en la nube, AWS podía haber sido reemplazado por Microsoft Azure o GCP (Google Cloud Platform). La elección es, y siempre debe ser del desarrollador, dependiente del tipo de proyecto y especificación de requisitos, para decidir qué tipo de herramientas y tecnología se debe usar.

Trabajar con una mentalidad DevOps, adoptando la agilidad como una forma de trabajo y usando pipelines CI/CD, tiene claros beneficios y mejora la calidad no solo del software sino de todo el proceso que conlleva. Es beneficioso en un sentido financiero para las empresas que adoptan estas prácticas y alientan a sus desarrolladores a usar un ciclo CI/CD automatizado. Además, ha quedado claro que tras la práctica e investigación de un ciclo automatizado CI/CD se puede reducir el riesgo de que el proyecto falle, al crearse una retroalimentación continua mejora la comunicación y aumenta la eficiencia en general de los equipos de desarrollo de software.

5.2. Líneas de trabajo futuro

Para los desarrolladores de software, equipos de desarrollo o cualquier entusiasta que quiera implementar la automatización en el ciclo de desarrollo visto en este trabajo, se le propone los siguientes puntos para perfeccionar la implementación.

En CodeMagic se puede optar por una configuración declarativa del pipeline CI/CD en formato **YAML** en lugar del uso de Workflow Editor (alternativa de configuración con interfaz gráfica), el uso de la configuración a través del archivo **codemagic.yaml** provee una configuración altamente personalizable que será útil para una definición más robusta del flujo de trabajo del pipeline, y a través del archivo será posible mantener un histórico de su desarrollo puesto que puede incluirse en el sistema de control de versiones. La razón por la que no se aplicó el uso

del archivo codemagic.yaml, radica en la justificación de este trabajo ya que se intenta demostrar que cualquier desarrollador de software puede implementar fácilmente dicha tecnología y esto fue posible con el uso de Workflow Editor, pues al utilizar el archivo YAML la configuración se torna más avanzada.

El despliegue web de la aplicación de Flutter en este trabajo es considerado como un MVP (Minimum Viable Product) pues se destaca la posibilidad de extender la compilación y despliegue de la aplicación para Android y iOS, añadiendo un par configuraciones en el pipeline CI/CD de CodeMagic en las etapas de construcción y distribución. En este trabajo ha primado el uso de herramientas gratuitas y de código abierto ya que uno de los objetivos del trabajo es tratar de no incurrir en costos adicionales, pues no se llegó a la distribución de tiendas de aplicaciones puesto que distribuirlas con lleva una inversión adicional en cuanto a suscripciones que se debe conseguir como desarrollador, tanto para la AppStore de Apple como para Google Play de Google. La distribución hacia las tiendas dentro del flujo de trabajo del pipeline de CodeMagic es de gran ayuda ya que libera tiempo con la entrega hacia estas y garantiza que el próximo lanzamiento de la aplicación sea rápido, confiable y simultaneo en las dos tiendas.

CodeMagic cuenta con un sinnúmero de integraciones sean estas de pruebas automatizadas, pruebas en dispositivos reales (AWS Device Farm, Firebase Test Lab), notificaciones (Discord, Microsoft Teams, Slack), detectores de incidencias (Jira o Trello), herramientas de desarrollo (Fastlane, Firebase, Gradle), distribución (GitHub releases, Firebase App Distribution), alojamiento de repositorios (Bitbucket, Azure DevOps, AWS CodeCommit, GitLab). Finalmente, en este trabajo se motiva a explorar cada una de las integraciones de CodeMagic ya que se logrará el desarrollo de un pipeline más robusto y automatizado.

Referencias bibliográficas

Alrumayh, A. S., Lehman, S. M., & Tan, C. C. (2021). Emerging mobile apps: challenges and open problems. *CCF Transactions on Pervasive Computing and Interaction*, 57-75. doi:https://doi.org/10.1007/s42486-020-00055-x

Aymeric, H. L. (2020). *From agile to DevOps: Smart skills and collaborations.* (Vol. 22). New York: Information Systems Frontiers. doi:http://dx.doi.org/10.1007/s10796-019-09905-1

Bellido, R. (9 de septiembre de 2021). *Aplicaciones híbridas: qué son, usos y ejemplos*. Obtenido de Deusto Formación: https://www.deustoformacion.com/blog/apps-moviles/todo-sobre-aplicaciones-hibridas

Codemagic - CI/CD for android, iOS, flutter and react native projects. (2022). Recuperado el 22 de junio de 2022, de Codemagic: https://codemagic.io/start/

Dharmwan, S. (16 de noviembre de 2021). *CYNOTECK*. Obtenido de https://cynoteck.com/es/blog-post/native-mobile-app-development-pros-cons-alternatives-and-cost-optimization/#What_is_Native_Mobile_App_Development

Domínguez Acosta, M. F. (Mayo de 2021). *https://semana.mat.uson.mx.* Obtenido de XXXI SEMANA NACIONAL DE INVESTIGACIÓN Y DOCENCIA EN MATEMÁTICAS: https://semana.mat.uson.mx/semanaxxxi/carteles/uploads/15-Relaci%C3%B3n%20entre%20DevOps%20y%20las%20Metodolog%C3%ADas%20%C3%81giles.pdf

Duvall, P. M., Matyas, S., & Glover, A. (2007). *Continuous Integration: Improving Software Quality and Reducing Risk.* Addison-Wesley Professional.

Edix España. (18 de 08 de 2021). Obtenido de https://www.edix.com/es/instituto/herramientas-gestion-proyectos/

Fernández, C. (17 de marzo de 2021). *Apps multiplataforma. Qué son y características*. Obtenido de ABAMobile: https://abamobile.com/web/apps-multiplataforma-que-son-y-caracteristicas/

Flutter.dev. (s.f.). Recuperado el 21 de junio de 2022, de Build apps for any screen: https://flutter.dev/

GoodFirms. (17 de Diciembre de 2020). *What are the SDLC phases?* Obtenido de GoodFirms: https://www.goodfirms.co/glossary/sdlc/

Hemon, A., Lyonnet, B., Rowe, F., & Fitzgerald, B. (2020). From agile to DevOps: Smart skills and collaborations. Information Systems Frontiers.

Herranz, R. (2016). *Despegar con Scrum.* España: Utópica Informática.

Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Pearson Education.

Ionicframework.com. (s.f.). Recuperado el 22 de junio de 2022, de https://ionicframework.com/docs

Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016). What is DevOps? A Systematic Mapping Study on Definitions and Practices. *Association for Computing Machinery, 12*(1-11). doi:https://doi.org/10.1145/2962695.2962707

Jenkins. (2022). Recuperado el 22 de junio de 2022, de https://www.jenkins.io/

Kniberg, H., & Skarin, M. (2010). *Kanban y Scrum – obteniendo lo mejor de ambos.* Estados Unidos de América: C4Media.

Niemand, J. J. (s/f). *Building a CI/CD pipeline for mobile app deployment.* Obtenido de The OfferZen Community Blog: https://app.bibguru.com/p/8c520041-4e3f-46f2-b7fb-e8e67a438a55

Pragma, S. (2021). *Academia Pragma*. Obtenido de https://www.pragma.com.co/academia/conceptos/devops

Reactnative.dev. (s.f.). Recuperado el 21 de junio de 2022, de https://reactnative.dev/

Red Hat, I. (31 de Enero de 2018). *¿Qué son la integración y la distribución continuas (CI/CD)?* Obtenido de https://www.redhat.com/es/topics/devops/what-is-ci-cd

Stack Overflow. (2021). Obtenido de https://insights.stackoverflow.com/survey/2021#most-popular-technologies-tools-tech

Sverrisdottir, H. S., Ingason, H. T., & Jonasson, H. I. (2014). The Role of the Product Owner in Scrum-comparison between Theory and Practices. *Procedia - Social and Behavioral Sciences, 119*, 257-267. doi:https://doi.org/10.1016/j.sbspro.2014.03.030

TechTarget. (26 de agosto de 2019). *Web application (Web App)*. Obtenido de TechTarget: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app

The Digital Project Manager. (28 de enero de 2022). Obtenido de https://thedigitalprojectmanager.com/es/tools/mejores-herramientas-colaboracion/

Tsitoara, M. (2019). *Beginning Git and GitHub.* Berkeley: Apress.

Welcome to Bitrise documentation! (2022). Recuperado el 22 de junio de 2022, de Devcenter.bitrise.io: https://devcenter.bitrise.io/

Anexo A. Código fuente de pruebas unitarias

pokemon_unit_test.dart

```dart
import 'package:flutter_test/flutter_test.dart';
import 'package:flutter_app/pokemon.dart';
void main() {
  test('El mapeador debe funcionar', () {
    var pokemon = Pokemon(id: 1, num: '001', name: 'Agumon');
    Map<String, dynamic> contPokemon = pokemon.toJson();
    expect(contPokemon, isNotNull);
    expect(contPokemon.values.elementAt(0), 1);
    expect(contPokemon.values.elementAt(1), '001');
    expect(contPokemon.values.elementAt(2), 'Agumon');
  });
}
```

Anexo B. Código fuente de pruebas de integración

app_test.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:integration_test/integration_test.dart';
import 'package:flutter_app/main.dart' as app;
void main() {
  group('App Test', () {
    IntegrationTestWidgetsFlutterBinding.ensureInitialized();
    testWidgets('Full app test', (tester) async {
      app.main();
      await tester.pumpAndSettle();
      final cartaPokemon = find.byType(InkWell).first;
      await tester.tap(cartaPokemon);
      await tester.pumpAndSettle();
      final nombrePokemon = find.byKey(const Key("txtPokemonName"));
      expect(
          tester.getSemantics(nombrePokemon),
          matchesSemantics(
              label:
                  "Bulbasaur\nAltura: 0.71 m\nPeso: 6.9
kg\nTipos\nDebilidades\nEvolución Previa\nNo tiene evoluciones
previas.\nSiguiente Evolución"));
    });
  });
}
```

Anexo C. Código fuente de archivo main.dart

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:flutter_app/pokemon.dart';
import 'package:flutter_app/pokemondetalle.dart';
import 'package:http/http.dart' as http;

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Pokemon App CI/CD',
      theme: ThemeData(
        primarySwatch: Colors.deepOrange,
      ),
      home: const MyHomePage(
        title: 'Pokemon App',
      ),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key, required this.title}) : super(key: key);
  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  var url =
      "https://raw.githubusercontent.com/sayrihero/flutter_app/main/pokedex.js
on";

  int crossAxisCount = 2;
  PokeHub? pokeHub;

  @override
  void initState() {
    super.initState();
```

```
    fetchData();
  }

  @override
  Widget build(BuildContext context) {
    double tamPantalla = MediaQuery.of(context).size.width;
    if (tamPantalla < 350.0) {
      crossAxisCount = 1;
    }
    if (tamPantalla >= 350.0 && tamPantalla <= 520.0) {
      crossAxisCount = 2;
    }
    if (tamPantalla > 520.0 && tamPantalla <= 950.0) {
      crossAxisCount = 3;
    }
    if (tamPantalla > 950.0) {
      crossAxisCount = 4;
    }

    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: pokeHub == null
          ? const Center(
              child: CircularProgressIndicator(),
            )
          : GridView.count(
              crossAxisCount: crossAxisCount,
              children: pokeHub!.pokemon!
                  .map((poke) => Padding(
                        padding: const EdgeInsets.all(2.0),
                        child: InkWell(
                          onTap: () {
                            Navigator.push(
                              context,
                              MaterialPageRoute(
                                builder: (context) => PokeDetalle(
                                  pokemon: poke,
                                ),
                              ),
                            );
                          },
                          child: Hero(
                            tag: poke.id!,
                            child: Card(
                              child: Column(
                                mainAxisSize: MainAxisSize.max,
                                mainAxisAlignment:
```

```
                                MainAxisAlignment.spaceEvenly,
                            children: <Widget>[
                              Container(
                                height: 100,
                                width: 100,
                                decoration: BoxDecoration(
                                  image: DecorationImage(
                                    image: Image.network(poke.img!).image,
                                  ),
                                ),
                              ),
                              Text(
                                poke.name!,
                                style: const TextStyle(
                                  fontSize: 20.0,
                                  fontWeight: FontWeight.bold,
                                ),
                              ),
                            ],
                          ),
                        ),
                      ),
                    ),
                  ))
              .toList(),
        ),
      drawer: const Drawer(),
    );
  }

  void fetchData() async {
    var res = await http.get(Uri.parse(url));
    var decodedJson = jsonDecode(res.body);
    if (res.statusCode == 200) {
      pokeHub = PokeHub.fromJson(decodedJson);
      // ignore: no-empty-block
      setState(() {});
    } else {
      throw Exception('Fallo al cargar los datos');
    }
  }
}
```