Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

ANÁLISIS Y DISEÑO DE INFRAESTRUCTURA PARA SISTEMA DE VISUALIZACIÓN DE INFORMACIÓN UTILIZANDO PRÁCTICAS DE AUTOMATIZACIÓN DE PROCESOS MEDIANTE DEVOPS

Gustavo Adolfo Gamboa Cruz
Asesorado por el Ing. Jhonatan Wilfredo Pú Morales

Guatemala, enero de 2021

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ANALISIS Y DISEÑO DE INFRAESTRUCTURA PARA SISTEMA DE VISUALIZACIÓN DE INFORMACIÓN, UTILIZANDO PRÁCTICAS DE AUTOMATIZACIÓN DE PROCESOS MEDIANTE DEVOPS

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

GUSTAVO ADOLFO GAMBOA CRUZ

ASESORADO POR EL ING. JHONATAN WILFREDO PÚ MORALES
AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, ENERO 2021

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Inga. Aurelia Anabela Cordova Estrada
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Christian Moisés de la Cruz Leal
VOCAL V	Br. Kevin Vladimir Armando Cruz Lorente
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Inga. Aurelia Anabela Cordova Estrada
EXAMINADOR	Ing. César Augusto Fernández Cáceres
EXAMINADOR	Ing. César Rolando Batz Saquimux
EXAMINADORA	Inga. Devora Emperatriz Meza Orellana
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

ANALISIS Y DISEÑO DE INFRAESTRUCTURA PARA SISTEMA DE VISUALIZACIÓN DE INFORMACIÓN, UTILIZANDO PRÁCTICAS DE AUTOMATIZACIÓN DE PROCESOS MEDIANTE DEVOPS

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en ciencias y sistemas, con fecha 30 de junio 2019.

Gustavo Adolfo Gamboa Cruz

Guatemala, 29 de septiembre de 2020

Ingeniero
Carlos Alfredo Azurdia
Coordinador de Privados y Trabajos de Tesis
Escuela de Ingeniería en Ciencias y Sistemas
Facultad de Ingeniería - USAC

Respetable Ingeniero Azurdia:

Por este medio hago de su conocimiento que en mi rol de asesor del trabajo de investigación realizado por el estudiante **GUSTAVO ADOLFO GAMBOA CRUZ** con carné **201504429 y CUI 2903 61370 0101** titulado **"ANALISIS Y DISEÑO DE INFRAESTRUCTURA PARA SISTEMA DE VISUALIZACIÓN DE INFORMACIÓN UTILIZANDO PRACTICAS DE AUTOMATIZACIÓN DE PROCESOS MEDIANTE DEVOPS"**, lo he revisado y luego de corroborar que el mismo se encuentra concluido y que cumple con los objetivos propuestos en el respectivo protocolo, procedo a la aprobación respectiva.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Jhonatan Wilfredo Pú Morales
Colegiado No. 15,628

Jhonatan Wilfredo Pú Morales
Ingeniero en Ciencias y Sistemas
Colegiado 15,628

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 7 de octubre de 2020

Ingeniero
Carlos Gustavo Alonzo
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Alonzo:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **GUSTAVO ADOLFO GAMBOA CRUZ** con carné **201504429 y** CUI **2903 61370 0101** titulado **"ANÁLISIS Y DISEÑO DE INFRAESTRUCTURA PARA SISTEMA DE VISUALIZACIÓN DE INFORMACIÓN UTILIZANDO PRÁCTICAS DE AUTOMATIZACIÓN DE PROCESOS MEDIANTE DEVOPS"** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo aprobado.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE INGENIERÍA EN
CIENCIAS Y SISTEMAS

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación ***"ANÁLISIS Y DISEÑO DE INFRAESTRUCTURA PARA SISTEMA DE VISUALIZACIÓN DE INFORMACIÓN UTILIZANDO PRÁCTICAS DE AUTOMATIZACIÓN DE PROCESOS MEDIANTE DEVOPS",*** *realizado por el estudiante, GUSTAVO ADOLFO GAMBOA CRUZ aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

Msc. Carlos Gustavo Alonzo
Director
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 20 de noviembre 2020

Decanato
Facultad de Ingeniería
24189101- 24189102

DTG. 004.2021.

La Decana de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **ANÁLISIS Y DISEÑO DE INFRAESTRUCTURA PARA SISTEMA DE VISUALIZACIÓN DE INFORMACIÓN UTILIZANDO PRÁCTICAS DE AUTOMATIZACIÓN DE PROCESOS MEDIANTE DEVOPS,** presentado por el estudiante universitario: **Gustavo Adolfo Gamboa Cruz,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANA
FACULTAD DE INGENIERÍA

Inga. Anabela Cordova Estrada
Decana

Guatemala, enero 2021.

AACE/asga

ACTO QUE DEDICO A:

Dios	Porque su mano fuerte nos ha sostenido.
Mis padres	Osmar Orlando Gamboa y María Cristina Cruz, porque en todo momento me apoyaron durante la carrera; cada uno impregnó herramientas que me fueron útiles para alcanzar esta meta.
Mis hermanos	Osmar Josué y Pablo Daniel Gamboa Cruz, por ser un brazo de ayuda en todo momento.
Mis amigos	Julio Alberto Arango Godínez, Julia Argentina Sierra Herrera y Alba Jeanneth Chinchilla Morales, por ser las personas que me acompañaron y apoyaron durante esta travesía; cada uno de ustedes le dio sentido al día a día de la universidad.

AGRADECIMIENTOS A:

Dios	Por darme vida, sabiduría, salud y guía en todas mis decisiones.
Mis padres	Osmar Orlando Gamboa y María Cristina Cruz Patzán, como reconocimiento a sus múltiples esfuerzos, y por apoyarme para alcanzar mis metas.
Mi asesor	Ing. Jhonatan Wilfredo Pú Morales, por su paciencia y ayuda en la etapa final de mi carrera.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES V
GLOSARIO VII
RESUMEN IX
OBJETIVOS XI
INTRODUCCIÓN XIII

1. DEVOPS 1
   1.1. ¿Qué es Devops? 1
   1.2. Orígenes de Devops 2
      1.2.1. Modelo cascada 2
      1.2.2. Desarrollo de software ágil 3
      1.2.3. Orígenes del concepto Devops 4
   1.3. CALMS 5
      1.3.1. Cultura 5
      1.3.2. Automatización 6
      1.3.3. Limpieza 7
      1.3.4. Medición 8
      1.3.5. Comunicación 10
   1.4. Ciclo de vida Devops 10
      1.4.1. Desarrollo 11
      1.4.2. Integración continua 12
      1.4.3. Entrega y despliegue 14
2. PRÁCTICAS Y HERRAMIENTAS EN EL CICLO DE VIDA DEVOPS 15
   2.1. Infraestructura de desarrollo 15
      2.1.1. Versionamiento de código 15

2.1.2. Versionamiento sintáctico 15
2.1.3. Manejo de repositorios de software e integración a Devops 16
2.1.3.1. Git Flow 17
2.1.3.2. Desarrollo basado en truncamiento 18
2.1.3.3. GitLab Flow 18
2.1.3.4. Github Flow 19
2.1.4. Herramientas de manejo de código fuente 20
2.1.4.1. Github 20
2.1.4.2. GitLab 21
2.2. Prueba continua 21
2.2.1. Pruebas funcionales 22
2.2.2. Pruebas no funcionales 22
2.2.3. Creación de ambientes de calidad de software 23
2.3. Ambiente automatizado para la ejecución de pruebas y retroalimentación continua 24
2.3.1. Análisis estático de código 24
2.3.2. Análisis dinámico de código 24
2.4. Diseño del flujo de automatización 25

3. DEVOPS BASADO EN DOCKER Y JENKINS 29
3.1. Docker 29
3.1.1. Imágenes 29
3.1.2. Contenedores 30
3.1.3. Redes 30
3.1.4. Almacenamiento 31
3.1.5. Repositorio de imágenes 32
3.2. Jenkins 32
3.2.1. Agentes 34

3.2.2. Pasos 34
3.2.3. Nodos 35
3.2.4. Etapas 35
3.2.5. Notificaciones 36

4. DEVOPS Y MICROSERVICIOS 37
4.1. Puerta de enlace 38
4.2. Escalabilidad 38
4.3. Manejo de errores 38
4.4. Tecnologías 39
4.5. Administración de código 39
4.6. Fácil administración de equipos 39
4.7. Agilidad 39
4.8. Calidad y gobierno 40
4.9. Redes 40

5. CASO DE ESTUDIO Y SOLUCIÓN PROPUESTA 41
5.1. Descripción del caso de estudio 41
5.2. Solución propuesta 42
5.3. Análisis de aplicabilidad de un entorno Devops al problema propuesto 42
5.4. Justificación de arquitectura microservicios 45
5.5. Arquitectura propuesta 47
5.6. Herramientas e infraestructura del entorno Devops 49
5.6.1. Costos de herramientas 49
5.6.2. Compatibilidad 50
5.6.3. Creación de flujos automatizados 50
5.6.4. Alojamiento 51
5.6.5. Evaluación manejo de repositorio 51

5.6.6. Metodología de desarrollo 52
5.7. Análisis ambientes de ejecución de código 53
5.8. Herramientas de despliegue y entrega continua 54
5.8.1. Kubernetes 54
5.8.2. Docker Swarm 55
5.8.3. Comparativa 55
5.8.3.1. Definición de arquitectura 55
5.8.3.2. Alta disponibilidad 56
5.8.3.3. Administración automática de recursos 56
5.8.3.4. Soporte 56

6. IMPLEMENTACIÓN DEVOPS 59
6.1. Estructura del proyecto 59
6.2. Configuración del ambiente de ejecución 61
6.3. Inicializadores del proceso Devops 63
6.4. Clonación del repositorio de versiones 63
6.5. Pruebas 64
6.6. Construcción y publicación 65
6.7. Despliegues 66
6.8. Ejemplo del flujo Devops 67

CONCLUSIONES 71
RECOMENDACIONES 73
BIBLIOGRAFÍA 75

ÍNDICE DE ILUSTRACIONES

FIGURAS

1. Ciclo básico Devops 2
2. Flujo base de integración continua 13
3. Ejemplo básico de flujo automatizado 27
4. Ejemplo de creación de agente 34
5. Ejemplo de pasos en Jenkins 35
6. Flujo de análisis previo a implementación 44
7. Diagrama de despliegue 47
8. Diagrama de componentes 48
9. Desarrollo basado en truncamiento 53
10. Estructura del proyecto 59
11. Flujo del proceso automatizado 60
12. Configuración del ambiente de ejecución 62
13. Configuración del repositorio de versiones 63
14. Resultados de consola servidor de integración continua 64
15. Configuración y ejecución del ambiente de pruebas 64
16. Construcción y publicación 65
17. Configuración de variables del proyecto 66
18. Despliegue a clúster de prueba y producción 66
19. Configuración de despliegues 67
20. Estado inicial del tablero 68
21. Publicación en entorno de pruebas previo a autorización o rechazo ... 68
22. Rechazo o autorización del cambio 69
23. Publicación en entorno de producción 69

TABLAS

I. Análisis de arquitectura propuesta.....45

GLOSARIO

Automatización	Es el uso de herramientas de tecnología para simplificar los procesos rutinarios, delegando estas tareas a elementos programables.
Contenedor informático	Tecnología que permite empaquetar y aislar sistemas, incluyendo todos los elementos necesarios para su funcionamiento, sin perder características.
Escalabilidad	Habilidad de un sistema a adaptarse al crecimiento o disminución de las solicitudes que procesa, sin perder atributos de calidad.
Integración	Acción de unir o acoplar elementos individuales para que formen parte de un todo.
Repositorio	Espacio destinado para el manejo del mantenimiento, organización y distribución de elementos, ya sea físicos o simbólicos.
Servidor informático	Elemento que expone un servicio capaz de satisfacer las solicitudes realizadas por los solicitantes o también llamados clientes.

Orquestador	Plataforma encargada del despliegue, monitoreo y administración de los contenedores que forman parte de un sistema.
Versionamiento	Elemento encargado de gestionar los distintos cambios que pueden llevarse a cabo en un proyecto informático.

RESUMEN

El término Devops surge como una combinación de dos áreas sobre las cuales se cimentan los sistemas de información, siendo estas: el desarrollo de software y las operaciones realizadas para el mantenimiento; puesta en marcha y monitoreo de los sistemas.

Se busca orientar la cultura de la organización a un enfoque colaborativo para hacer los procesos más eficientes y así entregar un software de mejor calidad; para esto es necesario realizar un análisis de los requerimientos en donde se puedan evidenciar los beneficios que se van a obtener y los recursos necesarios para implementar una solución Devops, para tomar la decisión correcta.

En el presente trabajo se muestra el proceso completo de los diferentes factores que se deben tomar en cuenta; se describe un caso de estudio basado en el requerimiento de una entidad financiera que busca flexibilizar sus sistemas de análisis y presentación de información.

De manera que se define la posibilidad de orientar la solución a un entorno Devops, donde se destaca la característica de habilitar la opción de utilizar múltiples herramientas mediante el uso de tecnologías basadas en contenedores, pero esto no satisface las necesidades completamente; es necesario acoplar el sistema para centralizar el gobierno y manejo de estas tecnologías; por tal motivo se integra como parte de la solución un orquestador de contenedores, el cual tendrá el rol de verificar el estado y correcto funcionamiento de cada elemento a lo individual.

Para lograr una implementación donde se puedan obtener los beneficios más destacables de una solución Devops, es necesario apoyarse de herramientas como un repositorio de versiones de código, imágenes de contenedores, servidores de integración continua y prácticas de desarrollo ágil. En el presente trabajo se muestra la arquitectura global de la solución propuesta y cómo interactúa cada elemento con el sistema en general.

OBJETIVOS

General

Realizar un análisis de los beneficios que se obtienen al aplicar técnicas de automatización en el desarrollo de software a través de un caso de estudio, identificando los diferentes aspectos que se deben de tomar en cuenta en una implementación de automatización de procesos.

Específicos

1. Proponer una infraestructura Devops para el caso de estudio enfocado a un ambiente web.

2. Implementar una infraestructura que utilice buenas prácticas en la automatización de procesos.

3. Hacer un análisis de un conjunto de herramientas que permitan completar un ciclo Devops.

INTRODUCCIÓN

La escalabilidad de los sistemas de software tiende a convertirse en un tema burocrático, lento y complejo para la administración, mantenimiento y configuración; esto puede generar lentitud en los tiempos de ciertas tareas, por lo que surgen prácticas como Devops, que expresan un cambio de cultura y la forma en la que personas con múltiples disciplinas trabajan en equipo para el logro del objetivo principal: hacer eficiente la forma en que se desarrollan los proyectos de software.

Dichas prácticas pueden ser difíciles de implementar para organizaciones o equipos de trabajo que no están preparados para cambios drásticos, como los que implican un cambio de cultura y herramientas; por tal motivo es necesario realizar un análisis de impacto y beneficios previo a constatar si la organización de verdad necesita una implementación Devops, así como realizar el análisis si cuenta con los recursos necesarios para poner en marcha una infraestructura de este tipo.

Es necesario conocer en un principio las arquitecturas de software con las que se pueden conseguir mejores beneficios para mejorar la administración de los sistemas. Las herramientas compatibles y soportadas deben ser analizadas para evitar problemas en las implementaciones.

.

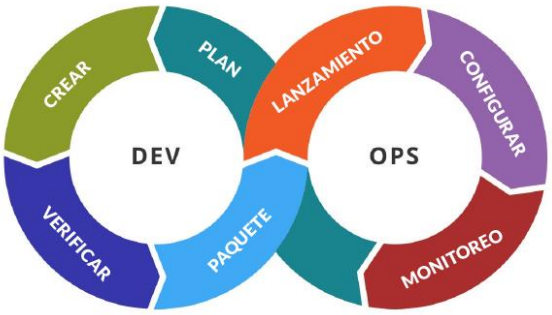
1. DEVOPS

1.1. ¿Qué es Devops?

Devops pretende la unificación y colaboración de personas de distintas ramas de las tecnologías de la información, siendo las principales interesadas las que integran equipos de desarrollo de software y operaciones de TI; la integración de equipos de seguridad e ingeniería de calidad proveen un enfoque mucho más alto, en donde se busca cerrar brechas entre ellos, para lograr agilizar los distintos procesos que van desde el desarrollo hasta el despliegue en producción, pasando por los ambientes de calidad de software, procurando la máxima agilidad posible.

No solo implica un cambio en los procesos; si se busca aprovechar al máximo las ventajas que provee el concepto de Devops es necesario buscar arquitecturas de software que se adapten mejor a los cambios constantes de código y así lograr no solo producir más rápido, sino garantizar la fiabilidad del producto que se le entrega al cliente.

Figura 1. **Ciclo básico Devops**



Fuente: Devops. *Evaluando software*. https://www.evaluandosoftware.com/que-es-Devops/. Consulta: mayo de 2020.

1.2. Orígenes de Devops

A lo largo del tiempo existieron diferentes corrientes de pensamiento que prepararon el camino hasta lo que hoy se conoce como Devops; Tecnologías, metodologías y formas de trabajo influyeron y pusieron las bases para formar el concepto que hoy se utiliza.

1.2.1. Modelo cascada

En la publicación de 1970 de Wiston Royce respecto del tema: 'Administrando el desarrollo de grandes sistemas de software', expone una metodología que consta de 7 pasos secuenciales: requerimientos de sistema, requerimientos de software, análisis, diseño del programa, programación pruebas y operación, que aunque en la publicación no se menciona el concepto de cascada, este significaría la base que precedería al conocido modelo; es interesante que en el mismo documento se hace un análisis de los

inconvenientes en el flujo de trabajo de esta metodología, debido a que la retroalimentación del proceso de pruebas se realiza al final del proyecto.

Para remediar esta situación el autor propone el concepto de un diseño preliminar que se conoce como prototipo; esto para la obtención de retro alimentación temprana.

1.2.2. Desarrollo de software ágil

Se define como un conjunto de prácticas que propone cómo gestionar proyectos bajo la filosofía del desarrollo iterativo e incremental, en el cual se busca que el producto final sea analizado, probado y mejorado durante su desarrollo, con un total involucramiento por parte del cliente; de esta manera se tendrá la perspectiva de lo que necesita; la forma de trabajo se basa en entregas continuas totalmente funcionales para el cliente, fomentando la comunicación entre los interesados mediante reuniones periódicas; esto hace posible que durante el desarrollo del proyecto se puedan identificar y realizar cambios no previstos en el análisis inicial, de tal manera que se puedan entregar las características que el negocio valore más.

Las alternativas que promueven la agilidad fueron fuertemente influenciadas por corrientes como Manufactura esbelta (*Lean manufacturing)* originada en las fábricas de Toyota en los años 40’s; la cual propone la reducción de desperdicios y el concepto de justo a tiempo (*just in time)* que significa producción a medida para disminuir los costos de almacenamiento; de tal manera que se fueron madurando ideas más sólidas con enfoques puramente al desarrollo de software como: programación en parejas (*programming pairs*) y scrum en 1995; desarrollo guiado por funcionalidades (*feature driven development*) en 1997, programación extrema (*extreme*

programming) en 1999; manifiesto ágil en 2001, desarrollo guiado por pruebas (*test driven development*) en 2002, entre algunas otras metodologías que propusieron una alternativa sobre cómo gestionar proyectos de software.

1.2.3. Orígenes del concepto Devops

En primera instancia se buscó aplicar conceptos de agilidad y manufactura esbelta a las tecnologías de la información, en donde se enfocaba más a la gestión de proyectos, específicamente en el desarrollo de software; de tal manera que la industria se comenzó a inclinar a este tipo de administración, logrando así grandes resultados; pero este enfoque deja una brecha qué cerrar en la cual se trata de incluir la administración de sistemas a la agilidad. Esto fue lo que planteó Andrew Shafer cofundador de Puppet, una empresa de automatización de tecnología; en el 2008 ofreció una conferencia en Toronto llamada “Infraestructura ágil”, en la cual solamente un administrador de sistemas y consultor del gobierno de Bélgica llamado Patrick Debois se enlistó para asistir.

Patrick Debois como ingeniero administrador de sistemas del gobierno de Bélgica se sentía frustrado, ya que tenía la tarea de migrar un centro de datos gubernamental en el cual existía dificultad colaborativa entre los diferentes equipos de trabajo, por lo cual se veía interesado en temas de agilización del trabajo realizado en la administración de infraestructura de sistemas, pero se enfrentó con el hecho que, debido a que solamente una persona había sido confirmada a la conferencia de Andrew Shafer, esta no se llevó a cabo; pero sí existió un acercamiento entre Patrick Debois y Andrew Shafer, del cual surgió un grupo llamado “Administración de sistemas ágiles” en donde compartían ideas sobre cómo facilitar las diferentes tareas de la administración de

sistemas; estos grupos marcarían el inicio de foros, de donde nacería y maduraría el concepto de Devops.

En la conferencia O'Reilly Velocity 09, dos personajes: John Allspaw y Paul Hammond ofrecieron una plática llamada "10+ despliegues por día: Desarrollo y operaciones cooperando" a la cual Patrick Debois lamentó no poder asistir, pero surgió la pregunta de por qué él no realizaba conferencias de la misma índole en Bélgica, en donde en octubre de 2009 se consolidó la idea y se llevó a cabo la conferencia, en ese entonces bautizada conjugando letras de las palabras *development*, *operations* y *days*. En ese entonces una vez terminada la conferencia se popularizó en *twitter* la palabra Devops como concepto de colaboración entre personas de desarrollo y administradores de sistemas.

1.3. CALMS

Es un marco de trabajo que provee una perspectiva y evaluación del nivel de progreso de una organización o equipo en relación con criterios referentes a Devops; establece un punto de referencia inicial y proporciona una ruta con los pasos para alcanzar los objetivos deseados. Los pilares del marco CALMS son: cultura, automatización, limpieza, medición y comunicación.

1.3.1. Cultura

Se define como la forma en la que se realizan ciertas actividades por parte de una comunidad humana, tomando en cuenta el entorno en el que se desenvuelve, la forma en la que razona y comunica, así como los valores que rigen al grupo de personas; este concepto es importante, ya que implica cómo

los equipos afrontan y buscan conjuntamente la solución a problemas influenciados por el entorno en el que fueron instruidos.

- División del trabajo: en la actualidad el ser humano se ve en la necesidad de estar rodeado de información y requiere sistemas cada vez más integrados, incluso entre organizaciones; esto produce la comunicación entre equipos, por ejemplo: el equipo de desarrollo provee el software que se va a implementar en determinada infraestructura, en donde se deben tomar en cuenta los protocolos de comunicación, direcciones o recursos en donde se necesitará acceder, entre otros elementos. Estas interacciones implican la necesidad de aplicar el pensamiento sistémico que está orientado a examinar el funcionamiento, la interacción entre los sistemas, la influencia de uno con otro y cómo logran el fin común. Visualizar los sistemas desde esta perspectiva permite solucionar problemas que no se pueden solucionar de manera aislada, sino que requieren diferentes perspectivas.

- Revolución digital: el término Devops es conocido por cumplir con las características del concepto de la revolución digital, que se define como el proceso de usar tecnologías digitales para crear o modificar la forma en la que operan las organizaciones; esto exige la capacidad de los equipos de adaptarse al cambio constante, aprendizaje continuo, nuevas formas de trabajar, así como abandonar zonas de confort, esto para aumentar la eficiencia operativa y la satisfacción del cliente.

1.3.2. Automatización

Es un proceso en el cual se confía en la tecnología una tarea que se realiza de manera manual, para optimizar el tiempo del recurso humano,

asegurar la constante retroalimentación para solventar posibles fallas en los sistemas y reducir el tiempo en el que se entrega el producto al cliente.

En la industria de las tecnologías de la información, en donde el cambio es contante y rápido, lo que hoy se considera novedoso en tiempos cortos puede llegar a ser obsoleto; el concepto de tiempo a mercado (*time to market*) se define como el lapso entre el momento que un producto o servicio es concebido hasta que se coloca en producción; para industrias altamente competitivas como la de los teléfonos inteligentes es fundamental ofrecer al usuario final lo último en tecnología en el menor tiempo posible y sobre todo antes que la competencia; desde este aspecto competitivo, se debe reducir el uso de tiempo en tareas, las cuales pueden confiarse en manos de la tecnología. Esta es una importante oportunidad.

Es importante estimar las consecuencias de acelerar los procesos, de tal forma que se pueda entender qué es lo que necesita hacer más rápido, cómo funciona y cuál es el impacto en el sistema en general, ya que el desconocimiento podría provocar que los procesos mal realizados solo se estén llevando a cabo más rápido, y ante posibles fallas o problemas, se tenga una capa de complejidad más en la cual preocuparse.

1.3.3. Limpieza

El constante cambio, la competitividad y las necesidades de información exigen que se produzca más rápido, a menor costo, sin perder la calidad; por lo cual es necesario adoptar nuevas filosofías como tecnologías de la información esbeltas que consisten en principios nacidos de las fábricas de Toyota conocidos como Lean Manufacturing, basados en la mejora continua de los procesos mediante la eliminación de actividades que no agregan valor a

negocio; esto provoca la optimización del uso de los recursos, utilizando solamente lo necesario e indispensable.

Como punto de partida se requiere el conocer a fondo las actividades que conlleva el producto final; esto es conocido como cadena de valor. Para hacer una representación visual del proceso y se pueda identificar claramente qué actividades no agregan valor al producto final para su posterior optimización, una vez realizado este análisis, se deben identificar los posibles cuellos de botella en el proceso, para así disminuir los tiempos de espera; según la filosofía *lean* o esbelta este proceso se lleva a cabo tantas veces como sea necesario, hasta que llegue al punto en el que se realicen actividades puramente de valor para el cliente.

1.3.4. Medición

Cuando se tienen entornos basados en la automatización de procesos es necesario dar un enfoque diferente a la forma en la que se cuantifica la producción, eficiencia, eficacia, entre otros, de las actividades involucradas, no solo para medir e identificar cómo se desempeña un proceso, sino también la calidad con la que se lleva a cabo; se deben establecer métricas válidas que representen adecuadamente los objetivos aceptables para cada proceso; una vez que se han determinado los elementos de medición, se deben plasmar en herramientas de visualización para el monitoreo del sistema.

Las métricas de estado de infraestructura, representan el comportamiento de un conjunto de elementos que interactuarán entre sí para ofrecer un servicio; por ejemplo, métricas como el apdex (*application performance index*) que indican el nivel de satisfacción del usuario final, estableciendo fronteras en las que se tienen diferentes reacciones, establece intervalos en los que el cliente

puede trabajar satisfactoriamente en un sistema tolerable que indica que el usuario se ve afectado, pero no lo suficiente para no desempeñar la tarea, y frustrado en donde corra el riesgo de que no siga con las actividades planeadas.

Otra métrica importante es cuánto tiempo le lleva a un sistema procesar una solicitud y dar respuesta al cliente; este tipo de métricas se ve afectado por factores como el porcentaje de uso de memoria y recursos de procesamiento, lectura y escritura a elementos de almacenamiento y velocidad de comunicación de redes; yanto estas como otras métricas describen la salud de un sistema, enfocado a la forma en que una infraestructura se comporta en diferentes entornos.

Las mediciones de confiabilidad del sistema determinan su disponibilidad y cómo se comportan en determinadas situaciones. Los indicadores de tiempo medio entre fallas y tiempo medio de reparación denotan el lapso transcurrido desde que ocurre una falla y la siguiente, y el tiempo que implica la reparación luego de una falla; estos indicadores ayudan a establecer el impacto de una incidencia en los servicios ofrecidos y determinan los niveles de tolerancia o rechazo, métricas que también se reflejan por ejemplo en los SLA (*Service Level Agreemet*) que es un documento en el que se acuerdan los compromisos de calidad que un proveedor ofrecerá, con el afán de establecer objetivos claros, tanto al proveedor como al cliente, en cuanto al funcionamiento de los sistemas.

La medición de los equipos de desarrollo se ve influenciada por el tipo de metodología adoptada por los equipos de trabajo; dan visibilidad de aspectos como: salud del proceso, trabajo en proceso, tareas en espera, tiempos de espera y velocidades de desarrollo; se enfocan en la forma en la que se

organiza el trabajo; también existen métricas un poco más técnicas relacionadas con el ciclo Devops, como por ejemplo tiempo y frecuencias de despliegue, de tal forma que indiquen los resultados del flujo Devops.

Dentro del entorno de las métricas una de las más importantes es el control de calidad, que incluye pruebas estáticas del código donde no es necesaria la compilación del código y se analizan los estándares de codificación utilizados, complejidad y duplicidad de código, documentación y análisis dinámicos del código donde se necesita compilarlo e incluye retroalimentar resultados de las pruebas funcionales y no funcionales.

1.3.5. Comunicación

La comunicación y colaboración entre equipos es la base en la que se opera Devops, desde el momento en el que se analiza una solución; los diferentes procesos que se deben implementar en el flujo Devops necesitan conformar equipos multidisciplinarios que colaboren para alcanzar los objetivos. Para que la comunicación pueda ser efectiva es necesario el entendimiento y comprensión desde una perspectiva más amplia y considerar el sistema como un todo.

1.4. Ciclo de vida Devops

Es un ciclo iterativo e incremental que consiste en organizar y optimizar la forma de cómo se llevan nuevas funcionalidades a los ambientes de producción de la manera más rápida y eficiente posible, evitando los fallos de seguridad.

1.4.1. Desarrollo

En la fase de desarrollo es importante destacar como primer punto la adopción de alguna metodología ágil, que se acomode a los objetivos del proyecto y los principios Devops, metodologías como: Scrum, Kamban, Scrumban y programación extrema, les dan a los equipos la capacidad de responder a necesidades de un mercado que necesita respuestas rápidas con la capacidad de adoptarse al constante cambio, dándole al cliente el empoderamiento del producto. Como segundo punto, la adopción de prácticas que doten al desarrollador un entorno Devops:

- Diseño iterativo e incremental: independiente de la metodología ágil adoptada, es necesaria la división en pequeñas tareas para garantizar las entregas frecuentes al cliente.

- Propiedad colectiva del código: la pieza fundamental del entorno Devops donde se requiere una correcta organización del código mediante repositorios de versiones.

- Desarrollo dirigido por pruebas: se escriben pruebas que fallan; a continuación, se implementa el código y se verifica que dichas pruebas pasen satisfactoriamente para después refactorizar el código.

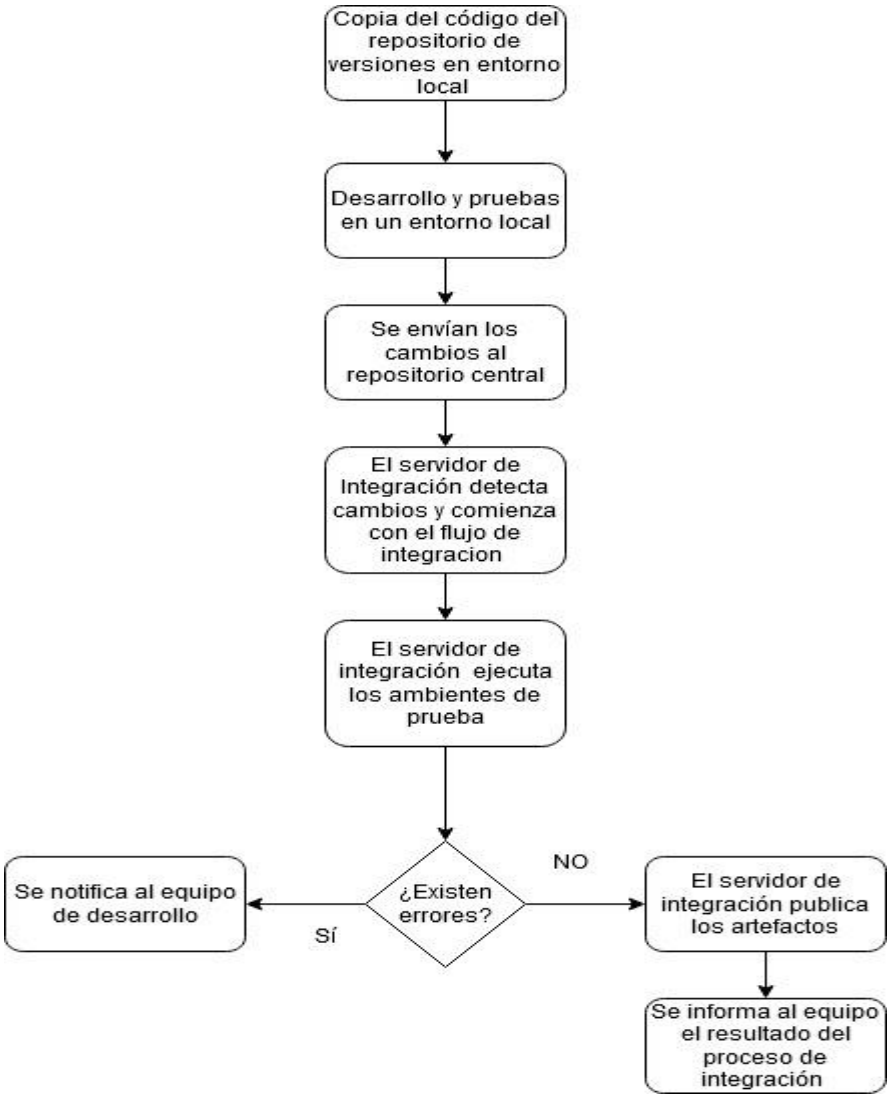
- Estándares de codificación: consiste en un conjunto de convenciones que dependen del lenguaje establecido y describen el formato para escribir el código.

- Ritmo sostenible: busca que el ritmo de producción sea sostenible y se mantenga la constante entrega al cliente.

1.4.2. Integración continua

Consiste en una práctica que busca la unificación constante de los cambios del código para que los equipos puedan obtener retroalimentación rápida del trabajo realizado, y se dé la detección temprana de errores, buscando entregar código de calidad al cliente y optimizando el tiempo invertido en tareas repetitivas.

Figura 2. **Flujo base de integración continua**



Fuente: elaboración propia, empleando el programa DRAW.IO 13.8.8.

1.4.3. Entrega y despliegue

La diferencia entre estos dos conceptos radica en la forma en que se entrega el producto final al cliente, en cuanto a entrega continua se refiere, a colocar los artefactos probados en ambientes no productivos, en donde se pueda hacer una verificación y autorización final previo a colocar en producción el sistema; el despliegue continuo se caracteriza por poner en producción los artefactos sin intervención humana.

2. PRÁCTICAS Y HERRAMIENTAS EN EL CICLO DE VIDA DEVOPS

2.1. Infraestructura de desarrollo

Para una correcta implementación de un ambiente Devops, desde el principio se mencionó que es necesario que haya cambios de cultura, así como adoptar nuevas prácticas que sean compatibles con los principios de una cultura de automatización, por lo cual es conveniente la organización de la metodología de trabajo orientada a la agilidad y al hardware donde se ejecutarán las aplicaciones, con infraestructuras que puedan ser gestionadas mediante código, si el problema lo requiere. También es necesario buscar y adoptar algunas prácticas de desarrollo que a continuación se mencionan.

2.1.1. Versionamiento de código

En la gestión de desarrollo de software, cuando se tornan los proyectos cada vez más grandes, es necesario un control de las liberaciones que se hacen al cliente; esto debido a dependencias de los sistemas y compatibilidad entre las diferentes entregas, la adopción de estándares de versionamiento expresan y dan nombre a las características agrupadas de un sistema.

2.1.2. Versionamiento sintáctico

Consiste en un conjunto de reglas que indican cómo y por qué asignar una etiqueta a una versión de software; se caracteriza por ser simple y entendible, ya que se basa principalmente en la utilización de 3 números separados por

punto X.Y.Z, denominados estos números como mayor, menor y parche. Parche indica arreglos cuando no se ve afectada la compatibilidad del sistema; menor cuando solo se agregan funcionalidades y mayor cuando existen cambios substanciales donde la compatibilidad se ve comprometida.

- Los incrementos deben realizarse utilizando números no negativos y en incrementos de 1.

- El versionamiento inicialmente debe empezar con numeración mayor a 0.

- Las versiones parche definen la corrección de errores que corresponden a una versión menor y deben ser incrementados, según vayan sucediendo.

- La versión menor es utilizada para registrar funcionalidades nuevas en el sistema y se deben restablecer en 0 las versiones parche.

- La versión mayor es utilizada para registrar cambios en donde se compromete la compatibilidad con versiones anteriores; se establece que las versiones “menor” y “parche” deben reiniciarse a 0.

2.1.3. Manejo de repositorios de software e integración a Devops

La forma en la que se administra el código en los proyectos de software desde el almacenamiento hasta el manejo de versiones implica un paso importante en el diseño de una solución basada en Devops, ya que es necesario tomar en cuenta la distribución del proyecto entre los equipos de

desarrollo, la cantidad de personas involucradas directamente en la producción de código, políticas de calidad de código, requerimientos, entre otros.

Estos factores son importantes, ya que es necesario establecer un estándar en el repositorio que se acople a los principios Devops dentro de los cuales destaca la rapidez con la que se entrega el producto al cliente y la rápida retroalimentación a los equipos de programación; asumiendo que se estableció una metodología ágil orientada a entregas continuas se puedan encontrar diferentes estilos de desarrollo para el manejo de repositorios, cada uno con características enfocadas en diferentes necesidades.

2.1.3.1. Git Flow

Uno de los estilos de administración de repositorios más conocidos y caracterizado por la cantidad de ramas utilizadas y las diferentes políticas necesarias para que un cambio pueda desplegar en producción (*master*).

- Máster y desarrollo (*develop*) juegan un papel importante, ya que en cierto momento todo el código pasa por alguna de estas ramas y estas no deben ser modificadas directamente.
- Desarrollo (*develop*) juega un rol en el cual se busca el almacenamiento de todos los cambios que serán incluidos en la siguiente versión.
- Las funcionalidades se manejan en ramas llamadas “características” (*feature*), de manera separada y toman como fuente el contenido de la rama desarrollo.
- Para el manejo de errores filtrados en producción existe una rama llamada “corrección en caliente” (*hotfix*) la cual se origina en la rama master donde se realizan las modificaciones necesarias para integrar de nuevo en la rama de producción.

- Para realizar un despliegue en producción (desarrollo a master) existe el concepto de ramas “lanzamiento” (*release*) cuyo papel es análogo a preproducción, donde se realizan diferentes pruebas para verificar si es confiable pasar a producción; de encontrarse errores los cambios se realizan sobre esta rama.

2.1.3.2. Desarrollo basado en truncamiento

Se caracteriza por estar enfocado en entornos ágiles en donde se busca la simplicidad, reduciendo la burocracia y el número de ramas utilizadas en el repositorio, buscando realizar cambios lo más cortos posibles, para luego integrar nuevamente en la rama de producción.

- Existe una rama base llamada truncamiento (*trunk*) donde los desarrolladores agregan el código directamente.
- En la rama truncamiento se realiza la integración continua directamente.
- En ocasiones se crean ramas para agregar funcionalidades, tomando como base la rama truncamiento y procurando que estas sean lo más cortas posibles.
- La integración y despliegue se realiza después de realizar la unión con la rama truncamiento.

2.1.3.3. GitLab Flow

Esta forma de administrar el repositorio busca la adaptabilidad con entornos de despliegue y entrega continua, pero tratando de reducir errores en producción por medio del manejo de ambientes donde se pueda verificar la calidad del código y hasta ese momento pasar a ambientes productivos.

- Utiliza ambientes donde se integra el código para asegurar la calidad, nombrando las ramas como preproducción y producción.
- Se utilizan ramas para agregar funcionalidades; no se realizan directamente sobre la rama máster.
- Busca realizar pruebas de calidad del sistema en cada confirmación, no solamente sobre ramas específicas.
- Todas las funcionalidades inician en la rama máster.

2.1.3.4. Github Flow

Este método trabaja sobre la filosofía en donde se busca que en cada rama creada para desarrollo de una funcionalidad también se prepare un ambiente de despliegue, donde se pueda verificar la calidad del sistema, entregado a producción.

- No se realizan despliegues sobre la rama master.
- Los despliegues se realizan en las ramas donde se desarrollan las funcionalidades
- Los despliegues en las ramas de desarrollo se realizan cuando se ejecutan solicitudes de extracción.
- Después de garantizar la calidad del código en ramas de desarrollo, solo hasta ese momento se pueden realizar integraciones con la rama máster.
- El ambiente de producción siempre será la rama máster; esto quiere decir que inmediatamente después de hacer una integración a rama master, esta se debe desplegar.

2.1.4. Herramientas de manejo de código fuente

Los sistemas de control de versiones apoyan con la tarea de registrar todos los cambios de un documento o archivo, por lo que se puede recuperar la versión más antigua trabajada de cualquier documento.

2.1.4.1. Github

Es una plataforma que provee servicio distribuido de control de versiones, funcionalidades de manejo de código fuente, así como la capacidad de integración con diferentes tecnologías en el ámbito de desarrollo de software, ofreciendo fácil personalización de los flujos de trabajo; una de las características más importantes es que provee un repositorio básico sobre el cual existe gran cantidad de aplicaciones especializadas compatibles.

- Calidad de código: gracias a la gran cantidad de aplicaciones externas que usan como base la plataforma Github, es posible encontrar diversidad de opciones para automatizar las tareas de calidad de código como análisis estático, estándares de código, pruebas de cobertura de código, duplicación de código y complejidad, entre otras pruebas de calidad para cada confirmación; tiene la capacidad de analizar directamente el código en el repositorio, identificar problemas y notificar acerca de estos, proporcionando interfaces entendibles acerca del estado del proyecto, alguna de las aplicaciones más relevantes compatibles con Github son: Codacy, Coderefactor, Sonacube.

- Integración continua: otro aspecto que se debe tomar en cuenta es la compatibilidad del repositorio con herramientas de integración continua, las cuales realizarán las tareas de compilación y pruebas en ambientes

no productivos de manera automatizada. Github ofrece integración con tecnologías basadas completamente en la nube, así como herramientas híbridas de manera local como por ejemplo Travis CI, CircleCI, Codefresh y Azure Pepilines para tecnologías nativas de la nube; así como servidores de integración, los cuales se pueden montar de manera local como Jenkis.

2.1.4.2. GitLab

Provee un servicio enfocado en ofrecer una solución integrada de todo el ciclo de vida de software que abarca desde el manejo y administración de proyectos por medio de herramientas como Jira, Wikis, manejadores de versiones, tableros Kanban, hasta herramientas nativas para implementación del ciclo de vida Devops y acoplamiento para realizar despliegues en las diferentes nubes del mercado; todas estas características disponibles y centralizadas en una plataforma buscan la simplicidad en tareas como revisión de conflictos de código, despliegue del estado de los flujos de integración y entrega continua.

2.2. Prueba continua

Proceso para validar el producto desarrollado con base en los requerimientos funcionales y no funcionales que se han establecido; esta fase provee una retroalimentación rápida de posibles fallas en el sistema donde se define un conjunto de configuraciones preparadas en un entorno para ejecutar casos de prueba.

2.2.1. Pruebas funcionales

- Pruebas unitarias: son pequeñas porciones de código encargadas de verificar el comportamiento de funciones específicas de dicho código.
- Pruebas de integración: consisten en realizar pruebas de los elementos unitarios que conforman el sistema y cómo interactúan entre sí.
- Pruebas de sistema: están catalogadas como pruebas de caja negra y buscan verificar el cumplimiento de los requerimientos establecidos.
- Pruebas de interfaz: proceso en que se valida la comunicación entre dos sistemas diferentes.
- Pruebas de regresión: son encargadas de corroborar el comportamiento de funciones del sistema después de algún cambio en el mismo.
- Pruebas Alpha Beta: estas buscan posibles errores en los sistemas, utilizando el método de caja negra y simulando usuarios reales.

2.2.2. Pruebas no funcionales

- Pruebas de rendimiento: estas verifican cuantitativamente el comportamiento del sistema en determinados escenarios; pueden incluir métricas como: velocidad de procesamiento, transferencia de datos, rendimiento, estado de memoria y tiempos de respuesta.
- Pruebas de carga: estas pruebas se diferencian con las de rendimiento que se efectúan en entornos donde se simulan escenarios en condiciones a los que serán expuestos en la vida real.
- Pruebas de estrés: estas buscan comprobar la estabilidad, robustez y fiabilidad del sistema ante casos de pruebas exigentes en relación con cantidad de solicitudes por satisfacer.
- Pruebas de volumen: verifican el comportamiento y tiempo de respuesta de los sistemas cuando se ven sometidos a volúmenes altos de datos;

estos se enfocan en verificar la optimización que se tiene en la base de datos.

2.2.3. Creación de ambientes de calidad de software

En el entorno de las metodologías ágiles integradas con Devops, el principal objetivo es la entrega continua al cliente; esta premura por la entrega de código puede provocar vulnerabilidades o aumentar la probabilidad de que existan problemas en ambientes de producción afectando directamente a usuarios finales, por lo cual en entornos de desarrollo maduros comúnmente existen diferentes ambientes donde se verifica la calidad del producto entregado.

En estos ambientes se realizan las configuraciones necesarias para ejecutar el sistema en un entorno parecido o igual al que los usuarios estarán consultando; aunque muchas veces las prácticas de desarrollo dirigido por pruebas aseguran la calidad de código desde el desarrollo, pero esto no es suficiente, ya que es común que las pruebas unitarias sean escritas por los mismos programadores; esto causa que se vea el sistema desde perspectivas reducidas.

Estos ambientes deben operar con conjuntos de datos que emulen el funcionamiento al cual estarán expuestos, por lo cual se tienen las opciones de generar información ficticia de prueba o replicar y extraer información que se tiene en producción cuando sea posible; el principal problema de utilizar datos ficticios es que se realizaran pruebas desde perspectivas limitadas.

2.3. Ambiente automatizado para la ejecución de pruebas y retroalimentación continua

En el análisis previo a establecer una solución Devops, es necesario definir las herramientas que serán utilizadas para una tarea tan importante como asegurar la calidad del producto de software; estas deben ser adaptables con entornos automatizados; se listarán algunas opciones disponibles para análisis estático y dinámico de código.

2.3.1. Análisis estático de código

Es un análisis enfocado en evaluar el código fuente sin ejecutarlo y que dé una perspectiva en forma de métricas para optimizar la compilación o interpretado del código; estas herramientas proporcionan una vista automática y profunda buscando defectos como: encontrar código que no es usado en la ejecución, código repetido, complejidad, mantenibilidad, entre otros tipos de análisis; las herramientas que pueden servir de apoyo para estas tareas son:

- SonarQube: Java, JavaScript, PHP, PL/SQL, VB, Python
- Coverity: C, C++, C#, JavaScript, Ruby, Php y Python.
- Veracode: C#, VB, Java, C/C++, JavaScript, Python, Ruby
- Gamma: C, C++, C#, Java, JavaScript, Python, PHP, Go

2.3.2. Análisis dinámico de código

A diferencia del análisis estático, en este es necesaria la ejecución del código fuente, que evalúa el comportamiento en condiciones controladas y situaciones extremas en las que estará sometido el sistema en ambientes de

producción; estas son usadas para validar los requerimientos, tanto funcionales como no funcionales; algunas de las herramientas utilizadas son:

- TestComplete
- Source Labs
- Jmetter
- PracticTest
- Selenium
-

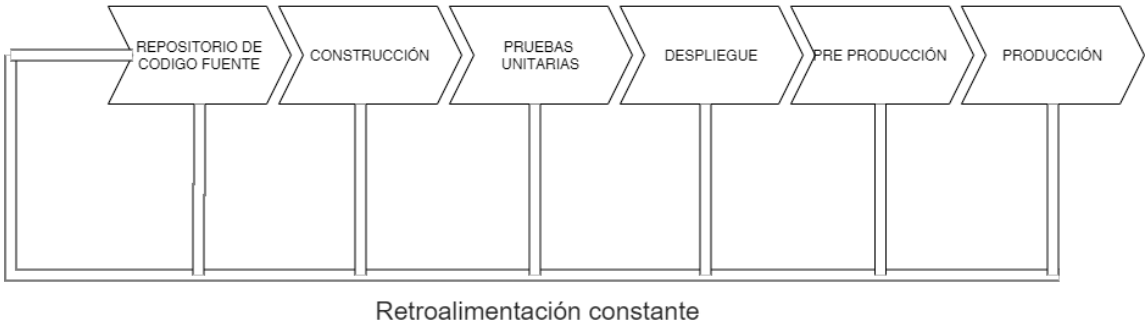
2.4. Diseño del flujo de automatización

Consiste en la creación de un mapeo en donde se plasman los diferentes pasos necesarios para llevar el código fuente a ser desplegado; en esta parte del diseño de un flujo Devops es necesario auxiliarse de herramientas que sean capaces de proporcionar funcionalidades que agilicen procesos repetitivos como por ejemplo: compilaciones, revisiones de estándares de programación, ejecución de diferentes pruebas de calidad de código, entre otras tareas; estas herramientas son comúnmente conocidas como servidores de integración, los cuales deben poseer las siguientes características básicas:

- Flujos: son un conjunto de pasos y acciones que se ejecutan en determinado momento, caracterizados por agrupar diferentes acciones llamadas etapas.
- Pasos: representan la mínima acción ejecutada por la herramienta de integración; las tareas pueden ser desde clonar un repositorio hasta desplegar en producción.
- Etapa: consiste en agrupar diferentes pasos secuencialmente distribuidos; algunas etapas utilizadas podrían ser: compilación, clonado de repositorio, instalación de dependencias o despliegue.

- Disparadores: definen las condiciones necesarias para iniciar un flujo Devops.
- Notificadores: a lo largo de todo el procedimiento puede que se den diferentes sucesos, los cuales se deben informar a los equipos de desarrollo; algunos de los estados utilizados pueden ser: cambio, fallo, satisfacción, abortado o inestable.

Figura 3. **Ejemplo básico de flujo automatizado**



Fuente: elaboración propia, empleando el programa DRAW.IO 13.8.8.

La creación y el diseño de un flujo Devops depende puramente de las necesidades del desarrollo; es decir estos flujos pueden extenderse considerablemente cuando es necesario agregar ambientes con otros enfoques; por ejemplo: prueba unitaria, calidad de código, pruebas de aceptación, preproducción, producción, así como la creación de ambientes personalizados para tareas específicas; a continuación, se explica una implementación básica de un sistema web.

- Repositorio de código fuente: este paso comúnmente es iniciado por alguna acción como por ejemplo una confirmación; algunas de las acciones básicas que se realizan en los pasos iniciales son: clonar la rama configurada en el servidor de integración y configuración del directorio de archivos donde se realizarán las diferentes tareas posteriores.

- Construcción: esta etapa depende de las tecnologías utilizadas, pero comúnmente se aplican herramientas de compilación como: Graddle o

Maven, las cuales entregan artefactos resultantes del código fuente; luego serán entregados a los distintos ambientes de calidad de software.

- Pruebas unitarias: consisten en ejecutar las pruebas unitarias realizadas por los desarrolladores, en donde se garantiza en primera instancia la calidad del software entregado.

- Despliegue: en esta parte del ciclo se realizan las tareas de publicación de los artefactos resultantes de la compilación en entornos como un servidor web.

- Preproducción: consiste en un ambiente que contiene las mismas características y configuraciones del servidor de producción en donde se realizan las pruebas no funcionales relacionadas con rendimiento y tiempos de respuesta.

- Despliegue en producción: en el caso en que los pasos anteriores fueran satisfactorios, se puede proceder a la publicación de los cambios en el ambiente productivo.

3. DEVOPS BASADO EN DOCKER Y JENKINS

3.1. Docker

Su función principal es empaquetar código con todas sus dependencias, para su rápido despliegue en diferentes ambientes y plataformas; estos corren sobre el mismo kernel de sistema operativo como procesos del sistema, en comparación con máquinas virtuales que también pueden ejecutar varias instancias sobre una misma máquina, con la diferencia que cada máquina virtual necesita los recursos para emular el sistema operativo real.

Docker necesita una forma *standard* para empaquetar las aplicaciones y realizar las configuraciones necesarias en un contenedor para su correcto funcionamiento; también necesita de una interfaz centralizada que pueda escuchar las solicitudes realizadas por los elementos del ambiente Docker; esta es la funcionalidad del *docker engine*, que consiste en un servicio dentro de la máquina hospedadora llamado Dockerd, el cual es el encargado de la creación de imágenes, volúmenes, entre otros objetos; un *rest api* es la interfaz entre la línea de comandos y el servicio Docker; la línea de comandos corresponde al elemento con el que el usuario final interactúa y acepta instrucciones como *docker ps, docker images*.

3.1.1. Imágenes

Las imágenes de Docker consisten en una plantilla que contiene las diferentes instrucciones para la construcción de un ambiente especifico; nace con un archivo llamado *dockerfile,* el cual contiene instrucciones como la

imagen base que será utilizada, e instrucciones personalizadas para la creación de nuevas imágenes con funciones y características especiales para las tareas que se realizarán, por ejemplo: crear una nueva imagen utilizando una base con un sistema operativo Ubuntu en el cual se instalará y pondrá en funcionamiento un servidor apache; esta personalización, una vez compilada, es almacenada en una máquina local, pero puede ser almacenada en plataformas como Dockerhub; esto para interactuar con otras herramientas como kubernetes.

3.1.2. Contenedores

Es un ambiente empaquetado con las configuraciones necesarias para realizar tareas específicas como iniciar servicios, compilar y ejecutar programas, entre otros; estos necesitan una imagen base en la cual se deben especificar los recursos a los que tendrá acceso el contenedor, en cuanto a especificaciones de redes, puertos y almacenamiento. En palabras no tan técnicas, se puede definir un contenedor como una instancia donde se especifican sus características iniciales y los elementos con los cuales tendrá interacción.

3.1.3. Redes

Son las aplicaciones empaquetadas y listas para su ejecución en cualquier máquina, las cuales poseen Docker instalado. Esta es una de las ventajas que este tipo de tecnologías ofrece, pero existe la necesidad tanto de comunicación entre contenedores como entre elementos fuera de este entorno, como bases de datos; por tal motivo, existe una capa de abstracción adecuada para estos casos.

Cuando Docker es instalado se crean tres redes por defecto “decker0” sobre la cual cada contenedor nuevo es vinculado si no es especificado lo contrario; “host”: los elementos que están en esta red no se encuentran aislados y pueden ser accedidos a nivel de la máquina hospedadora y “*none*”: en donde los contenedores son vinculados a redes específicas.

Para funcionalidades como comunicación entre contenedores, Docker, asigna un dominio de resolución de nombres a cada uno, donde se puede asociar una ip específica a un nombre para habilitar la comunicación entre los elementos; para la comunicación fuera de la red es necesario exponer y mapear los puertos a los cuales se tendrá acceso y especificar qué puertos de la máquina hospedadora se redirigirán a los contenedores; de esta forma Docker provee funcionalidades básicas y especificas según requiriera el caso, para hacer sistemas agnósticos y portables.

3.1.4. Almacenamiento

El manejo del almacenamiento en Docker no es persistente; es decir este solo existe dentro del contenedor durante el tiempo en el que está en ejecución; el trabajo de extraer datos puede llegar a ser trabajoso, por lo que existen opciones para el manejo de los datos.

En ocasiones en las que se desea compartir información entre contenedores, por ejemplo una aplicación que requiere que varios contenedores accedan a un solo almacén de datos en común como imágenes, código o incluso bases de datos, los recursos se deben encontrar disponibles, incluso si el contenedor se detiene; para esto se puede utilizar el sistema de archivos de la máquina hospedadora, utilizando en la instrucción de creación del contenedor comandos como: “–v <dirección del hospedador>: <dirección del contenedor>”.

Otra de las alternativas es montar un directorio local dentro de un contenedor; para soluciones más específicas existen complementos de Docker para plataformas como Amazon o Google, que proveen almacenamiento persistente.

3.1.5. Repositorio de imágenes

Este es un servicio ofrecido por plataformas como DockerHub, Google o Amazon, que permite por medio de comandos como *docker pull* y *docker push* obtener imágenes base o personalizadas.

3.2. Jenkins

Es conocido como un servidor de integración continua de código abierto, que ofrece de manera sencilla el manejo de procesos repetitivos involucrados en el desarrollo de software, como por ejemplo tareas de compilación y pruebas de calidad de código. Se entiende como integración al conjunto de procesos que van desde el momento en el que se obtiene el código de algún repositorio de versiones con las últimas funcionalidades o errores corregidos, hasta el momento en el que se llevan estos cambios a entornos de producción, en el menor tiempo posible y de forma automática.

Esta herramienta tiene complementos de terceros como repositorios, bases de datos, lenguajes de programación, integración con nubes que proveen y enriquecen las funcionalidades de Jenkins; estos complementos pueden ser instalados de dos formas: mediante la interfaz de usuario donde se pueden buscar directamente los complementos existentes, así como la descarga de los archivos .war directamente de la plataforma pública de Jenkins e instalación en el servidor de integración.

Un procedimiento básico de integración continua puede consistir en los siguientes pasos:

- Obtener el código del repositorio de versiones
- Ejecutar tareas de construcción
- Ejecutar pruebas realizadas al sistema
- Entregar el producto en otros ambientes de prueba
- Desplegar la aplicación en producción

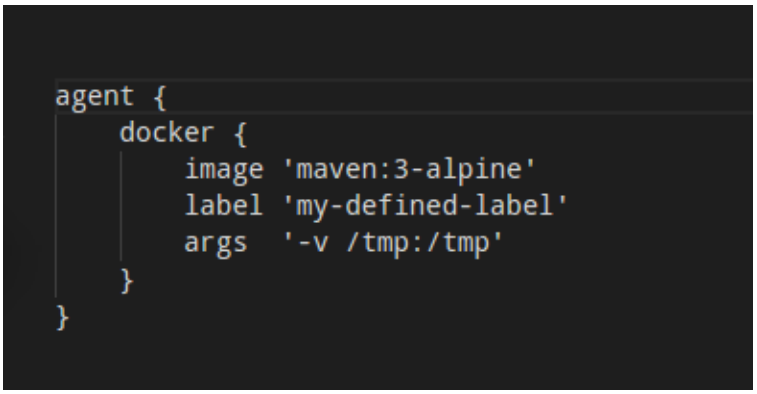
En cuanto a la creación de procesos se tienen dos posibilidades que caracterizan la secuencia de pasos predefinidos; en los que no existen condicionales son llamados flujos estáticos donde la única acción posible de realizar si algo falla es marcar el flujo como error y notificar a los involucrados; para resolver esto existe otro tipo de flujo llamado *Pipeline;* este se caracteriza por ser más flexible que un flujo estático y posibilita utilizar condicionales y trabajos en paralelo para lograr satisfacer necesidades más complejas.

El manejo del *Pipeline* como código permite administrar el servidor de integración desde el repositorio de versiones; esto posibilita que Jenkins pueda utilizar recursos de diferentes repositorios de versiones y múltiples ramas; se logra colocando un archivo llamado Jenkinsfile en la raíz del proyecto, que consiste en un *script* que contiene la definición de los pasos a realizar; una buena práctica consiste utilizar proyectos de tipo múltiples ramas, lo cual hace posible funcionalidades como cambiar la forma en que se construye o despliega el proceso, según las condiciones del proyecto.

3.2.1. Agentes

Un agente define dónde será realizado el *pipeline* y lo que se ejecutará en cada agente es definido por la posición en la que la instrucción se coloca en el código; por ejemplo: si se desea que determinadas instrucciones se ejecuten sobre un contenedor Docker se podría utilizar la siguiente instrucción:

Figura 4. **Ejemplo creación de agente**



Fuente: elaboración propia, utilizando Visual Studio Code 1.50.

3.2.2. Pasos

Consiste en un conjunto de instrucciones separadas, lógicamente relacionadas con una tarea específica; es una forma de organizar las instrucciones que serán ejecutadas consecutivamente; estas podrían ser instrucciones como la descarga del código del repositorio de versiones.

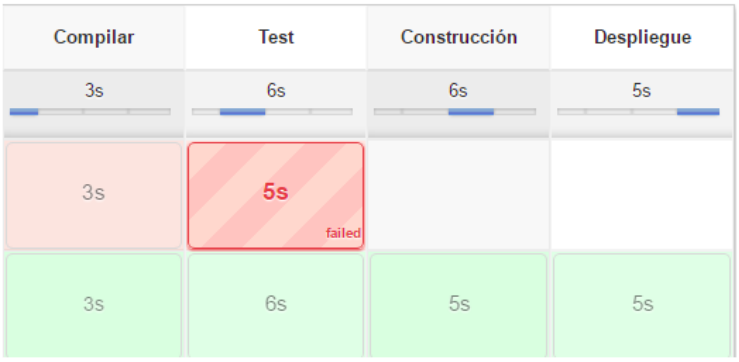
3.2.3. Nodos

Son agrupaciones de tareas o pasos que comparten un mismo espacio de trabajo; este conjunto de tareas por realizar espera la liberación de recursos del agente en el cual se encuentra asignado; cada nodo es independiente y con su propio espacio de trabajo, es decir que cada paso hace referencia al mismo espacio de trabajo y por ende accede a archivos, directorios, entre otros.

3.2.4. Etapas

Son un conjunto de tareas o pasos agrupados lógicamente; las etapas pueden contener pasos y nodos; esta es una buena práctica, ya que ordena las diferentes fases con las que cuenta el proceso.

Figura 5. **Ejemplo de pasos en Jenkins**



Fuente: elaboración propia, utilizando Jenkins Server 2.46.2.

3.2.5. Notificaciones

Son un conjunto de instrucciones de alerta que van dirigidas a los desarrolladores, para informar acerca de algún suceso; la notificación más utilizada es la de correo electrónico.

4. DEVOPS Y MICROSERVICIOS

Uno de los casos de éxito más conocidos de la unión de los microservicios y Devops es la de Amazon; en los alrededores del 2001 toda su infraestructura se basaba en un solo monolito, en donde la mayor parte de la atención se centraba en los requerimientos funcionales, pero se dejaban a un lado los requerimientos no funcionales; con el crecimiento del proyecto y necesidades de nuevas funcionalidades se volvían complicadas debido a la gran cantidad de equipos de desarrollo, pero solamente un equipo para atender los despliegues, lo cual producía cuellos de botella.

La solución para esta problemática obviamente era dividir el sistema en pequeños módulos administrables por pequeños equipos, pero seguía existiendo una problemática, aunque el manejo del proyecto era más fácil, la puesta a producción requería de varios equipos encargados del despliegue. por lo que se vieron la necesidad de desarrollar una herramienta en la cual se pudieran realizar las tareas de compilación, calidad de código y despliegue, en producción de forma automática; así fue el nacimiento de AWS CodeDeploy, una vez automatizado cada proceso de despliegue para cada microservicio, se resolvió gran parte de la problemática en la que se veían envueltos.

Los microservicios son pequeñas unidades independientes que deben representar un conjunto de funcionalidades específicas, donde un pequeño grupo de personas las puede administrar; los datos son manejados por cada microservicio y administrados independientemente por una capa de abstracción diferente; las tecnologías utilizadas en un microservicio no necesitan ser

homologadas, pero la comunicación entre ellos debe ser estandarizada para su correcto funcionamiento.

4.1. Puerta de enlace

Estas son puertas de enlace en donde se centraliza el tráfico de la aplicación; su principal función es administrar la comunicación entre los servicios, y se caracteriza por ofrecer un desacoplamiento e independencia, ya que los servicios pueden ser actualizados de forma transparente para el cliente; entre las funcionalidades que se pueden realizar está la de autenticación y verificación de permisos de acceso; esto debido a que es un punto de acceso central.

4.2. Escalabilidad

Una de las características de los sistemas distribuidos es la posibilidad de escalar horizontalmente conforme se requiera más recursos; esto hace necesaria la utilización de herramientas especiales para el manejo, control y monitoreo como Kubernetes, Docker Swarm, entre otras.

4.3. Manejo de errores

El aislamiento de los servicios garantiza que una falla en uno de los microservicios no comprometa el funcionamiento entero del sistema y permita la continuidad del negocio.

4.4. Tecnologías

Es posible utilizar tecnologías totalmente diferentes en cada microservicio, ya sea por los requerimientos específicos de las funcionalidades o habilidades de los programadores, pero es necesario mantener un estándar en la comunicación de los microservicios.

4.5. Administración de código

Debido a que las funcionalidades de los microservicios son básicas e independientes, esto reduce grandes cantidades de código con dependencias enredadas, ya que cada microservicio se enfoca en funcionalidades específicas; esto promueve el orden.

4.6. Fácil administración de equipos

Los equipos de desarrollo, como buena práctica, es aconsejable que sean reducidos; esto debido a que equipos numerosos dificultan la comunicación, haciéndola más lenta y reduciendo la agilidad.

4.7. Agilidad

La característica de independencia permite fácil implementación, corrección de errores y nuevas funcionalidades; si existen problemas esto no afectará al sistema general y al ser sistemas pequeños, habrá un mejor control y rápida identificación de errores.

4.8. Calidad y gobierno

Entre las dificultades que se pueden encontrar está la de los sistemas, que son puestos a prueba de manera individual; hay dificultades en generar pruebas al sistema en general; otro aspecto que compromete la calidad del producto es la gobernanza del sistema, el cual se compromete debido a que en situaciones en las que existen varios lenguajes y marcos de trabajo, se puede llegar a tener riesgos en la seguridad y los estándares para el proyecto.

4.9. Redes

El concepto de los microservicios expone que se centralizan las solicitudes en un intermediario, quien hace una redirección de las solicitudes, el mal diseño de la comunicación de los servicios puede llegar a congestionar la comunicación, esto debido a la interdependencia para satisfacer una solicitud; es necesario tener cuidado en la forma en que se hace el diseño del sistema en general, para evitar este tipo de problemas.

5. CASO DE ESTUDIO Y SOLUCIÓN PROPUESTA

En el presente capítulo se expone un ejemplo práctico sobre el cual se propone una solución basada en Devops, según los requerimientos planteados en el caso de estudio.

5.1. Descripción del caso de estudio

Una determinada entidad bancaria a lo largo de los años ha expandido sus operaciones fuera de Guatemala y se ha esforzado para satisfacer las necesidades internas y externas de información, en donde se ha visto en la necesidad de recurrir a distintas herramientas de visualización de datos fuera de una infraestructura común de un *Datawarehouse*, esta entidad cuenta con las siguientes necesidades:

- Demanda de diversos tipos de análisis y visualización.
- Requiere que la información se encuentre disponible en todo momento.
- Los usuarios hacen requerimientos específicos que las herramientas de inteligencia de negocios no logran satisfacer.
- Para requerimientos específicos es necesario utilizar diversas herramientas, lenguajes de programación o librerías específicas, las cuales se manejan aisladamente sin algún control o gobierno de las herramientas utilizadas.

5.2. Solución propuesta

Se propone una infraestructura basada en tecnologías que garantice la alta disponibilidad y escalabilidad, según demanda para el ahorro de costos; esto se logra mediante la utilización de microservicios que permitan separar el sistema en pequeñas partes y la posibilidad de utilizar diversas herramientas o lenguajes de programación, manejados por contenedores y administrados por herramientas de orquestación de los mismos, logrando las siguientes características.

- Manejo de la infraestructura como código.
- Fácil migración del sistema de un proveedor de servicios de nube a otro.
- Reducción de costos debido a las características de escalado automático.
- Manejo de entornos de prueba para garantizar la calidad.

5.3. Análisis de aplicabilidad de un entorno Devops al problema propuesto

Es necesario realizar un análisis antes de invertir en un entorno de automatización de procesos, debido a que existen distintos elementos que se deben tomar en cuenta, si se va a implementar una solución de este tipo.

- Personas: el material humano es el ingrediente más importante; las diferentes características que poseen las personas incidirán en el logro de un sistema integrado y automatizado, ya que como se sabe se realizará una tarea de integración de diferentes roles como administradores de sistemas, desarrolladores, equipos de calidad de código, redes, entre otros. Por lo que implica un equipo multidisciplinario

que sea capaz de configurar las herramientas utilizadas para cumplir su objetivo.

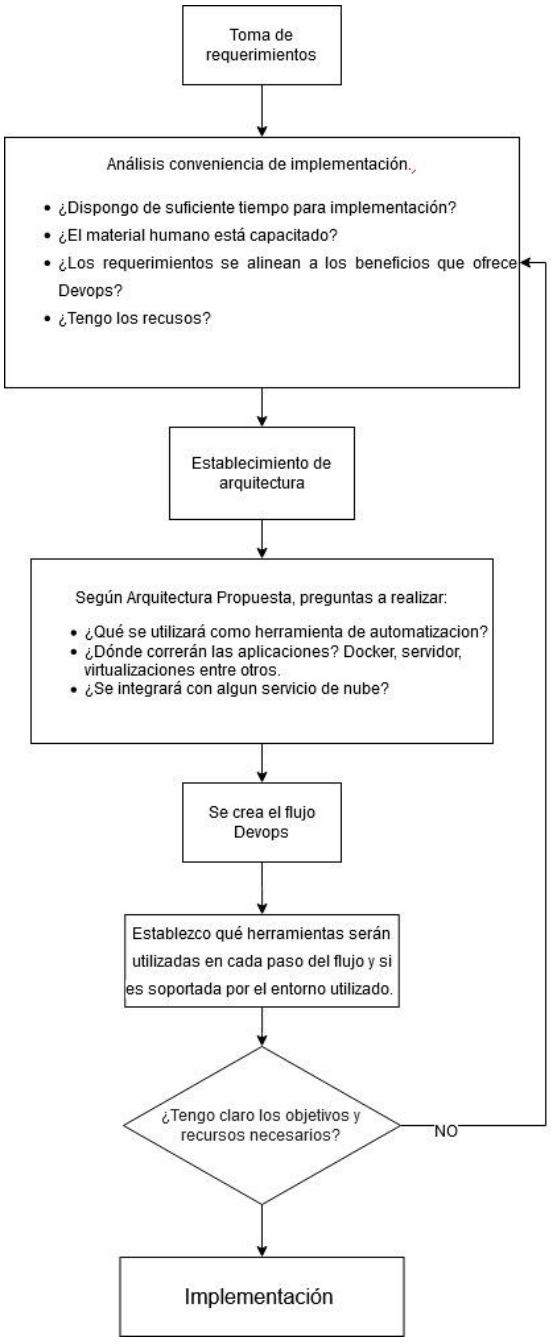
- Inversión inicial: se debe considerar el factor de tiempo inicial que se invertirá en la implementación; esta dependerá de la experiencia y las capacidades del equipo, debido a que aquí es donde se encuentra la mayor parte del trabajo y se debe considerar un lapso de tiempo para empezar a ver resultados.

- Cultura: la organización entera debe estar comprometida y saber hacia dónde se quiere dirigir; así como identificar las dificultades y beneficios de una cultura colaborativa, en la que todos buscan hacer eficientes los procesos y la satisfacción del cliente mediante prácticas como Devops; esto implicará un cambio completo de la forma en que las personas hacen las cosas, que va desde adoptar prácticas como metodologías ágiles hasta la manera en que se entregan al cliente los sistemas.

- Tiempos de entrega: un factor que se debe tomar en cuenta es el tiempo que los clientes permanecen en espera para recibir un requerimiento; he aquí uno de los factores más importantes para saber si vale la pena o no adoptar estas prácticas, las cuales dependen de las tareas post desarrollo y el tiempo que se invierte en colocar los sistemas en producción.

- Herramientas: se debe considerar que al implementar este tipo de prácticas es necesario un cambio total en las herramientas y cómo se utilizan habitualmente; esto debido a que tienen que acoplarse a tecnologías como: contenedores, servidores de integración continua, orquestadores de contenedores, servicios en la nube, entre otras.

Figura 6. **Flujo de análisis previo a implementación**



Fuente: elaboración propia, empleando el programa DRAW.IO 13.8.8.

5.4. Justificación de arquitectura microservicios

Se establece que las características que una infraestructura desacoplada provee lo necesario para el problema propuesto, ya que permitirá disponer de cualquier herramienta o librería en un entorno aislado a las demás herramientas utilizadas.

Tabla I. **Análisis de arquitectura propuesta**

Requerimiento	**Microservicios**	**Monolito**
Desempeño	Distribuye el procesamiento según sea la solicitud en pequeñas porciones de recursos asignados a determinadas tareas, lo cual hace eficiente las respuestas, ya que estas son completamente independientes de cualquier otro módulo.	Centraliza y acopla en un solo recurso el procesamiento y funcionalidades de un sistema; sus beneficios se centran en la disposición inmediata de los elementos necesarios para satisfacer una solicitud, ya que se encuentran en un único servidor.
Facilidad de prueba	Requiere un trabajo adicional, ya que las pruebas deben ser personalizadas a cada módulo, así como pruebas de integración del sistema como tal.	Posee ventajas, ya que todo el sistema está desarrollado en un mismo entorno y el trabajo de calidad se puede realizar directamente.

Continuación de la tabla I.

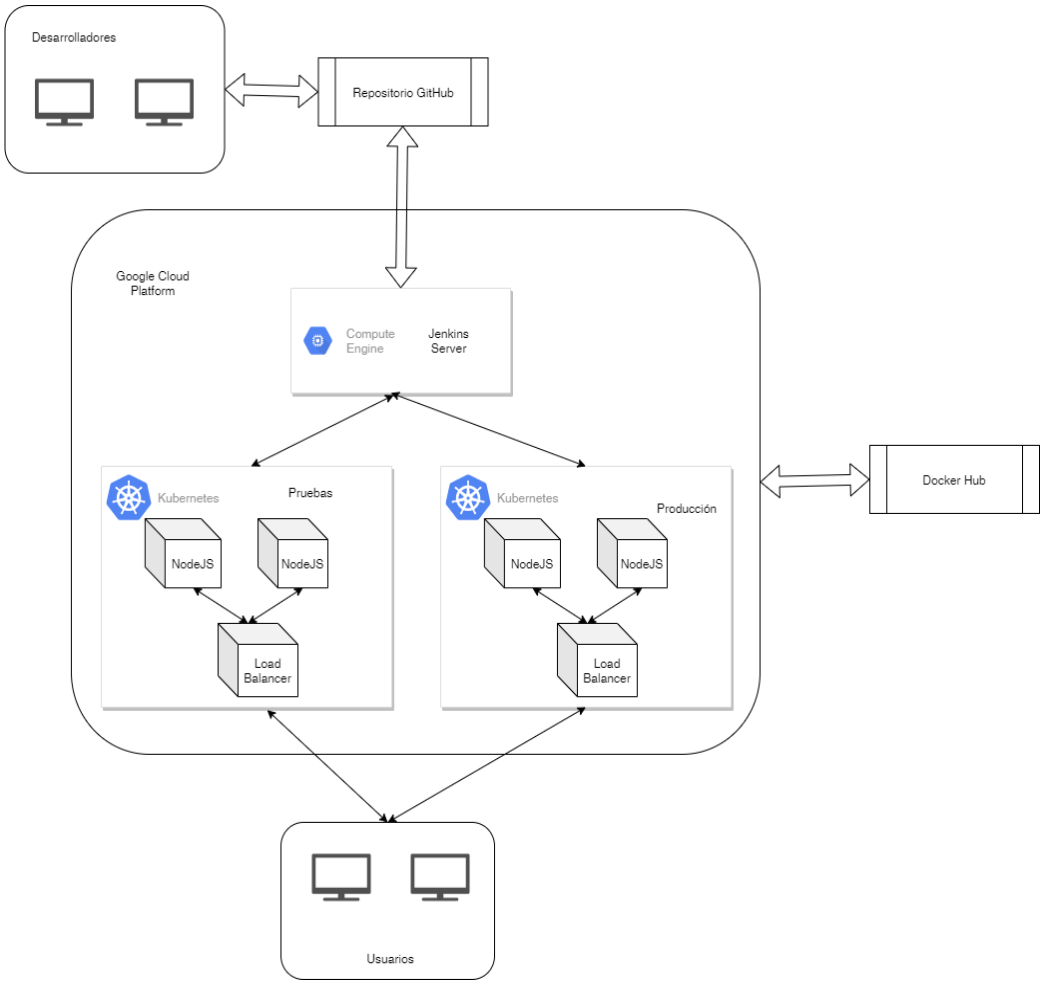
Seguridad	Cualquier problema existente en un módulo del sistema es totalmente independiente y no compromete su funcionalidad en general.	Centraliza en un solo lugar las tareas de monitoreo, pero al existir problemas en un elemento del sistema puede comprometer todo el funcionamiento.
Modificabilidad	Al ser un sistema distribuido tiene ventajas en su desarrollo, ya que es totalmente independiente; aunque puede llegar a tener problemas en el acoplado en el sistema en general.	Tiene ventajas, de manera que todo el sistema se encuentra ejecutado en un solo programa, pero debido a estas características puede llegar a tener complicaciones si el desarrollo de este no ha sido adecuado.
Usabilidad	Explota sus beneficios, ya que cada módulo se encarga de una tarea específica.	No requiere integraciones, ya que el sistema es pensado como un todo y responde específicamente para las funcionalidades que fue desarrollado.

Fuente: elaboración propia, utilizando Excel 2016.

5.5. Arquitectura propuesta

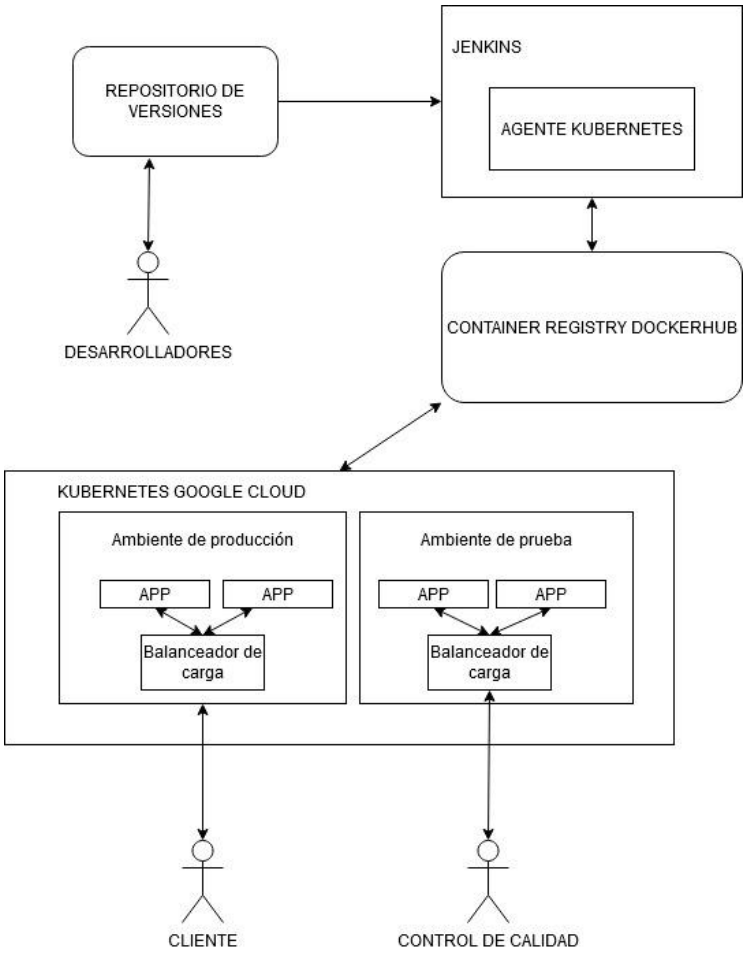
La arquitectura propuesta consta de elementos como Kubernetes, el cual es utilizado para hospedar las diferentes imágenes de Docker en el sistema, dando la capacidad de incrementar o reducir los recursos según la exigencia a la que se vea sometido dicho sistema.

Figura 7. **Diagrama de despliegue**



Fuente: elaboración propia, empleando el programa DRAW.IO 13.8.8.

Figura 8. **Diagrama de componentes**



Fuente: elaboración propia, empleando el programa DRAW.IO 13.8.8.

En el diagrama de despliegue se muestra la interacción de los microservicios de análisis de datos, interactuando con el balanceador de carga, quien será el encargado de distribuir las solicitudes; este diagrama también evidencia la relación que existe entre los componentes como el repositorio de versiones, servidor de integración continua y el repositorio de imágenes Dockerhub.

5.6. Herramientas e infraestructura del entorno Devops

Consiste en el análisis en todas aquellas herramientas compatibles con las prácticas que se implementarán para asegurar el funcionamiento adecuado, tanto del sistema como del ciclo Devops.

5.6.1. Costos de herramientas

En cuanto al costo de las herramientas de automatización de procesos o más conocidos como servidores de integración continua, estos varían según las necesidades de la solución a implementar, ya que servidores como TravisCi y CircleCi se enfocan en proveer plataformas en las cuales se evita el trabajo de configuración de la herramienta en sí y se centran en la integración con los diferentes elementos que estarán interactuando en el flujo automatizado.

Aunque existen versiones de instalación local, el soporte puede ser limitado, por lo cual si se quieren aprovechar al máximo las funcionalidades que ofrecen, es recomendable usar las versiones de pago en la nube; en su contraparte Jenkins tiene la particularidad que nació como servidor de integración continua orientado principalmente a instalaciones en entornos locales (no nube) y al ser código abierto da la posibilidad de obtener el abanico principal de funcionalidades, limitándose al costo del entorno donde se ejecuta el servidor de integración; también existen soluciones ya configuradas, listas para el uso de Jenkins, ofrecidas por proveedores de servicios en la nube.

5.6.2. Compatibilidad

Este es uno de los elementos principales a la hora de elegir un servidor de integración continua; por tal motivo es aconsejable realizar un análisis y diseño de la infraestructura a utilizar, en donde se puedan definir claramente las interacciones con las que se van a enfrentar, ya que todo se centra y gobierna desde este punto; Jenkins se maneja por medio de complementos descargables desde su página principal, en la que la comunidad se encarga de desarrollar y acoplar distintos tipos de funcionalidades; en su repertorio de complementos se pueden encontrar alrededor de tres mil complementos para temas específicos, por lo cual da una amplia variedad de opciones.

En cuanto a soluciones como CicleCi y TravisCi, estas pueden llegar a tener complicaciones por la característica que ofrece ambientes ya configurados limita la posibilidad de realizar tareas específicas; las cuales se podrían realizar fácilmente si se ejecutaran sobre un servidor dedicado, aunque es muy poco probable encontrar estas situaciones, ya que proveen una gran variedad de opciones para las implementaciones.

5.6.3. Creación de flujos automatizados

Jenkins provee varias opciones en cuanto a la automatización de tareas, en donde es posible auxiliarse con los complementos desde una interfaz gráfica web, donde se pueden crear los pasos del flujo automatizado, aunque estas prácticas muchas veces van en contra de los principios Devops, ya que hacen dependiente a una solución específica, dificultando las tareas como por ejemplo migración de la solución a otro proveedor de servicios en la nube.

Jenkins también da la opción de la creación de flujos por medio de scripts, lo cual provee mayor control de las funcionalidades y el manejo directo desde un repositorio de versiones, algunas de las dificultades aparecen a la hora de utilizar complementos debido al poco soporte existente.

Travis Ci ofrece la funcionalidad de creación de flujos a través de script, aunque realizar despliegues por este medio no es complicado, ya que utiliza un lenguaje sencillo y fácil de entender.

5.6.4. Alojamiento

Tanto Travis CI como Circle Ci son denominados nativos de la nube; lo cual quiere decir que estos se enfocan en ofrecer servicios completamente alojados en la nube, aunque ambos poseen versiones donde se puede configurar en servidores dedicados; esto agrega una capa de complejidad a la solución propuesta.

En cuanto a Jenkins, tiene la particularidad que nació como un proyecto para ejecutarse en servidores dedicados; la migración a un sistema en la nube no implica mayor diferencia a la utilizada localmente.

5.6.5. Evaluación manejo de repositorio

Se establece que se utilizará un repositorio Github por la fácil integración con aplicaciones de terceros para personalizar las distintas tareas a realizar. Existe un modelo de ramificación llamado desarrollo basado en truncamiento (*trunk based development*) por la compatibilidad e integración con entornos ágiles, en donde se establece la siguiente premisa: la menor cantidad de ramas existentes y entregas rápidas a producción.

5.6.6. Metodología de desarrollo

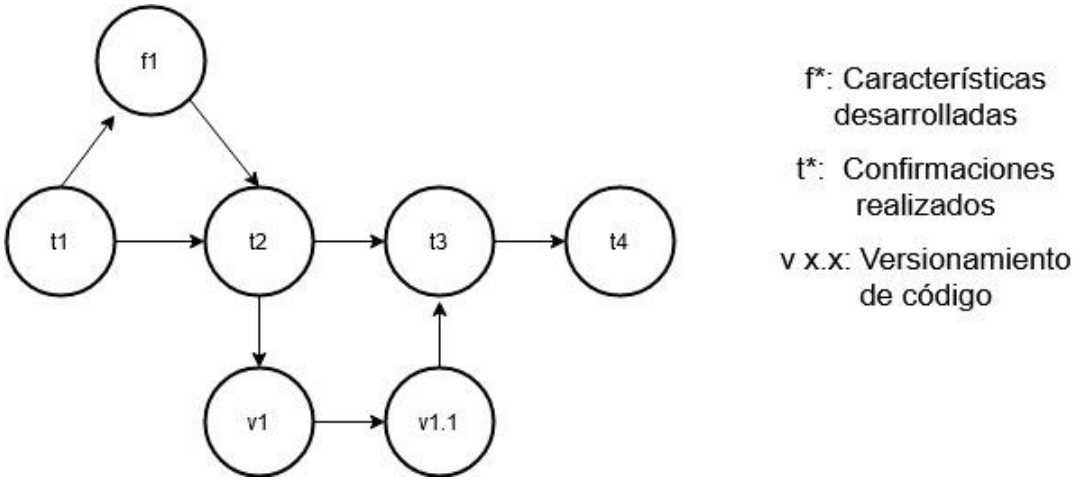
La forma en la que se administra el proyecto impacta directamente en los recursos como costo y tiempo, por lo que se recomienda utilizar metodologías ágiles como Scrum o Kanban, las cuales fueron creadas para proporcionar flexibilidad y entregas a clientes en ciclos cortos.

Scrum está basado en un término utilizado en *Rugby*; es una de las jugadas más usadas para conseguir el balón y ponerlo en juego nuevamente; aplicado a un proyecto de software implica basarse en el desarrollo incremental; en lugar de realizar un análisis general de una solución, la planificación detallada es definida según avanza el proyecto.

Kanban ofrece una organización visual de los proyectos enfocada en el proceso desde que se hace el requerimiento hasta el despliegue en producción; el estudio de los tiempos como duración del trabajo, tiempos de espera y tiempos de entrada y de salida, son métricas utilizadas para optimizar los resultados.

En cuanto a la compatibilidad que existe en un ambiente Devops, tanto Kanban como Scrum pueden ser adecuados, donde Scrum está enfocado a la división del proyecto en pequeñas entregas totalmente funcionales al cliente, a diferencia de Kanban, que, aunque puede ser adecuado se centra en la división de las tareas para hacer fluido el progreso del proyecto.

Figura 9. **Desarrollo basado en truncamiento**



Fuente: elaboración propia, empleando el programa DRAW.IO 13.8.8.

En la anterior figura se presenta un modelo básico del versionamiento de la aplicación, en el cual se destaca la poca utilización de ramas y las rápidas integraciones a la rama máster.

5.7. Análisis ambientes de ejecución de código

Para la ejecución de las aplicaciones es necesario un entorno que contenga todas sus dependencias para ejecutar el proyecto; una de las opciones para esto es utilizar servidores dedicados, los cuales tienen la característica de no compartir recursos y enfocarse exclusivamente a las tareas asignadas, bajo una perspectiva tradicional; esta opción puede llegar a ser interesante, ya que provee entornos conocidos, lo cual provoca un nivel de confianza en control de las tareas que se realizan; aunque muchas veces puede ser ventajoso, también tiene sus puntos negativos, uno de los principales es el desperdicio de recursos, ya que para ejecutar un software es necesario iniciar

un sistema operativo completo, desperdiciando recursos en tareas que no son de valor para el sistema que está en ejecución.

Entre las alternativas encontradas para empaquetar, distribuir y ejecutar aplicaciones está la utilización de contenedores en donde se administran dependencias y bibliotecas, garantizando la ejecución en cualquier entorno, no importando las configuraciones existentes en la máquina o servidor, en donde se ejecute el contenedor.

5.8. Herramientas de despliegue y entrega continua

Dentro la entrega y despliegue continuo, hoy en día el uso de contenedores y microservicios ha revolucionado la forma en la que se diseñan las arquitecturas de software, lo cual implica que, aunque se tiene el sistema distribuido en partes pequeñas, la suma de todos los elementos puede llegar a crecer y surge otra problemática: el manejo de los elementos de la infraestructura, lo que requiere de herramientas conocidas como orquestadores.

5.8.1. Kubernetes

Es una herramienta para administrar el despliegue de contenedores que se caracteriza por el escalado, replicación, técnicas de despliegue, entre otras; su mayor fortaleza se enfoca en la rápida entrega de ambientes y la portabilidad del sistema.

5.8.2. Docker Swarm

Consiste en una herramienta de orquestación de contenedores que utiliza múltiples hospedadores para permitir su ejecución de manera distribuida, utilizando una arquitectura maestro-esclavo; cada clúster está formado al menos por un nodo maestro y los nodos esclavos.

5.8.3. Comparativa

Docker Swarm y Kubernetes comparten características, pero se define que los elementos más importantes a analizar son: definición de arquitectura, alta disponibilidad, administración de recursos y soporte.

5.8.3.1. Definición de arquitectura

Tanto la definición en Kubernetes como en Docker Swarm se realiza en archivos YAML; la diferencia radica que Kubernetes utiliza la definición de clústeres que engloban los diferentes elementos de un proyecto; estos pueden ser llamados pods, que son la unidad mínima manejada por Kubernetes; se definen como un contenedor que puede ser administrado y replicado, ya sea de forma automática o manual; las arquitecturas más compatibles pueden ser basadas en microservicios o servicios.

Docker Swarm permite la definición de las arquitecturas por medio de Docker Compose, que es una herramienta para la definición de arquitecturas basadas en contenedores donde se pueden configurar redes, servicios almacenamiento, entre otros. Estos archivos son utilizados como plantillas que se administran en los nodos de Docker Swarm; en este aspecto ambos son similares, la diferencia radica que Kubernetes obliga a la creación de imágenes

personalizadas y que esta sea subida a un repositorio de imágenes para luego ser utilizada.

5.8.3.2. Alta disponibilidad

Kubernetes administra la disponibilidad utilizando redireccionamiento Haproxy según el servicio establecido; el cual define la redireccion de solicitudes. Kubernetes administra el estado de estos *pods,* excluyendo los que se encuentren con problemas y agregando nuevos automáticamente, para garantizar que el sistema responda cuando sea solicitado.

Docker Swarm utiliza el manejo de maestros y esclavos para garantizar el cumplimiento de las solicitudes; el administrador Swarm es el encargado de distribuir y orquestar solicitudes y los elementos desplegados entre los nodos que se disponen en la implementación.

5.8.3.3. Administración automática de recursos

Aquí existe una diferencia importante entre ambas tecnologías, ya que Kubernetes permite la replicación automática mediante la definición del uso de recursos pertenecientes a un clúster; esta funcionalidad es importante para garantizar atributos como la disponibilidad; Swarm no provee exactamente esta funcionalidad.

5.8.3.4. Soporte

Kubernetes, creado y utilizado por Google, ha sido difundido y popularizado entre la comunidad del desarrollo; se encuentra en crecimiento su uso; nubes como Amazon, Azure y Google ofrecen plataformas para

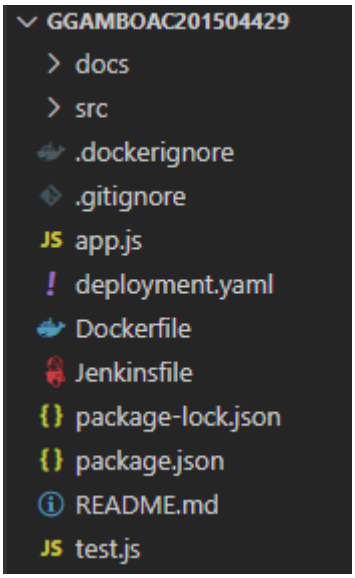
implementaciones de Kubernetes, aunque cabe destacar que la complejidad de configuración es mayor en Kuberne.

6. IMPLEMENTACIÓN DEVOPS

6.1. Estructura del proyecto

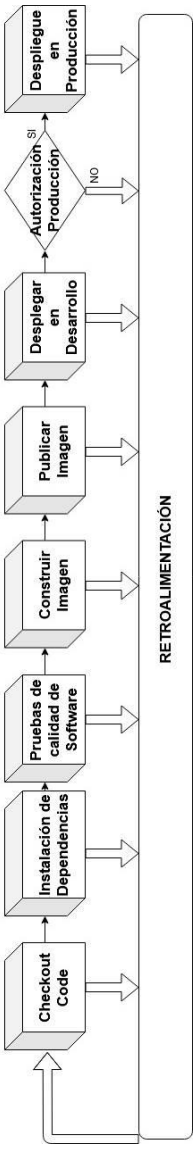
La estructura del proyecto se debe organizar para facilitar la implementación del despliegue; en este caso se acomodó a conveniencia según el estilo de manejo del repositorio basado en truncamiento.

Figura 10. **Estructura del proyecto**



Fuente: elaboración propia, utilizando Visual Studio Code versión 1.48.2.

Figura 11. **Flujo del proceso automatizado**



Fuente: elaboración propia, empleando el programa draw.io 13.8.8.

6.2. Configuración del ambiente de ejecución

La aplicación se ejecutará sobre contenedores Docker manejados por Kubernetes; esta configuración se realiza por medio de un archivo yaml, que contiene la definición de los contenedores con los que se estará trabajando, así como los servicios que se expondrán para que el usuario final tenga acceso al mismo.

Figura 12. **Configuración del ambiente de ejecución**

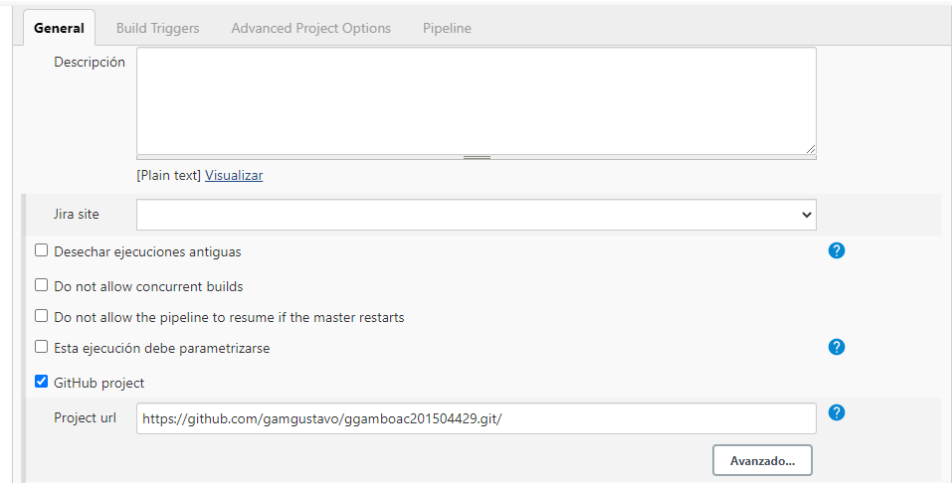
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: devops-demo-app
spec:
  selector:
    matchLabels:
      app: devops-demo
  replicas: 3
  template:
    metadata:
      labels:
        app: devops-demo
    spec:
      containers:
      - name: devops-demo
        image: "gustavogamboa/devops-demo:latest"

---

apiVersion: v1
kind: Service
metadata:
  name: ilb-service
  annotations:
    cloud.google.com/load-balancer-type: "External"
  labels:
    app: devops-demo
spec:
  type: LoadBalancer
  selector:
    app: devops-demo
  ports:
  - port: 80
    targetPort: 5000
    protocol: TCP

```

Fuente: elaboración propia, empleando el programa DRAW.IO 13.8.8.

6.3. Inicializadores del proceso Devops

El proceso Devops es inicializado automáticamente por medio de la configuración del repositorio de versiones Github, en el cual se configura él envío de una notificación al servidor de integración continua cada vez que existe un cambio.

Figura 13. **Configuración del repositorio de versiones**



Fuente: elaboración propia, empleando el programa DRAW.IO 13.8.8.

6.4. Clonación del repositorio de versiones

En esta parte del proceso se realizan las tareas de limpieza del espacio de trabajo, inicialización de un repositorio en el directorio local y clonación del código en el entorno del servidor de integración.

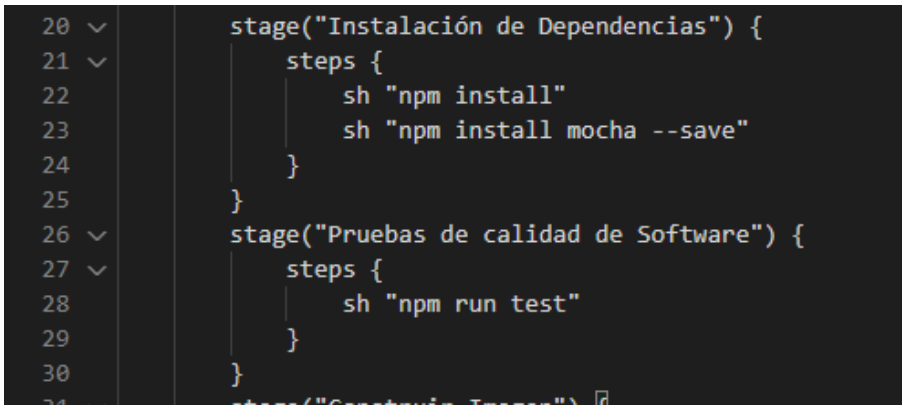
Figura 14. **Resultados de consola servidor de integración continua**

Fuente: elaboración propio, utilizando Jenkins Server.

6.5. Pruebas

Para la ejecución de pruebas es necesario instalar las dependencias de la herramienta utilizada para la calidad de software; de esta forma se puede acceder y ejecutar automáticamente las pruebas realizadas para el proyecto.

Figura 15. **Configuración y ejecución del ambiente de pruebas**



Fuente: elaboración propia, utilizando Visual Studio Code 1.50.

6.6. Construcción y publicación

Se debe establecer el contenedor que se construirá y publicará en el repositorio de contenedores de DockerHub; para esto es necesario configurar en el servidor de integración continua las credenciales de DockerHub y de Google Cloud.

Figura 16. **Construcción y publicación**

```
31      stage("Construir Imagen") {
32          steps {
33              script {
34                  myapp = docker.build("gustavogamboa/devops-demo:${env.BUILD_ID}")
35              }
36          }
37      }
38      stage("Publicar Imagen") {
39          steps {
40              script {
41                  docker.withRegistry('https://registry.hub.docker.com', 'dockerhub') {
42                          myapp.push("latest")
43                          myapp.push("${env.BUILD_ID}")
44                  }
45              }
46          }
47      }
```

Fuente: elaboración propia, utilizando Visual Studio Code 1.50.

- Variables de entorno: como buena práctica es recomendable utilizar variables para la fácil configuración y migración de los proyectos; donde se detallan las variables más importantes como el nombre del proyecto, aplicación, clúster, zona, entre otros.

Figura 17. **Configuración de variables del proyecto**

```
    agent any
    environment {
        PROJECT_ID = 'ggamboac-201504429-01'
        CREDENTIALS_ID = 'ggamboac-201504429-01'

        CLUSTER_NAME_TEST = 'devops-demo-test'
        LOCATION_TEST = 'us-east1-c'

        CLUSTER_NAME_PROD = 'devops-demo-prod'
        LOCATION_PROD = 'us-west1-a'

    }
```

Fuente: elaboración propia, utilizando Visual Studio Code 1.50.

6.7. Despliegues

Los despliegues a producción se realizarán por medio de una autorización; es decir los cambios se publicarán previamente en un entorno de pruebas donde se pueda verificar el sistema, y si el resultado es satisfactorio, los cambios pueden ser desplegados en el entorno de producción.

Figura 18. **Despliegue a clúster de prueba y producción**

Fuente: elaboración propia, empleando Jenkins Server.

Figura 19. **Configuración de despliegues**

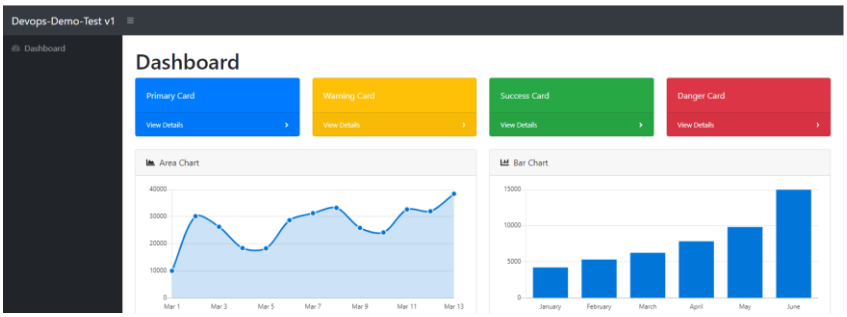
```
48          stage('Desplegar en Pruebas') {
49              steps{
50                  sh "sed -i 's/devops-demo:latest/devops-demo:${env.
51                  step([$class: 'KubernetesEngineBuilder',
52                  projectId: env.PROJECT_ID,
53                  clusterName: env.CLUSTER_NAME_TEST,
54                  location: env.LOCATION_TEST,
55                  manifestPattern: 'deployment.yaml',
56                  credentialsId: env.CREDENTIALS_ID,
57                  verifyDeployments: true])
58              }
59          }
60          stage('Despliegue en Producción') {
61              steps{
62                  input message:"¿Autorizar Despliegue?"
63                  step([$class: 'KubernetesEngineBuilder',
64                  projectId: env.PROJECT_ID,
65                  clusterName: env.CLUSTER_NAME_PROD,
66                  location: env.LOCATION_PROD,
67                  manifestPattern: 'deployment.yaml',
68                  credentialsId: env.CREDENTIALS_ID,
69                  verifyDeployments: true])
70              }
71          }
```

Fuente: elaboración propia, utilizando Visual Studio Code 1.50.

6.8. Ejemplo del flujo Devops

En dicho momento que se origina una solicitud de cambio y se procede a realizar los cambios requeridos es necesario entender el flujo que se llevará a cabo para ejemplos ilustrativos simulará que se solicitó el cambio del título “Devops-Demo-Test v1” a “Devops-Demo-Test v2”.

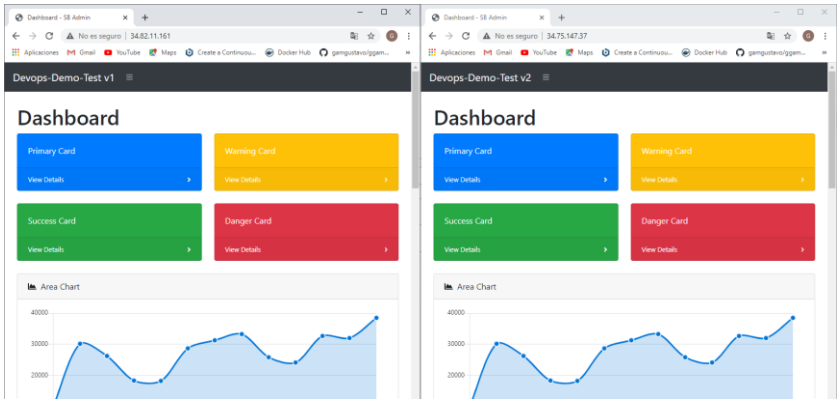
Figura 20. **Estado inicial del tablero**



Fuete: elaboración propia, empleado Google Chrome 86.0.42.

Luego de realizar el cambio y publicar los cambios en la rama máster, se da el inicio automático del proceso Devops, pasando por las fases establecidas; de tal forma que los cambios se publican en el entorno de pruebas, y una vez completada la revisión final se pueda proceder a autorizar o rechazar la publicación en el entorno de producción.

Figura 21. **Publicación en entorno de pruebas previo a autorización o rechazo**

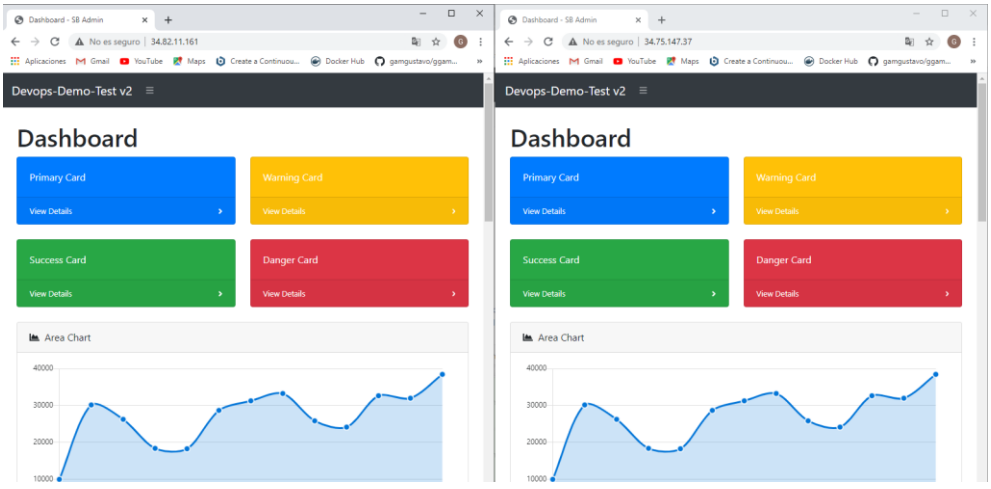


Fuente: elaboración propia, empleado Google Chrome 86.0.42.

Figura 22. **Rechazo o autorización del cambio**

Fuente: elaboración propia, empleando Jenkins Server.

Figura 23. **Publicación en entorno de producción**



Fuente: elaboración propia, empleado Google Chrome 86.0.42.

CONCLUSIONES

1. Después de realizar el análisis de los diferentes factores que se deben tomar en cuenta en una implementación Devops define que los beneficios más destacables son: aumento de la productividad, ahorro de tiempo en tareas repetitivas, más despliegues y con mayor frecuencia y software de mejor calidad.

2. Se identificó que para alcanzar una implementación Devops satisfactoria dentro de la planificación, se debe asegurar que hay equipos multidisciplinarios de la rama de la administración de sistemas y el desarrollo de software, así como contar con el tiempo inicial suficiente para la configuración de la infraestructura Devops.

3. Se propuso una infraestructura que tiene las características de escalabilidad horizontal que permite tanto aumentar como disminuir los recursos para soportar el tráfico al que se verá sometido, así como centralizar el gobierno de la infraestructura en un orquestador de contenedores para facilitar el manejo de la seguridad del sistema; se utilizarán contenedores para dar flexibilidad y poder usar diferentes herramientas y lenguajes de programación en la misma solución.

4. Dentro de las buenas prácticas utilizadas en la solución Devops están: utilizar diferentes marcos de trabajo acordes a una cultura de agilidad y entrega continua como Scrum en el entorno del desarrollo de software; otra práctica importante es el manejo de la mayor parte de la solución

como código manejable desde el repositorio de versiones; esto da flexibilidad de migración.

5. Para el problema propuesto se define que el conjunto de herramientas más adecuado incluye: Github para el manejo del repositorio de versiones, por la facilidad de integración a herramientas de terceros; Docker, para el manejo de los ambientes de ejecución de software; Jenkins como servidor de integración continua y Kubernetes como orquestador de contenedores.

RECOMENDACIONES

1. Es necesario que las organizaciones cuenten con normativas de documentación de procesos para garantizar la fácil integración de nuevos miembros y que haya transiciones transparentes entre el personal que administra los sistemas.

2. Seleccionar cuidadosamente los proyectos a los cuales se aplicará Devops, para garantizar la máxima obtención de beneficios.

3. Elegir herramientas que se acoplen de forma nativa a ambientes Devops; utilizar herramientas no soportadas o desconocidas por el personal puede llegar a complicar el funcionamiento de los sistemas.

4. Es recomendable el manejo entero de las implementaciones Devops desde el repositorio de versiones, evitando costosas configuraciones sobre servidores dedicados.

5. Para garantizar la seguridad de los sistemas se debe separar y aislar los microservicios de una red a otra, cifrar la comunicación entre los contenedores, centralizar el manejo y gobierno de los sistemas y utilizar puertas únicas de acceso a los sistemas.

BIBLIOGRAFÍA

1. DAVIS, Jennifer; DANIELS, Ryn. *Effective Devops: building a culture of collaboration, affinity, and tooling at scale*. [en línea]. <https://sauleh.github.io/fc98/static_files/materials/Effective_DevOps.pdf>. [Consulta: 24 de febrero de 2019].

2. GOUIGOUX, Jean-Philippe. *Docker: primeros pasos y puesta en práctica de una arquitectura basada en microservicios.* 1a ed. Barcelona: ENI, 2018. 468 p.

3. KIM, Gene., BEHR, Kevin; SPAFFORD, George. *The Phoenix project: a novel about IT, Devops, and helping your business win*. 1a ed. Estados Unidos: IT Revolution Press, 2013. 345 p.

4. KIM, Gene., DEBOIS, Patrick., WILLIS, John., HUMBLE, Jez; ALLSPAW, John. *The Devops handbook: how to create world-class agility, reliability, & security in technology organizations*. 1 ed. Estados Unidos: IT Revolution Press, 2016. 655 p.

5. KIM, Gene. *The unicorn project: a novel about digital disruption, developers, and overthrowing the ancient powerful order*. 1 ed. Estados Unidos: IT Revolution Press, 2019. 352 p.

6. POPPENDIECK, Mary; POPPENDIECK, Tom. *Lean software development: an agile toolkit*. [en línea].

<https://sauleh.github.io/fc98/static_files/materials/Effective_DevOps.pdf>. [Consulta: 24 de febrero de 2019].

7. NAYYAR, Anand. *Instant approach to software testing: principles, applications, techniques, and practices*. 1a ed. India: BPB Pulications, 2019. 368 p.

8. NICKOLOFF, Jeff. *Docker in action*. 1a ed. Estados Unidos: Manning Publications, 2016. 304 p.

9. VERONA, Joakim. *Practical Devops: implement Devops in your organization by effectively building, deploying, testing, and monitoring code*. Reino Unido: Packt Publishing, 2018. 250 p.