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

# DISEÑO DE UN MODELO DE INTEGRACIÓN CONTINUA UTILIZANDO JENKINS PARA PROYECTOS DE DESARROLLO EN UNA EMPRESA DE TELECOMUNICACIONES PARA EL MÓDULO DE CONTABILIDAD

**Norman Eduardo Casasola Girón**
Asesorado por el Ing. David Armando Chang Ovando

Guatemala, abril de 2019

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA


FACULTAD DE INGENIERÍA

# DISEÑO DE UN MODELO DE INTEGRACIÓN CONTINUA UTILIZANDO JENKINS PARA PROYECTOS DE DESARROLLO EN UNA EMPRESA DE TELECOMUNICACIONES PARA EL MÓDULO DE CONTABILIDAD

TRABAJO DE GRADUACIÓN

PRESENTADO A JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

**NORMAN EDUARDO CASASOLA GIRÓN**
ASESORADO POR EL ING. DAVID ARMANDO CHANG OVANDO

AL CONFERIRSELE EL TÍTULO DE

**INGENIERO EN CIENCIAS Y SISTEMAS**

GUATEMALA, ABRIL DE 2019

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA


## NÓMINA DE JUNTA DIRECTIVA

| | |
|---|---|
| DECANO | Ing. Pedro Antonio Aguilar Polanco |
| 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. Luis Diego Aguilar Ralón |
| VOCAL V | Br. Christian Daniel Estrada Santizo |
| SECRETARIA | Inga. Lesbia Magalí Herrera López |

## TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

| | |
|---|---|
| DECANO | Ing. Murphy Olympo Paiz Recinos |
| EXAMINADOR | Ing. Edgar Estuardo Santos Sutuj |
| EXAMINADOR | Ing. Manuel Haroldo Castillo Reyna |
| EXAMINADORA | Inga. Virginia Victoria Tala Ayerdi |
| SECRETARIA | Inga. Marcia Ivonne Véliz Vargas |

## 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:

**DISEÑO DE UN MODELO DE INTEGRACIÓN CONTINUA UTILIZANDO JENKINS PARA PROYECTOS DE DESARROLLO EN UNA EMPRESA DE TELECOMUNICACIONES PARA EL MÓDULO DE CONTABILIDAD**

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 22 de noviembre de 2018.

**Norman Eduardo Casasola Girón**

Guatemala, 13 de febrero del 2019

Ingeniero
Carlos Alfredo Azurdia
Coordinador de Privados y Revisor de Trabajos de Graduación
Escuela de Ciencias y Sistemas
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

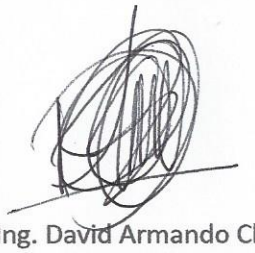
Ingeniero Azurdia:

Tengo el agrado de dirigirme a usted para informarle que he revisado el trabajo de graduación **"DISEÑO DE UN MODELO DE INTEGRACIÓN CONTÍNUA UTILIZANDO JENKINS PARA PROYECTOS DE DESARROLLO EN UNA EMPRESA DE TELECOMUNICACIONES PARA EL MÓDULO DE CONTABILIDAD"**, realizado por el estudiante universitario **NORMAN EDUARDO CASASOLA GIRÓN** con carné 199712623, quien contó con la asesoría del suscrito.

Considero que el trabajo realizado por el estudiante cumple con los objetivos bajo los cuales fue planteado y cumple satisfactoriamente cada una de las actividades planificadas, por lo que procedo a aprobarlo.

Agradeciendo la atención dada a la presente,

Atentamente,



MBA Ing. David Armando Chang Ovando
Colegiado: 8634

David Armando Chang Ovando
Ingeniero en Ciencias y Sistemas
Colegiado No. 8634

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

Guatemala, 28 de febrero de 2019

Ingeniero
**Marlon Antonio Pérez Türk**
**Director de la Escuela de Ingeniería**
**En Ciencias y Sistemas**

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **NORMAN EDUARDO CASASOLA GIRÓN** con carné **199712623 y CUI 2594 13070 0101** titulado **DISEÑO DE UN MODELO DE INTEGRACIÓN CONTÍNUA UTILIZANDO JENKINS PARA PROYECTOS DE DESARROLLO EN UNA EMPRESA DE TELECOMUNICACIONES PARA EL MÓDULO DE CONTABILIDAD** 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,

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
INGENIERÍA EN CIENCIAS Y SISTEMAS
COORDINACION TESIS

**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 INGENIERÍA
ESCUELA DE INGENIERÍA EN
CIENCIAS Y SISTEMAS
TEL: 24767644

*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* ***"DISEÑO DE UN MODELO DE INTEGRACIÓN CONTINUA UTILIZANDO JENKINS PARA PROYECTOS DE DESARROLLO EN UNA EMPRESA DE TELECOMUNICACIONES PARA EL MÓDULO DE CONTABILIDAD",*** *realizado por el estudiante, NORMAN EDUARDO CASASOLA GIRÓN aprueba el presente trabajo y solicita la autorización del mismo.*

*"ID Y ENSEÑAD A TODOS"*

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

*Ing. Marlon Antonio Pérez Türk*
***Director***
***Escuela de Ingeniería en Ciencias y Sistemas***

*Guatemala, 01 de abril de 2019*

Universidad de San Carlos
De Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG.173-2019

El Decano 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 del trabajo de graduación titulado: **"DISEÑO DE UN MODELO DE INTEGRACIÓN CONTINUA UTILIZANDO JENKINS PARA PROYECTOS DE DESARROLLO EN UNA EMPRESA DE TELECOMUNICACIONES PARA EL MÓDULO DE CONTABILIDAD"** presentado por el estudiante: **Norman Eduardo Casasola Girón** después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, se autoriza la impresión del mismo.

IMPRÍMASE.

Ing. Pedro Antonio Aguilar Polanco
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, Abril de 2019

/echm

## ACTO QUE DEDICO A:

| | |
|---|---|
| **Dios** | Sobre todas las cosas que, por su infinito amor y misericordia, me ha permitido realizar el desarrollo de este trabajo a pesar de los contratiempos y circunstancias de la vida, ya que cada dificultad es para un crecimiento persona y espiritual. |
| **Mi esposa** | A la mujer que Dios me dio para caminar juntos por este mundo, y llevar nuestro servicio a las personas que más lo necesitan. |
| **Mi papá** | Con su ejemplo de trabajo y lucha he aprendido que en la vida todo se puede lograr con un poco de empeño y disciplina. |
| **Mi mamá** | Quien me ha guiado espiritualmente por el camino recto que nos lleva hasta Dios, y poder servir a través de lo poco que tenemos y somos, a los demás. |
| **Mis hermanos** | Que a pesar de la distancia, siempre estaremos unidos a través del amor de hermanos. |

# AGRADECIMIENTOS A:


| | |
|---|---|
| **Dios** | Nuestro padre y creador, quien me ha guiado a través del Espíritu Santo para culminar este gran paso en el desarrollo académico, profesional y personal. |
| **Universidad de San Carlos de Guatemala** | Que, a través de sus catedráticos y mentores, he tenido la enseñanza necesaria para llegar hasta este nivel académico. |
| **Mi esposa Cristina Góngora de Casasola** | Por su amor y apoyo incondicional he logrado el cumplimiento de este paso en mi desarrollo académico. |
| **Mis padres Norman Casasola y Marielena de Casasola** | Quienes siempre lucharon para mi desarrollo académico enseñándome que, a través del estudio, y de las enseñanzas espirituales, llegaría a ser un hombre de bien. |
| **Mis hermanos Byron y Shandy Casasola** | Quienes siempre estuvieron, aunque de lejos, atentos y pendientes de este paso académico, y son parte del motivo por lo que lo he culminado. |

| | |
|---|---|
| **Mis amigos de la universidad** | Con quien es compartí todo el proceso de desarrollo académico, y aunque la vida nos haya llevado por caminos distintos, sé que cuento con cada uno de ellos. |

# ÍNDICE GENERAL

# ÍNDICE DE ILUSTRACIONES

## FIGURAS

# GLOSARIO

| | |
|---|---|
| **Agilidad** | Capacidad de generar un producto de software de calidad en el tiempo estipulado al inicio del proyecto en un ambiente de producción siguiendo una metodología de desarrollo. |
| ***Checkout*** | En términos de control de versiones, es crear una copia local de una versión de código fuente en un repositorio. |
| **Código fuente** | Líneas escritas por un ingeniero de software en un lenguaje de programación que representará el diseño solicitado inicialmente de un producto o servicio de software para convertirlo posteriormente en código objeto. |
| **Código objeto** | Lenguaje generado a través de la compilación de un producto de software que sea entendible por una computadora. |
| **Commit** | En términos de control de versiones, es colocar una versión local en un repositorio de control de versiones de código fuente. |

| | |
|---|---|
| **Contenedor** | Método de virtualización a nivel de aplicación permitiendo agilizar los despliegues de software, en una o más instancias configuradas previamente. |
| **Control de versiones** | Herramienta computacional para la gestión ágil del código fuente generado a través del tiempo por los distintos elementos del equipo de desarrollo de software y poder revertir cambios. |
| **Daily SCRUM** | En la metodología de desarrollo SCRUM, es una reunión diaria que se realiza en cada sprint por el equipo de trabajo, con una duración máxima de 15 minutos, para revisar el estado del proyecto. |
| **Despliegue** | Colocar el código objeto generado por el equipo de programación en un ambiente distinto al de desarrollo. |
| **GIT** | Herramienta de software diseñada para el control de versiones de código fuente. |
| **Java** | Lenguaje de programación orientada a objetos que no es dependiente de una plataforma, conocido como WORA por sus siglas en inglés (*write once*, *run anywere*) se escribe una vez, y se corre en donde sea. |
| **Ligereza** | Necesidad de generar un producto de software lo más pronto posible sin tomar en cuenta un proceso o |

| | |
|---|---|
| | flujo para su liberación en un ambiente de producción. |
| **Merge** | En términos de control de versiones, es el método que utilizan estas herramientas para unir cambios realizados por dos o más ingenieros de software sobre el mismo segmento del código fuente. |
| **Open Source** | Modelo de desarrollo de código abierto basado en colaboración abierta, con código fuente disponible para su uso y modificación. |
| **Plugin** | Aplicación que sirve como complemento para relacionar una herramienta de software con otra que hará uso de la misma, a través de una interface de comunicación. |
| **SCRUM** | Metodología de desarrollo basada en iteraciones ágiles y trabajo en equipo con una estrategia de trabajo incremental por cada proyecto. |
| **Servlet** | Clase de Java que permite la ejecución de aplicaciones en el servidor donde están alojadas. |
| **Sprint** | Metodología de desarrollo SCRUM, es el período en el cual se desarrolla un requerimiento de un producto o servicio de software previamente analizado y dividido en secciones. |

| | |
|---|---|
| **Sprint Backlog** | En la metodología de desarrollo de SCRUM, es el conjunto de requisitos ya analizados y segmentados para el siguiente *sprint*. |
| **Sprint Retrospective** | En la metodología de desarrollo SCRUM, es una reunión que se realiza al finalizar un sprint, para revisar lo bueno y lo malo realizado durante el proyecto, para mejora continua del equipo de trabajo. |
| **Test Case** | En la metodología de desarrollo SCRUM, son los casos de prueba básicos generados por el requirente del cambio para que le sirva al equipo de calidad, como pruebas mínimas para sus validaciones. |
| **Trigger** | En español, disparador; es un concepto utilizado para indicar que, a través de una acción, se ejecutará una o varias acciones desencadenadas por la primera. |
| **User Story** | Solicitud de un requisito o necesidad para un cambio o creación de un proyecto de software utilizada por metodologías de desarrollo ágil como SCRUM. |
| **Versión estable** | En la línea del control de versiones, es la versión desarrollada que se encuentra publicada en el ambiente de producción y que no cuenta con errores. |

# RESUMEN

El siguiente trabajo expone la importancia y la necesidad de adoptar una metodología de trabajo para equipos de desarrollo y operaciones, como DevOps, con el objetivo de llevar a la empresa a la realización de proyectos ágiles y de buena calidad, para tener un impacto menor sobre las operaciones de los usuarios finales de la aplicación.

También, se requiere que este trabajo se realice de una manera automatizada con la finalidad de cometer la menor cantidad de errores y de una manera rápida, utilizando la integración continua, a través de un orquestador como Jenkins en todo el proceso de desarrollo hasta su liberación en el ambiente de producción.

Se dará a conocer la importancia de tener un ambiente de control de calidad automatizado, gestionado a través del orquestador Jenkis como parte del proceso de integración continua y despliegue continuo; utilizando herramientas que de igual manera puedan automatizar todas las pruebas necesarias para entregar un producto de alta calidad con la menor cantidad de errores en el ambiente de producción en el tiempo estipulado al inicio del proyecto.

Finalmente, se realizará un diseño del flujo completo de un desarrollo de un cambio de un módulo de contabilidad para una empresa de telecomunicaciones, tratando de hacer el diseño lo más general posible para que pueda ser aplicado para cualquier tipo de desarrollo.

# OBJETIVOS

## General

Brindar una guía de conocimiento para la implementación de un proyecto de integración continua en un ambiente de desarrollo utilizando una herramienta de software especializada.

## Específicos

1. Brindar el conocimiento básico de un proceso de integración continua en la metodología de DevOps.

2. Integrar la herramienta de software Jenkins para facilitar el proceso de integración continua en un proyecto de desarrollo de software.

3. Incluir dentro de un proceso de integración continua, y dar a conocer la importancia del despliegue en control de calidad previo al lanzamiento en producción de un proyecto de desarrollo de software.

4. Crear una propuesta de un despliegue automatizado de soluciones de software para su puesta en producción, utilizando la herramienta Jenkins.

# INTRODUCCIÓN

A través del tiempo y la evolución del desarrollo de software con los distintos lenguajes de programación y las necesidades empresariales de desarrollar de manera cada vez más ágil sus propios módulos de software para cada área de la organización, se ha identificado la necesidad del uso de métodos de mejora continua, para optimizar los recursos de manera más eficiente.

Existen múltiples metodologías de desarrollo de software, unos más ágiles que otros, pero todos siempre con el mismo objetivo: desarrollar un software más robusto y estable con la menor cantidad de fallas en el menor tiempo posible.

La historia y la experiencia muestran una serie de fallas en el software al momento de implementar nuevos módulos o modificaciones en los mismos, que en su mayoría son a causa de errores humanos, probablemente por mala planificación, apresurarse en el lanzamiento, entre otros.

Al necesitar de estas metodologías, cada vez más ágiles, y más aún en empresas que el uso de la tecnología es vital, tal como las empresas de telecomunicaciones, existe en la necesidad de automatizar procesos, que eviten cada vez más las fallas por errores humanos y en otros tipos de errores, en una implementación de un nuevo o modificación de un módulo de software.

En la actualidad, existe una serie de herramientas de software que permiten automatizar estos procesos en el flujo de una implementación de

módulos de software; pero este trabajo de investigación se enfocará en el uso de Jenkins como orquestador en una implementación de software desde su desarrollo hasta su implementación, pasando por un ambiente de control de calidad.

# 1. EL MODELO DE DESARROLLO DEVOPS PARA LA GESTIÓN DEL DESARROLLO DE SOFTWARE Y SU PUESTA EN PRODUCCIÓN

## 1.1. DevOps en la ingeniería de software

A continuación, se muestra que es DevOps y los valores que podemos darle.

### 1.1.1. Qué es DevOps

En la ingeniería de software existen muchos modelos, metodologías, fases, etc., lo que le permite al ingeniero seguir procesos y procedimientos para la creación de un nuevo producto de manera ágil y efectiva para de esta manera entregarlo al consumidor final con la menor cantidad de fallas en un tiempo acordado entre las partes involucradas.

En este documento se darán a conocer las ventajas, desventajas y características de una práctica que permita integrar a equipos diversos en la solución de del proceso de desarrollo hasta su liberación y puesta en producción.

DevOps es la conjunción de dos palabras que representan a los equipos involucrados en el desarrollo de software, *developer* (desarrollador) y *operations* (operaciones).

El desarrollador o programador es aquella persona o grupo de personas que recibirán un requerimiento para convertirlo en una solución de software a través de una metodología y un lenguaje de programación.

Operaciones representa el área que consta de uno o más integrantes en una empresa de cualquier índole, que comúnmente pertenecen al área de IT y son los encargados de brindar el soporte necesario a la aplicación o producto de software ya liberado. Este equipo debe conocer el funcionamiento general de la aplicación, de cómo este fue integrado a los equipos tecnológicos de la empresa, el cómo debe ser configurado para el consumidor final. En operaciones se encuentra la línea especializada para la resolución de incidentes de la aplicación a la que este equipo les brinda soporte.

Existen más áreas en el proceso de desarrollo de software, pero más adelante en este documento se tratará sobre otra área que es de suma importancia para la liberación o puesta en producción de un producto de software: Quality Assurance (aseguramiento de la calidad o control de calidad).

Por lo tanto, DevOps es esta práctica o modelo de la que se habla, para poder integrar, no solo el desarrollo de software y su liberación, sino también a los equipos de trabajo involucrados en la búsqueda de entregar un producto de calidad y con la menor cantidad de defectos en el tiempo y con los recursos acordados en el inicio del proyecto.

### 1.1.2. El valor del modelo DevOps en la industria del software

En el mundo actual, la ingeniería de software es un elemento crucial para el mantenimiento, mejoramiento y crecimiento de las empresas, desde un portal

web o las redes sociales hasta un sistema empresarial complejo hecho a la medida.

Las necesidades comerciales del mercado son las que dan la pauta para que una empresa tenga que acelerar o desacelerar el desarrollo de nuevos productos o bien actualizar los ya existentes.

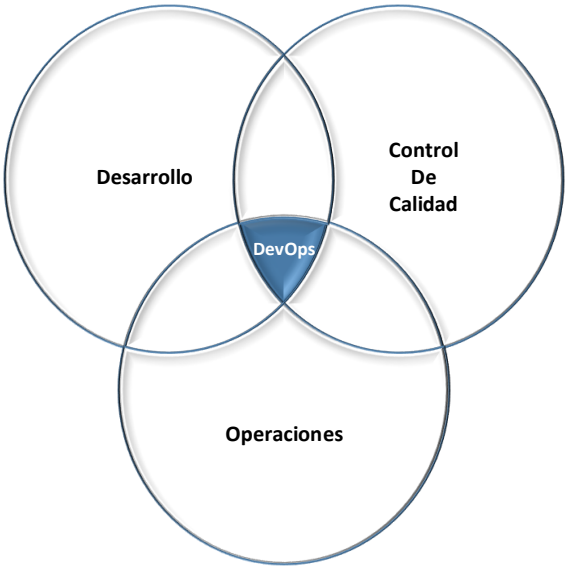
Estas empresas, sean pequeñas, medianas o grandes, deben contar con sistemas tecnológicos para competir con mayor agilidad y brindar un servicio especializado a los clientes; por lo tanto, requerirán de un experto en tecnología para crear, instalar, configurar y brindar soporte a las aplicaciones que brinden servicios a los clientes, ya sean internos o externos. Este servicio especializado puede ser contratado directa o indirectamente.

Teniendo esto en cuenta, el proceso de desarrollo de software también debe cambiar de una manera ágil, sobre todo para aquellas empresas medianas y grandes. Para ello es necesario que todos los cambios al software, ya sea por una mejora, un nuevo módulo en el sistema o una corrección al mismo, más allá de la agilidad, debe ser un producto o servicio de calidad, y para ello es necesario que los equipos involucrados en el proceso de su construcción y liberación estén coordinados.

El modelo de DevOps permitirá tener esa coordinación que se necesita utilizando herramientas tecnológicas que ayuden a minimizar el riesgo de un cambio, sobre todo en sistemas muy complejos, y ayudará a que los equipos de trabajo involucrados siempre estén comunicados de cada fase del proceso del desarrollo.

La intención de DevOps es brindar las herramientas necesarias y que puedan ser adaptables para que los equipos de desarrollo, QA y operaciones trabajen en conjunto como un mismo sistema para lograr el mismo objetivo.

Figura 1. **Marco de trabajo, DevOps**




Fuente: elaboración propia.

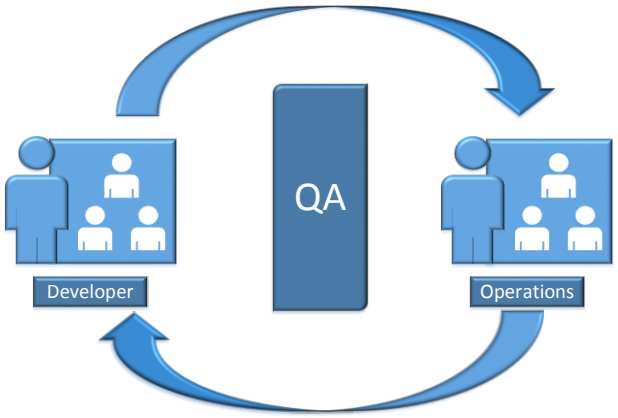
El modelo de DevOps es un ciclo iterativo, que como se verá más adelante, conlleva a una mejora continua, tanto de la construcción de nuevos productos de software, como de integración de los equipos de trabajo. Para ello es necesario saber que, en un producto ya existente, uno de los insumos para su mejora continua es la retroalimentación del equipo de operaciones.

Se sabe muy bien que este equipo de operaciones es el que está enlazado a la continuidad del negocio, y conoce plenamente la interacción del sistema con los clientes, tanto en las necesidades de mejora, como en las fallas que se van dando a través del tiempo. No se olvide que dentro de este marco

de trabajo, se encuentra QA, quien realizará las revisiones respectivas dentro de los nuevos requerimientos y modificaciones sugeridos por el equipo de operaciones.

Más adelante se ampliarán las etapas del desarrollo y las operaciones y como estas se van integrando al modelo. Por el momento se presenta la siguiente figura, que representa esta interacción entre los equipos.

Figura 2. **Ciclo de vida DevOps**




Fuente: elaboración propia.

## 1.2. El modelo de DevOps para la unificación del desarrollo de software y la puesta en producción

Como ya se ha mencionado anteriormente, la intensión es liberar a producción un producto o servicio tecnológico lo más rápido posible y con la menor cantidad de errores. Es por eso, por lo que la comunicación entre los equipos involucrados debe ser transparente, concisa y a tiempo.

Deben existir herramientas de comunicación en ambas vías: desarrollo -> control de calidad -> operaciones y de regreso, operaciones -> control de calidad -> desarrollo. con esta comunicación básica se puede decir que estaríamos cumpliendo lo que pide el modelo de DevOps.

Se Hablará del tipo de comunicación que debe existir entre cada uno de los equipos.

Construir un tablero de comunicación entre los equipos de DevOps.

Con esto se observa la importancia de este elemento clave en el modelo: es la comunicación. Si no se cuenta con este elemento, aunque tengamos las mejores herramientas tecnológicas en todo el proceso, los riesgos de brindar un mal servicio o crear un producto defectuoso se ven incrementados.

### 1.3. El marco de trabajo de DevOps CAMS

Para trabajar con este modelo, se debe entender que no solo es la adopción de procesos, y procedimientos elaborados de manera mecánica, la que llevará a lograr una madurez en un marco de trabajo de DevOps, sino que requiere un cambio en la mentalidad y la forma de trabajar, de todos los equipos involucrados en el proceso.

En la práctica, en las grandes empresas y corporaciones siempre existe un celo por las actividades realizadas en las distintas áreas, es por ello que es difícil cambiar esta mentalidad en un ambiente de trabajo donde siempre se ha trabajado aisladamente. Esto se realizará a través de herramientas que irán ayudando a gestionar de mejor manera la comunicación, sin entrar en conflicto

con las personas, sino mejorando los procesos y procedimientos con los que se trabajará en conjunto.

Para ello se profundizará un poco más en el marco de trabajo que se debe adoptar para llegar más allá de un simple procedimiento mecánico, a través de un conjunto de elementos que ayudarán a establecer de una manera más precisa y sencilla la relación entre los equipos de trabajo en la adopción de DevOps. Lo primero que se debe entender es el marco de trabajo sobre el cual DevOps funciona y este es llamado CAMS[1] por sus siglas en inglés, *culture* (cultura), *automatization* (automatización), *measure* (medición) y *share* (compartir).

### 1.3.1. Cultura

El primer elemento es la cultura, que se refiere a la comunicación entre las personas y lo primero que hay que hacer es romper barreras de trabajo entre los equipos.

Con base en la experiencia, mucho del tiempo dedicado a la liberación de un nuevo producto de software es por no ponerse de acuerdo en las distintas reuniones de trabajo, dependiendo de la metodología que se utilice, también en una gran cantidad de requerimientos de cambios que se quedan en la cola, o en la gran cantidad de documentación solicitada para liberar un pequeño cambio.

Para que la cultura tenga éxito, debe existir dentro del proceso el apoyo de las altas autoridades y trabajar de cerca con los equipos de trabajo, para ir despejando caminos y tomar decisiones inmediatas para liberar la

[1] DevOps dictionary wiki. *CAMS*. http://devopsdictionary.com/wiki/CAMS. Consulta: 14 de enero de 2019.

comunicación; es necesario también definir los procesos de comunicación de tal manera que agilicen el paso del producto entre equipos sin tantos elementos burocráticos. Esto se logrará solamente con el uso de herramientas de tecnología que lleven el control de cada uno de los pasos. Para ello se propondrán más adelante herramientas que soporten esta fase de comunicación.

### 1.3.2. Automatización

El siguiente elemento para la adopción de este marco de trabajo es la automatización, tanto de la comunicación, como de herramientas que ayuden a poner en producción una nueva fase del proyecto, llevando el control de las versiones desarrolladas, testeadas y liberadas en producción.

Este elemento ayudará a reducir el tiempo y las fallas, al momento de lanzar a producción un nuevo producto de software. Al principio, la automatización es un proceso que puede volverse tedioso y cansado de investigación ya que se tienen que definir cuáles serán las mejores herramientas para el tipo de trabajo que se desempeña. Es necesario contar con infraestructura y tiempo de instalación de estas herramientas, tanto como tiempo para el aprendizaje por parte de todos los elementos involucrados en cada una de las etapas en las que se irá automatizando. De igual manera, a través del desarrollo de este tema de investigación se estarán proponiendo una serie de herramientas para cada etapa de este modelo.

### 1.3.3. Medición

Como ya se mencionó anteriormente, el modelo de DevOps va amarrado al término de mejora continua; esto significa que a pesar de que un producto o

parte de un producto ya fue liberado a producción, este no debe quedar allí, tiene que mejorar. Para ello es necesario llevar un control de cada una de las fases de este ciclo, creando los indicadores necesarios que den información de utilidad y en tiempo para ir mejorando cada proceso.

Es importante recalcar que hay que tener mucho cuidado en la definición de los indicadores que estarán ayudando para mejorar cada fase de este ciclo, ya que, si se utiliza excesivamente el uso de estos indicadores, podría llegar a ser contraproducente, causando retrasos en la liberación de un nuevo producto.

En cada etapa del proceso es necesario medir tiempos, tanto de las personas como de los procesos ya automatizados; estas mediciones deberán ser compartidas entre todos los equipos de trabajo para que entre ellos se apoyen a mejorar los aspectos que puedan estar retrasando la liberación de un nuevo producto. Es muy importante, como se mencionó anteriormente, la colaboración de todos los equipos, para que exista la mejor comunicación entre ellos.

### 1.3.4. Compartir

El éxito de cualquier organización al utilizar DevOps es el compartir los hallazgos, los descubrimientos, las lecciones aprendidas, a través de las buenas prácticas y de los errores cometidos.

Esta parte es muy complicada, ya que, como se mencionó anteriormente, existe un celo sobre las actividades que cada área realiza. Pero para despejar el camino, y realizar esta integración con mayor flexibilidad, es necesario que las altas autoridades de la organización se involucren en este proceso de

cambio, y también se cuente con las herramientas concretas y los procedimientos adecuados para que esto se lleve a cabo.

En el numeral de medición se habla de que es necesaria la creación de ciertos indicadores relevantes que ayuden a ir mejorando el proceso de mejora continua, por lo que en la comunicación, todos estos indicadores deben ser compartidos a todos los equipos involucrados, para ello se puede brindar acceso a los registros y reportes a cada parte interesada. Es muy importante también dar a conocer cuáles son los umbrales mapeados por cada equipo a los demás, para que no ocasione malentendidos sobre los mismos. Es muy importante que unos equipos deben apoyar a los demás, brindando retroalimentación específica de los procesos y no poniendo trabas a los demás para que no logren avanzar.

# 2. INTEGRACIÓN CONTINUA EN UN MODELO DE DESARROLLO DE SOFTWARE

## 2.1. El modelo de integración continua

"Es una práctica de desarrollo de software donde la calidad y la integridad, se mantienen a través de pruebas y corrección de errores después de cada integración."[2]

"Este es un modelo informático que según Martin Fowler quien lo propone como una práctica, donde los miembros de un equipo integran su trabajo frecuentemente; usualmente cada persona lo realiza al menos una vez al día, destacando múltiples integraciones al día."[3]

Es muy importante destacar que, en el proceso de nuevas liberaciones, a medida que se vayan generando nuevas integraciones de software a un producto ya existente es más difícil llevar el control; por lo tanto, se propone automatizar los indicadores de cambio en cada una de las etapas del desarrollo del software hasta su implementación.

La intención de este trabajo es encaminar al lector en el uso de un modelo de integración continua, a través de la automatización de cada una de sus fases, en el cual se propondrán una serie de herramientas para realizarlo. Es importante dar a conocer que no se mostrarán todas las herramientas posibles

[2]UDEMI. *Learn continuous integration with Jenkins all in one guide*. https://www.udemy.com/learn-continuous-integration-with-jenkins-all-in-one-guide. Consulta: 19 de enero de 2018.
[3] FOWLER, Martín. *Continuous Integration*. https://www.martinfowler.com/articles/continuousIntegration.html. Consulta: 19 de enero de 2018.

que se utilizan en el mercado para este modelo de integración continua, sino que será específico en proponer cada herramienta para cada fase de un modelo de desarrollo de software.

Se destacarán dos grupos dentro del modelo de integración continua y es en el que se basará para el resto de este trabajo. desarrollo continuo y despliegue continuo, como se ve a continuación.

### 2.1.1. Desarrollo continuo

Para un ingeniero de software es muy importante llevar el control de los cambios que se estén realizando sobre un producto de software, y necesita agilizar los procedimientos de dichos cambios sin que este afecte al resto del equipo de desarrollo y también es importante que todos tengan el conocimiento y el control de lo que otro desarrollador esté realizando; sobre todo si se están utilizando las mismas clases y/o módulos del software.

En la actualidad, existen diversas metodologías de desarrollo de software, las cuales ayudan a entregar un software de mejor calidad en los tiempos establecidos; pero para el caso de este proyecto, se tratará de enfocarse en una herramienta que ayude en un proceso de desarrollo ágil; se entendiende por desarrollo ágil, aquel que puede ser entregado una o más veces en el día, con la menor cantidad de errores o fallas posibles.

Es muy importante entender la diferencia entre agilidad y ligereza en un modelo de desarrollo de software. Ligereza se refiere a la entrega de una solución lo más pronto posible, aún si no cumple con la totalidad de las pruebas de calidad para su liberación a producción, esto por la necesidad de un cambio por un error mayor que esté ocurriendo en producción; agilidad se refiere a la

realización completa del flujo de despliegue, pasando por la totalidad de las pruebas, que en su mayoría deben estar automatizadas.

En la ligereza se debe tener mucho cuidado ya que, al momento de liberar el producto de desarrollo, se puede caer en un error o falla mayor al que ya se tenía en producción; en la agilidad, se pasa por todo el flujo de liberación del proyecto, lo que incluye las pruebas totales del software a liberar, esto significa que el riesgo a un error o falla es muy bajo. Para ello todas, o la mayoría de las pruebas deben estar automatizadas. Más adelante se irá profundizando en cada uno de estos temas.

En las empresas grandes, la necesidad de mejoras y cambios en el software ya existente, como en la creación de nuevos productos de software, puede llegar a ser alta, lo que podría incurrir a cometer errores en cada despliegue si no se cuentan con los mecanismos y la metodología necesaria para su implementación.

Cuando se habla de desarrollo, se refiere a la codificación en un lenguaje de computadora de las necesidades del negocio y comerciales para una empresa en particular, esto incluye a sus equipos de trabajo, quienes se encargan de plasmar estas necesidades y hacerlas realidad. Esto incluye un previo análisis y diseño de lo que se necesita implementar. Este desarrollo debe contar con un control de versiones del código fuente de computadora que se está generando al cual todo el equipo involucrado en dicho desarrollo debe tener acceso.

Este control de versiones debe automatizarse a través de alguna herramienta que permita la agilidad y la comunicación constante, debe llevar el control de los distintos proyectos que se estén trabajando, debe brindar el

acceso necesario a los involucrados en cada proyecto. Debe tener la capacidad de la resolución de conflictos de codificación de las mismas clases o estructuras a las cuales tienen acceso el equipo de desarrollo, como regresar a versiones anteriores en caso de que la nueva versión tenga fallas. Debe tener la capacidad de integrarse a otras herramientas de software para tener interacción en las otras etapas del desarrollo para lograr una implementación, pasando por pruebas hasta llegar a producción.

El equipo de desarrolladores debe utilizar metodologías de desarrollo que les permitan interactuar entre sí, y en conjunto con los equipos de pruebas y operaciones (operación y mantenimiento de la plataforma de producción), la cual pueda ser integrable a un modelo de integración continua.

Más adelante, se hablará de estas herramientas para el control de versiones que permitirán que se lleve a cabo un correcto y eficiente desarrollo continuo.

### 2.1.2. Despliegue continuo

El objetivo de un ingeniero de software es entregar un producto de calidad en el tiempo acordado en un ambiente de producción, con toda la información necesaria de configuraciones para que el equipo de soporte sea capaz de brindar el seguimiento adecuado junto con el cliente y de esta manera retroalimentar a partir de sus beneficios y deficiencias.

Cuando ya se cuenta con un producto finalizado y revisado adecuadamente, el siguiente paso es lanzarlo al ambiente de producción y que empiece a generar valor para la organización y/o empresa a la cual se está poniendo en servicio.

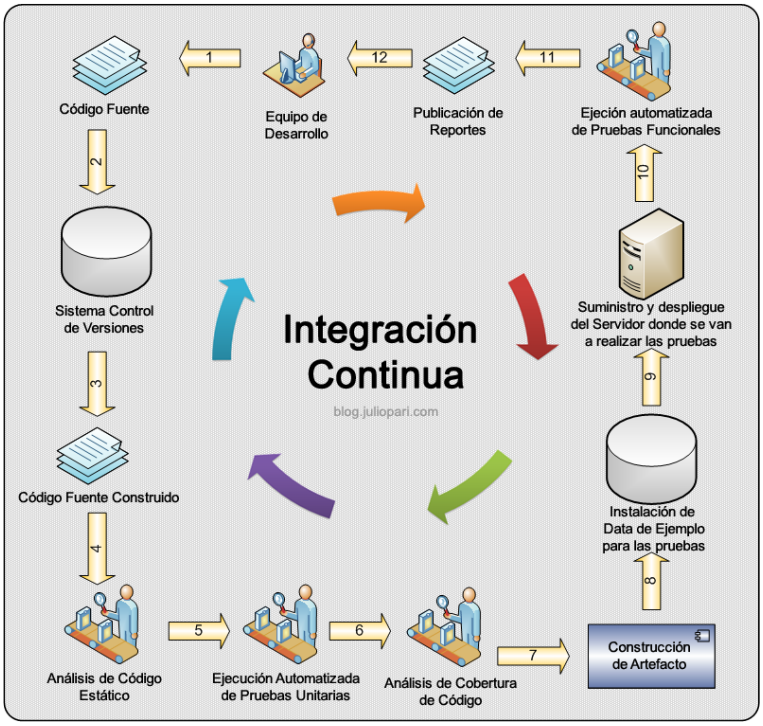
Si el equipo de desarrollo cuenta con herramientas automatizadas para agilizar el proceso de desarrollo y pruebas, también se necesitan herramientas que permitan poner en producción dichos cambios, mejoras o nuevos productos en los ambientes de producción sin poner en riesgo o minimizar afectación para la continuidad del negocio.

Este proceso para poner un producto en el ambiente de producción se le llama despliegue continuo, el cual será configurado específicamente para las necesidades de cada organización o empresa, dependerá de las necesidades de cambio; si es necesario realizar los cambios durante los períodos de operación o si es factible esperar horarios no productivos de cada empresa. También, dependerá de la arquitectura sobre la cual está diseñada la aplicación o aplicaciones de software, y se deberán analizar modificaciones sobre esta arquitectura para que paulatinamente se puedan realizar o mejorar los despliegues de los productos de software a producción.

## 2.2. Procedimientos y pasos para un modelo de integración continua

A continuación, se describirá una serie de elementos según el modelo propuesto por Martin Flower y la publicación realizada en devopsti.wordpress.com en 2014 para entender los pasos para la integración continua.

Figura 3. **Modelo de integración continua**




Fuente: PARI, Julio. *Integración continua en proyectos*. http://blog.juliopari.com/integracion-continua-en-proyectos-agiles-de-software/. Consulta: 29 de julio de 2018.

Como ya se mencionó anteriormente, existen muchas herramientas que ayudarán a automatizar cada una de las fases de la integración continua, hasta su puesta en producción, pero el enfoque de este trabajo es la definición de un modelo. Pero para este caso, se describirán herramientas en particular como modo de ejemplificación, con el entendido que puede utilizarse otras herramientas en cada una de las fases.

Dentro de las herramientas de ejemplo para ir definiendo de manera práctica las definiciones de los temas, se tomará como lenguaje de programación JAVA, y como metodología de desarrollo, se utilizará el marco de

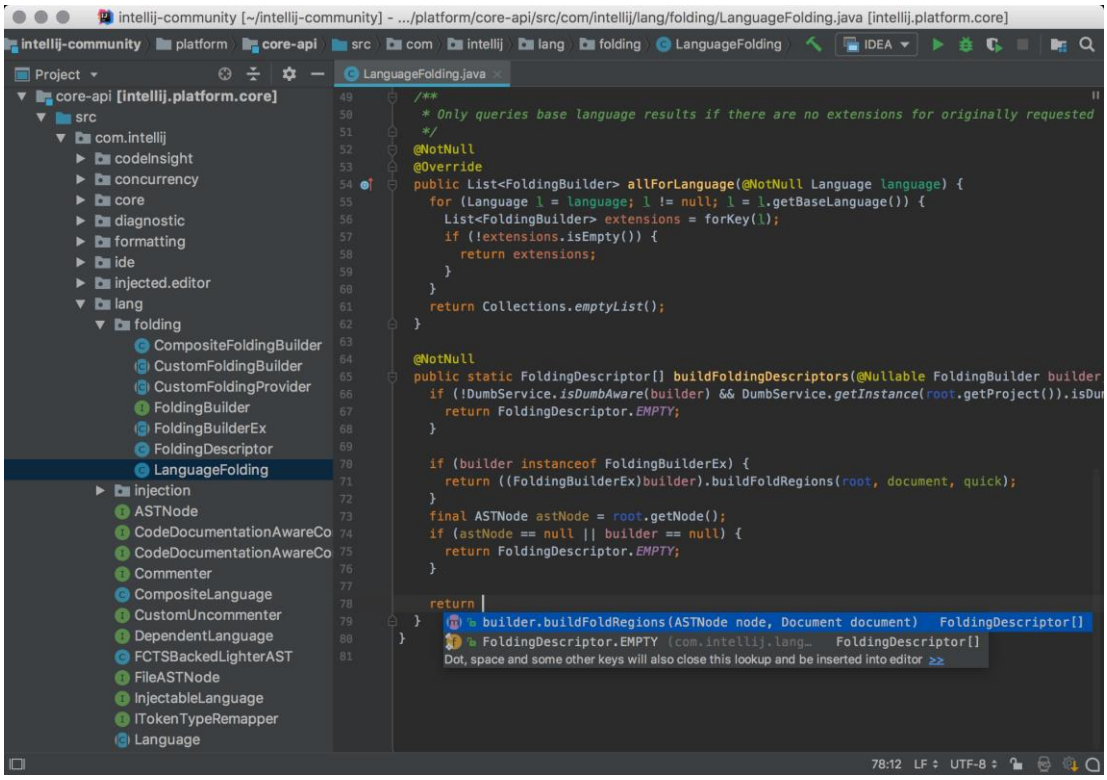
trabajo con SCRUM. A través de los temas se irá ampliando el lenguaje de programación y sobre la metodología a utilizar, pero para entender un poco más desde el principio daré una breve definición de cada uno: JAVA, es un lenguaje de programación orientado a objetos, que es codificado una vez y que puede ser ejecutado en distintos sistemas operativos gracias a sus múltiples máquinas virtuales, que incluye dispositivos móviles. SRUM es una metodología de desarrollo para proyectos ágiles y que cuenta con las herramientas para trabajar de una manera colaborativa entre los distintos equipos de trabajo y de esta manera obtener los mejores resultados en el menor tiempo posible.

Tomando en cuenta estas premisas de este trabajo, se continuará con los siguientes elementos que forman parte del modelo de integración continua.

### 2.2.1. Generación del código fuente

Luego de que un proyecto de software ya fue definido y consensuado con los equipos solicitantes y el equipo de desarrollo, se inicia con el proceso de generación de código fuente para lo cual se propondrá el IDE de desarrollo IntelliJ, aún que existen varios en el mercado (NetBeans, Eclipse, entre otros).

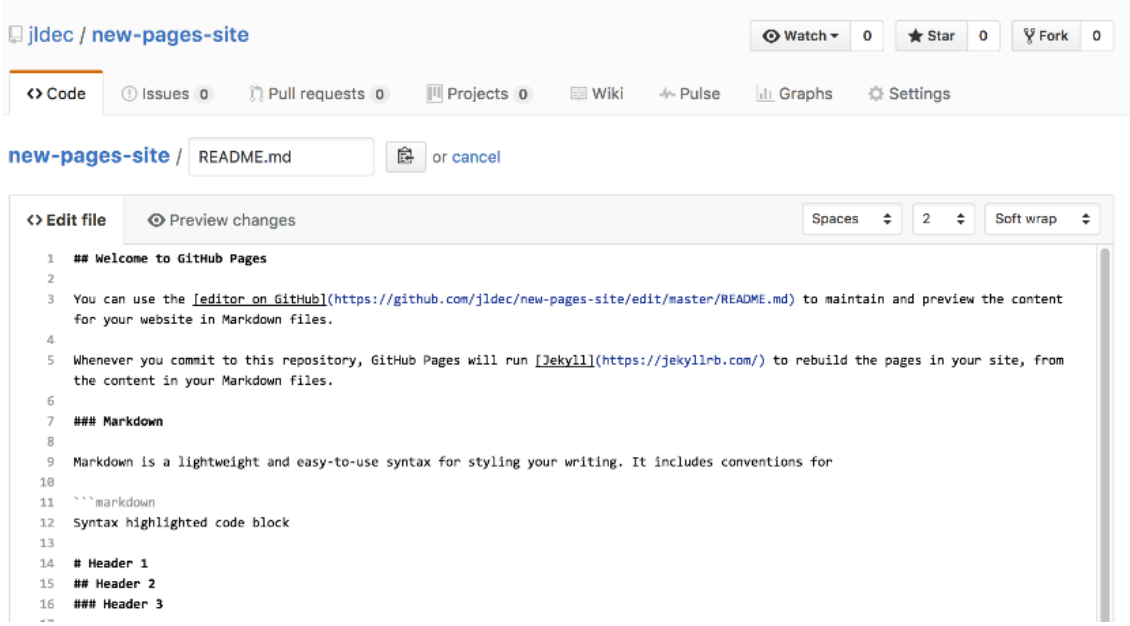
Figura 4. **IntelliJ IDEA**




Fuente: JetBrains. *IDE capaz y ergonómico para JVM*. https://www.jetbrains.com/idea/. Consulta: 29 de julio de 2018.

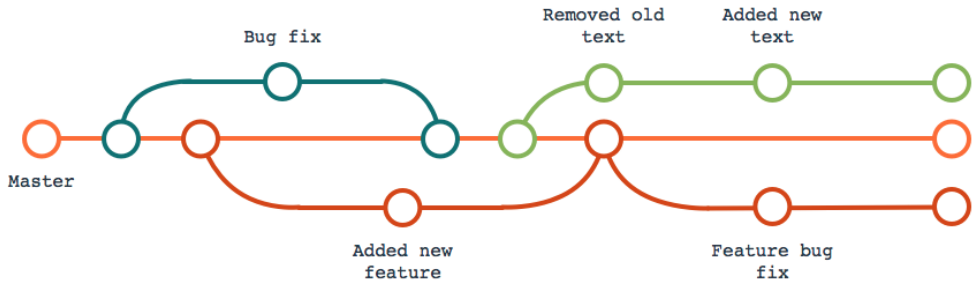
Para el control del código fuente se necesita un repositorio que lleve el control de versiones; para este caso se utilizará GitHub, aunque existen otros (SubVersion, TeamWare, entre otras).

Figura 5. **Repositorio Git**




Fuente: jldec. *Sitio de nuevas páginas*. https://pages.github.com/images/code-editor@2x.png. Consulta: 29 de julio de 2018.

Figura 6. **Modelo general de control de versiones**




Fuente: cPanel. *Introduciendo el control de versiones Git*. https://blog.cpanel.com/git-version-control-series-what-is-git/. Consulta: 29 de julio de 2018.

Se puede iniciar un nuevo proyecto creando la estructura dentro de GitHub llamado repositorio para llevar el control de los cambios de código a través de las versiones y las distintas líneas de programación llamadas ramas. Algo en lo que hay que enfatizar, es la importancia de la documentación, ya que como se mencionó anteriormente, al trabajar con integración continua en un modelo de DevOps, la comunicación es importante; y al mantener cada cambio documentado, el resto de los integrantes de los equipos sabrán exactamente qué fue lo que se realizó en un cambio o en un nuevo proyecto.

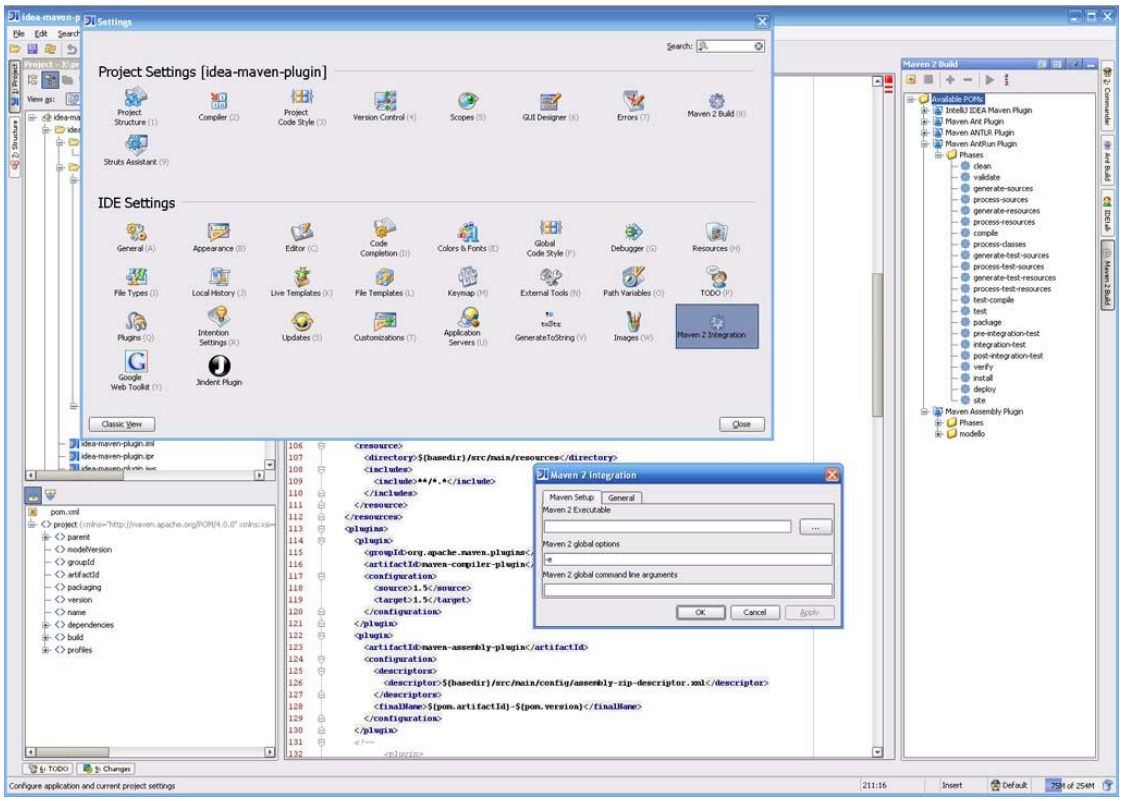
Si el proyecto ya existe, solamente es necesario gestionar una rama en el repositorio del proyecto y descargar la última versión estable para realizar las modificaciones y al finalizar los cambios; nuevamente, se realiza un versionamiento del cambio para que pueda ser llevado a pruebas antes de colocarlo como la última versión estable, en la rama principal llamada Master.

### 2.2.2. Compilación del código

El siguiente paso después de haber versionado los últimos cambios, es proceder con la compilación del código fuente para realizar las pruebas respectivas; pero antes de pasar al ambiente de pruebas, es necesario realizar verificaciones de código, pruebas unitarias sobre el cambio en un ambiente controlado por el desarrollador.

Para la automatización de la compilación del código fuente, se puede utilizar una serie de herramientas como Ant, Maven, Gradle, entre otros.

Figura 7. **Maven Plugin para IJ (IntelliJ)**




Fuente: Plugins jetbrains. *Captura de pantalla 299 png*. https://plugins.jetbrains.com/files/1166/screenshot_299.png. Consulta: 29 de julio de 2018.

Cuando se habla de la compilación del código fuente, se refiere al proceso de creación de los archivos ejecutables que servirán para la ejecución del programa que se está creando o modificando con un lenguaje entendible por la computadora, llamado generación de código objeto o código de máquina. Con Maven se puede automatizar este proceso, configurándolo una sola vez, para que cada vez que se necesite automáticamente pueda ser generado el código máquina y proceder con el siguiente paso del flujo.

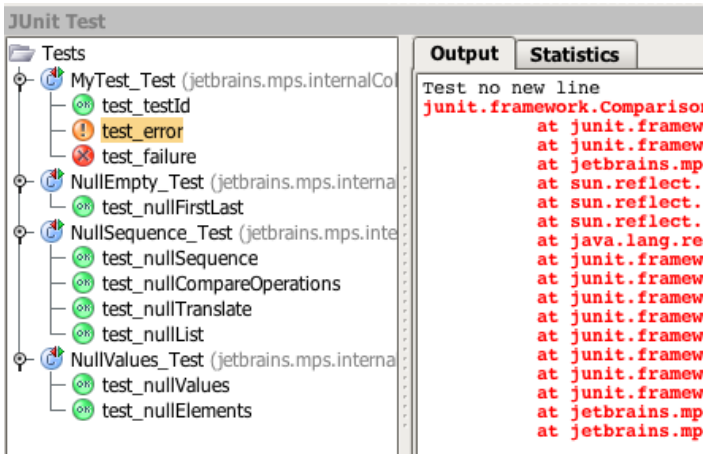
Previo a la compilación se puede realizar un análisis de la codificación realizada por el desarrollador, automatizando las pruebas de código con alguna herramienta como SonarQube o Simian. De esta manera se puede encontrar fallas que puedan afectar el producto final generado que evitan ciclos infinitos, código duplicado, mal uso de operadores, entre otros. De esta manera se puede también optimizar los tiempos de respuesta.

### 2.2.3. Análisis del código compilado

Previo a pasar el o los archivos ya compilados a un ambiente de pruebas, es necesario que el desarrollador genere los resultados de un juego de pruebas básicas, llamadas pruebas unitarias; también, un análisis estático de software y un análisis de cobertura de código.

Se puede utilizar una o varias herramientas para la automatización de estas pruebas FindBugs, JUnit, Check Style, Selenium, SoapUI entre otros. Estas pruebas permitirán revisar que el desarrollo que se ha creado no tenga fallas según los requerimientos iniciales del software. También, se puede validar a través de estas pruebas que los resultados no afecten a otra sección de la aplicación en el ambiente de producción.

Figura 8. **Resultados en JUnit**




Fuente: JUnit test. *Modificación de datos*. https://confluence.jetbrains.com/download/attachments/40207/junit-run-3.PNG?version=1&modificationDate=1209534692000&api=v2. Consulta: 10 de junio de 2018.

De esta manera se puede verificar si el código generado tiene alguna falla sobre la aplicación a la que se implementará. Si se encuentran fallas, estas generan resultados negativos los cuales indicarán al desarrollador un cambio en el código fuente y, por lo tanto, una nueva versión de este.

### 2.2.4. Construcción de ambiente de pruebas

Al obtener resultados satisfactorios por parte del desarrollador, se inicia con una nueva etapa en el flujo de la integración del software, como la construcción de un ambiente de pruebas.

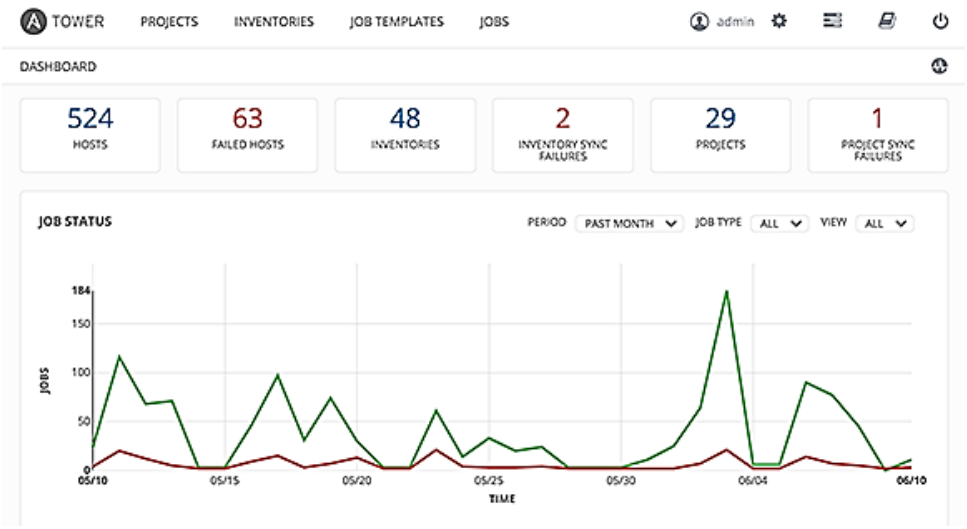
Este ambiente debe ser construido y preparado con los valores de las configuraciones del ambiente de producción para tener la certeza o lo más cercano a la verdad, de que si las pruebas que aquí se realicen fueron

satisfactorias, no se tendrá fallas en el ambiente de producción. Para la automatización de creación de ambientes se puede utilizar Ansible, Gradle, entre otros.

Al utilizar Ansible, como herramienta para la construcción del ambiente de pruebas, se debe crear una plantilla con todas las características básicas que se necesitan y desplegarla en uno o más servidores y servicios en donde será ejecutada la aplicación en pruebas. Esta plantilla es llamada PlayBook.

Existen herramientas para dashboards que se integran a Ansible para tener una visibilidad gráfica de lo que se está generando como DataDog, Tower, entre otros.

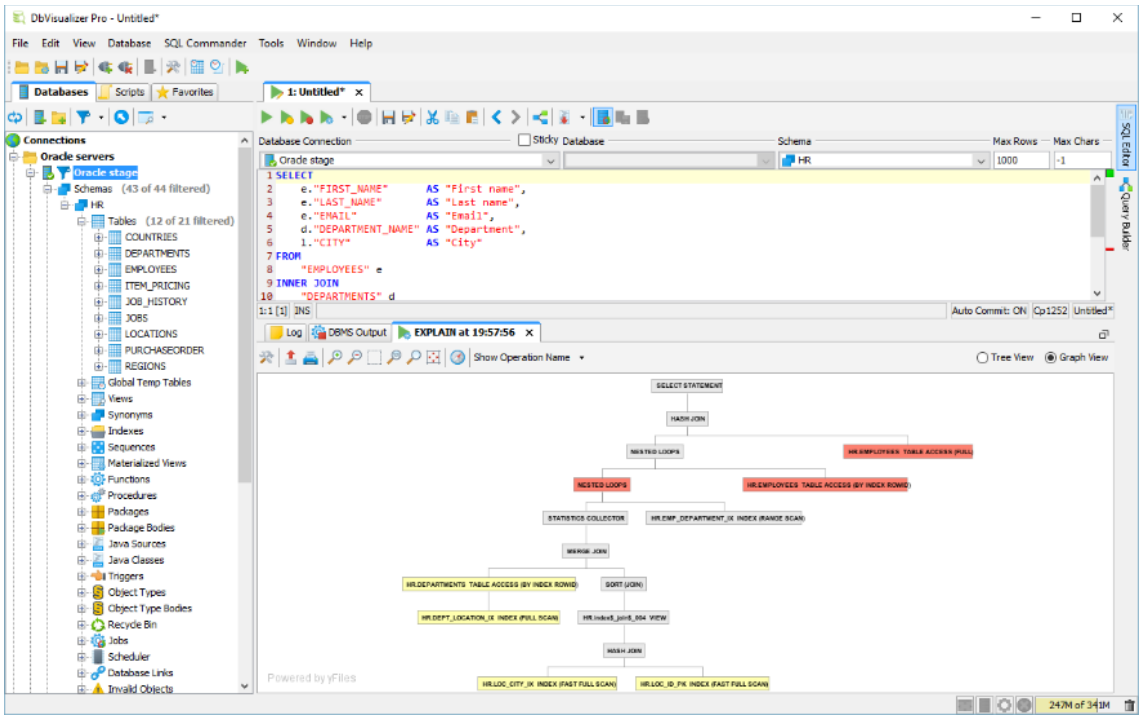
Figura 9. **Tower para Ansible**




Fuente: Redhat. *Archivos gestionados*. https://www.redhat.com/cms/managed-files/Ansible-Tower3-dashboard.png. Consulta: 10 de junio de 2018.

La base de datos para el ambiente de pruebas debe ser preparada con las configuraciones y los catálogos de información que se encuentran en ese preciso momento en producción; adicionalmente, contar con una serie de datos de prueba de las tablas transaccionales como parte de esta preparación; para la automatización se puede utilizar DbVisualizer entre otras herramientas.

Figura 10. **DbVisualizer**




Fuente: dbvis. *Imágenes, características y pantallas*. https://www.dbvis.com/images/features/screens/explainPlanGraph.png. Consulta: 10 de junio de 2018.

### 2.2.5. Ejecución de pruebas

La siguiente fase es la ejecución de pruebas. Ya habiendo establecido el ambiente de pruebas, el equipo encargado de las mismas da inicio a realizar las

configuraciones necesarias para la automatización de estas. Este paso es muy importante ya que de los resultados de esta etapa depende que el software pueda ser liberado a producción o regrese al equipo de desarrollo para realizar los ajustes necesarios. Siempre se debe tener en cuenta que uno de los objetivos es entregar un producto de calidad; también, se cuenta con un tiempo establecido para la entrega. En la planificación se debe tomar en cuenta el tiempo necesario para la ejecución de las pruebas y estar preparados por si es necesario que regrese al equipo de desarrollo para corregir las fallas encontradas en esta etapa.

Dependiendo del tipo de software, algunas de las pruebas deberán ser manuales, sobre todo cuando es un proyecto nuevo, ya que aún se están estableciendo los parámetros y las configuraciones del software que está próximo a ser liberado en el ambiente de producción.

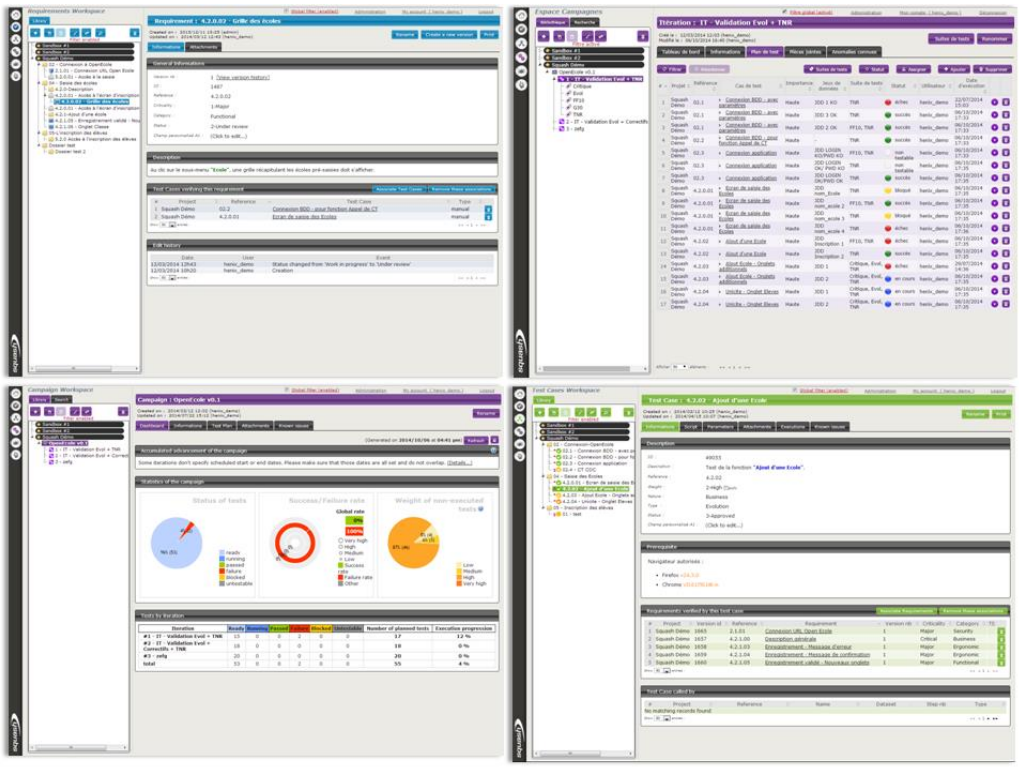
Existen diversas herramientas para la automatización de estas pruebas, muchas de ellas es necesario realizar procesos de programación para llevarlas a cabo. Es muy importante dar a conocer que las pruebas pueden realizarse desde diversos puntos de vista, y en su mayor parte, todos son necesarios, pero se hablará de algunas herramientas que son *open source*.

Antes de iniciar con las pruebas es necesario generar los casos de prueba (*test cases*), que son casos de uso específicos para pruebas operativas y funcionales de la aplicación. En la metodología de desarrollo Scrum, los casos de prueba vienen desde el requerimiento inicial a junto con la User Story, pueden ser uno o más casos de prueba, entre más casos de prueba se definan, mejor oportunidad a un buen análisis de las pruebas se podrá tener. Es muy importante mencionar que los casos de prueba no son un parámetro completo de que el software que se está revisando sea completamente funcional; por ello,

el ingeniero de software que esté encargado de dichas pruebas realice un análisis más profundo de las pruebas que se necesitan para asegurar un buen producto a liberarse en producción.

Se iniciará indicando que se necesita una herramienta que gestione todo el flujo de pruebas, que lleve el control de los resultados, como Squash, Bugzilla, Test Link, Data Generator, entre otros. Es muy probable que sea necesario más de una herramienta de gestión dependiendo las necesidades de las pruebas.

Figura 11. **Uso de Squash para casos de prueba**



Fuente: Squashtest. *Imágenes Squash TM1*. https://www.squashtest.org/images/Squash_TM1.png. Consulta: 8 de agosto de 2018.

Dentro de las pruebas necesarias para la entrega de un producto o servicio con buen funcionamiento se puede mencionar algunas; más adelante en este trabajo se tratará más a profundidad el tema del control de calidad en los proyectos de software; solamente se mencionarán cuáles son las que deberían al menos realizar:

- Pruebas de software funcionales
- Pruebas de software no funcionales
- Pruebas de seguridad
- Pruebas de software estructurales
- Pruebas de regresión
- Pruebas de carga y rendimiento

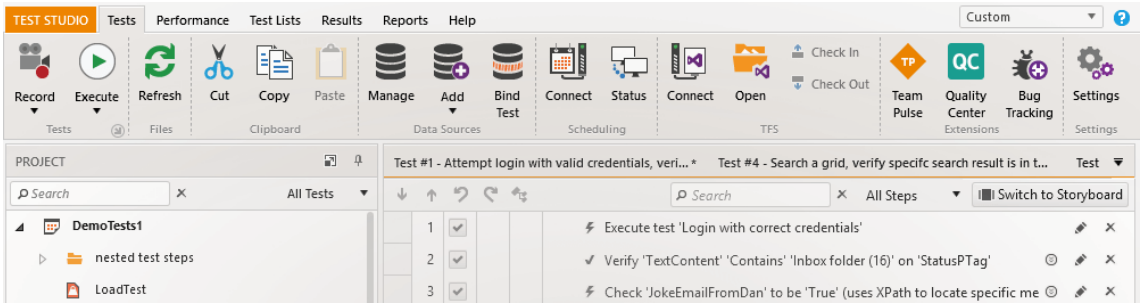
### 2.2.6. Generación de informes

En esta última sección es donde se toma la decisión de preparar el producto o servicio de software para ser desplegado al ambiente de producción o de ser enviado nuevamente al equipo de desarrollo para la corrección de las fallas encontradas durante el análisis de calidad del software.

Cada una de las herramientas que se describió en cada una de las secciones anteriores, puede generar un informe técnico final, que indicaría el éxito o fracaso de las pruebas. Pero como se indicó al inicio, es necesaria una herramienta de gestión de pruebas, por ejemplo Squash, la cual generará esos informes que se necesitan para retroalimentar a los equipos, ya se el equipo de soporte quien recibirá la solución final o el equipo de desarrollo quien realizará las correcciones necesarias.

También, es factible utilizar una herramienta como Test Studio, que cuenta con una amplia solución para generación de todas las pruebas que se necesitan y generará los informes necesarios que ya se mencionaron.

Figura 12. ***Test studio* (Telerik)**




Fuente: Test studio. *Frente a la nube sfimages*. https://d585tldpucybw.cloudfront.net/sfimages/default-source/labs/test-studio/demo-header.png?sfvrsn=e6648aac_3. Consulta: 29 de agosto de 2018.

# 3. JENKINS COMO HERRAMIENTA DE INTEGRACIÓN CONTINUA PARA PROYECTOS DE DESARROLLO DE SOFTWARE

## 3.1. Integración continua con Jenkins

En el transcurso de este documento se han tratado diversos temas, como DevOps, una metodología de trabajo entre distintos equipos involucrados en el proceso del desarrollo del software, hasta su puesta en producción. También, se trató el tema de integración continua, que es el proceso de llevar de una manera automatizada cada una de las fases del desarrollo del software, hasta prepararlo para su puesta en producción a través de un despliegue continuo, de igual manera automatizado.

Estos dos temas tienen el objetivo de agilizar todo el proceso de creación de un producto o servicio de software hasta colocarlo en un ambiente de producción en el menor tiempo posible con la mejor calidad.

Se describió también, que en la metodología de integración continua existen varias fases o secciones para llevar a cabo la solución de software de una manera adecuada y ágil desde su concepción hasta su liberación como producto o servicio final. Cada una de estas fases o secciones se introdujeron algunas herramientas para automatizar de la mejor manera el flujo de un proyecto de software y agilizar su salida a producción, pero cada una de las herramientas tiene una entrada y una salida que requiere múltiples configuraciones e intervención humana para ir saltando de fase en fase.

Por lo que en este documento se propone trabajar con una herramienta que se encargue de integrarlas a todas desde un solo gestor; esta herramienta es Jenkins, que funcionaría como el director de orquesta, que dirige a cada grupo de personas con distintos instrumentos musicales, es por ello por lo que a esta herramienta se le conoce como orquestador.

Más adelante se irá conociendo poco a poco esta herramienta, cómo funciona en los proyectos de software, cómo gestiona las herramientas para integración continua; de esta manera se agilizarán las salidas a producción de los nuevos desarrollos, para nuevos proyectos o sus modificaciones.

### 3.1.1. ¿Qué es Jenkins?

Jenkins es una herramienta *open source* que está diseñada para proyectos de integración continua, pero para conocer un poco más de este, se hará a hacer un poco de historia.

"Jenkins fue originalmente desarrollado con el nombre de Hudson por Kohsuke Kawaguchi, el cual fue inicialmente desarrollado a mediados del 2004 y su primera publicación se realizó en febrero del 2015".[4] En ese momento la empresa desarrolladora era Sun Microsystems, que inició este desarrollo para la gestión de proyectos de integración continua; el cual se comunica con muchas herramientas, tanto para control de versiones, herramientas para control de calidad, para compilar, entre otros.

[4] Wikipedia. *Historia de Jenkins*. https://en.wikipedia.org/wiki/Jenkins_(software). Consulta: 29 de agosto de 2018.

“En enero de 2010, se anuncia oficialmente que Oracle, adquiere la empresa Sun Microsystems”[5]; por lo tanto, adquiere todos sus proyectos, entre ellos el proyecto Hudson. “Pero no es sino hasta un año más tarde, en febrero del 2011 que el equipo de desarrollo de Oracle decide crear un proyecto llamado Jenkins”[6], pero deciden no hacerlo como un proyecto nuevo sino una rama del proyecto Hudson, sin desaparecer este, sino continuar con dos proyectos independientes con equipos de trabajo distintos, pero naciendo Jenkins de lo que Hudson llevaba avanzado hasta ese momento.

“Al 1 de febrero de 2019, el proyecto de Jenkins en GitHub cuenta con 661 miembros y más de 2000 repositorios”[7]; mientras que el proyecto de Hudson cuenta con 28 miembros y solamente 20 repositorios[8], esto da la pauta de cómo Jenkins ha crecido a través de los últimos años.

Figura 13. **Logo de Jenkins**

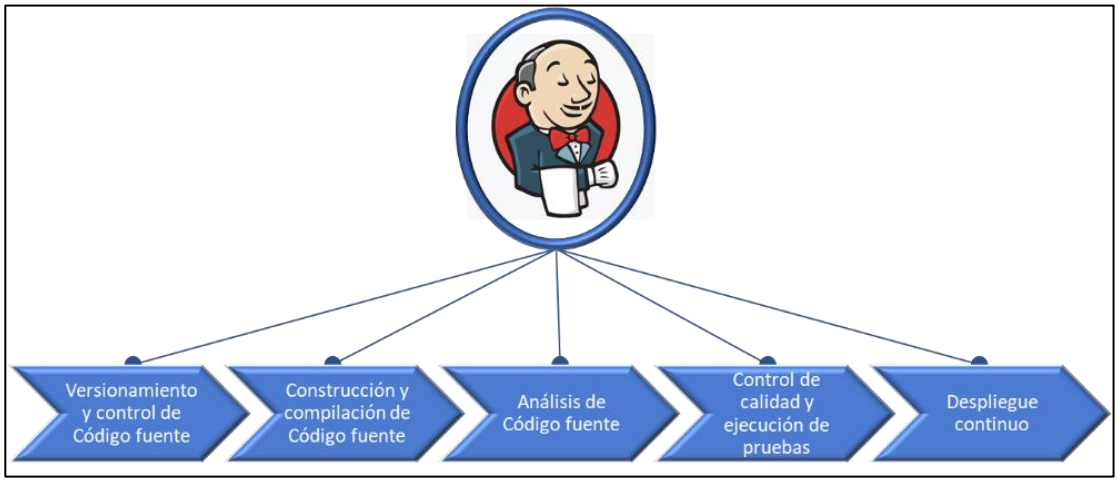
Fuente: Jenkins. *Repositorio Jenkins*. https://github.com/jenkinsci. Consulta: 1 de febrero de 2019.

[5] Oracle. *Compra de Sun Microsystems*. https://en.wikipedia.org/wiki/Sun_acquisition_by_Oracle. Consulta: 11 de octubre de 2018.
[6] Wikipedia. *Historia de Jenkins*. https://en.wikipedia.org/wiki/Jenkins_(software). Consulta: 29 de agosto de 2018
[7] Jenkins. *Repositorio Jenkins*. https://github.com/jenkinsci. Consulta: 1 de febrero de 2019.
[8] Hudson. *Repositorio Hudson*. https://github.com/hudson. Consulta: 1 de febrero de 2019.

### 3.1.2. Modelo de integración continua con Jenkins

Como se vio anteriormente, Jenkins es utilizado como una herramienta para la automatización de la integración continua, por lo que este cuenta con una serie de *plugins* o complementos, que no son más que conectores a otras aplicaciones para interactuar dinámicamente con ellas; es decir, Jenkins sería el encargado de las ejecuciones de cada aplicación en cada una de las fases de integración continua.

Figura 14. **Modelo básico de IC con Jenkins**




Fuente: elaboración propia.

Para Jenkins, el modelo es muy sencillo, ya que se basa en dichas conexiones y sus configuraciones de entrada para que cada aplicación ejecute las operaciones que cada una deba realizar en cada una de las fases del proceso de integración continua. Lo que lo hace complejo, es la configuración de las operaciones que cada herramienta de las que está conectada debe realizar independientemente.

Jenkins permite la ejecución de tareas manuales o programadas en cada fase; brindan como una respuesta, las estadísticas, reportes e informes del resultado de la operación, el cual puede servir como el disparador o trigger, para la siguiente operación dentro del flujo en el modelo previamente diseñado. Pero al final, la intención de Jenkins es tener todo el flujo automatizado para obtener un resultado final con la menor intervención humana posible en cada paso del flujo.

Jenkins es un software *open source*; su última versión estable puede ser descargada para distintas plataformas de manera gratuita de su link oficial, https://jenkins.io/download/. Existen también diversas fuentes, como lecturas y videos en donde indican paso a paso cómo instalar y configurar una versión de Jenkins de manera correcta, pero siempre se tiene la documentación oficial, la cual indica los requisitos tanto de infraestructura como de aplicaciones previamente instaladas, y los pasos que se necesitan para cada una de las plataformas (Linux, Windows, MacOS, inclusive en contenedores); https://jenkins.io/doc/book/installing/.

Con Jenkins se puede lograr el objetivo de utilizar la metodología DevOps, con proyectos de integración continua de manera transversal desde el equipo de desarrollo hasta el área de soporte; pasa siempre por el ambiente de control de calidad. Todo esto de manera automatizada, con la capacidad de tomar decisiones durante todas las fases del modelo.

Por definición, Jenkins ya cuenta con una serie de conectores (*plugins*) disponibles para su instalación; los que se pueden utilizar para comunicarse con las principales herramientas para cada una de las fases de la integración continua en el ámbito del desarrollo continuo y despliegue continuo.

A continuación, se describirá algunas herramientas básicas dentro de todo el proceso de integración continua que pueden ser integradas y orquestadas con Jenkins de manera automática.

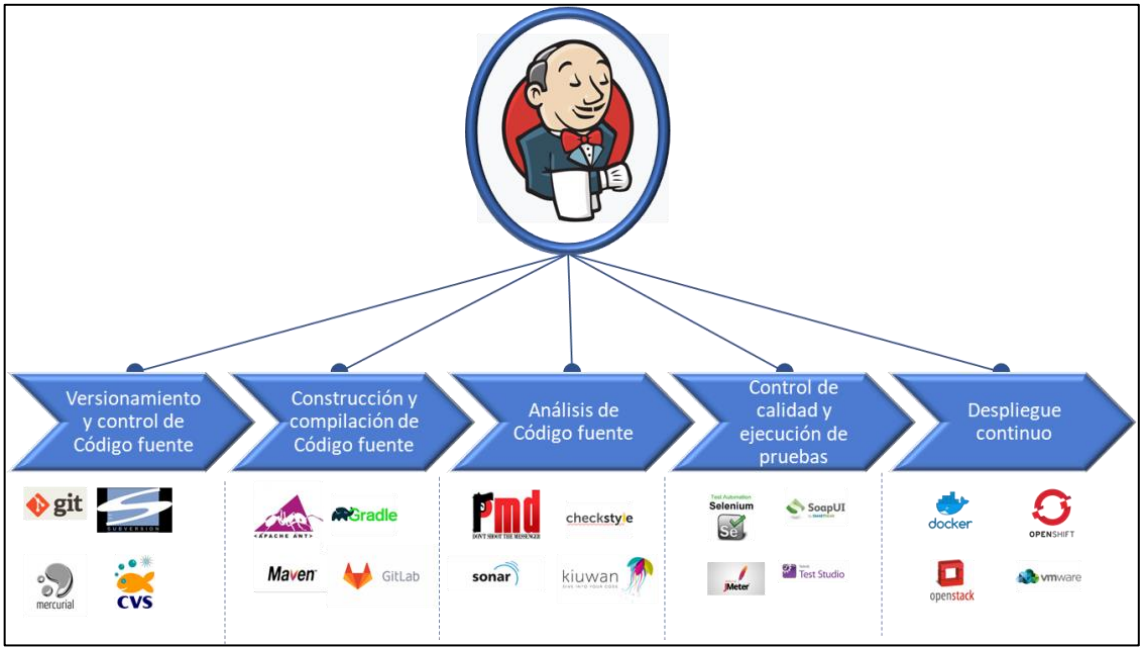
## 3.2. Herramientas básicas necesarias para una automatización en la integración continua con Jenkins

Como se vió anteriormente, el modelo de integración continua cuenta con una serie de fases que se debe ir trabajando para llevar un proyecto de software, desde un ambiente de desarrollo, hasta un ambiente de producción; dentro de los que se tiene, el control de código fuente, la generación y compilación de la aplicación, un análisis de código fuente, las gestión pruebas para el control de calidad y el despliegue de la aplicación en un ambiente de producción para la entrega del software, tanto al cliente final, como al equipo de soporte quien le dará seguimiento y de quien se recibirá retroalimentación de su uso.

Es importante recalcar, aunque ya se mencionó anteriormente, que la complejidad de la gestión de este modelo de integración continua no radica en el orquestador, sino que primero en la toma de decisión; de que herramientas se utilizará y en la configuración adecuada de cada una de las herramientas que jugarán un papel muy importante en cada una de las fases.

En este documento se mostrarán solamente algunas de estas herramientas ya que el mercado de *open source* es muy amplio; tanto así, que inclusive cada herramienta puede ser personalizada para cada organización o entidad y puede ser compartida también al público en general.

Figura 15. **Modelo de integración continua con Jenkins**




Fuente: elaboración propia.

### 3.2.1. Control de versiones para el control de código fuente

Cómo ya se ha venido indicando, existen diversas herramientas para el control y versionamiento de código fuente para los proyectos en los que los equipos de desarrollo trabajan. Este es el encargado de registrar los cambios realizados sobre el código fuente a lo largo del tiempo, de tal manera que pueda ser almacenado y recuperado en cualquier momento. Se hablarán tres formas en las que se pueden dividir los sistemas de control de versiones[9].

- Sistema de control de versiones locales

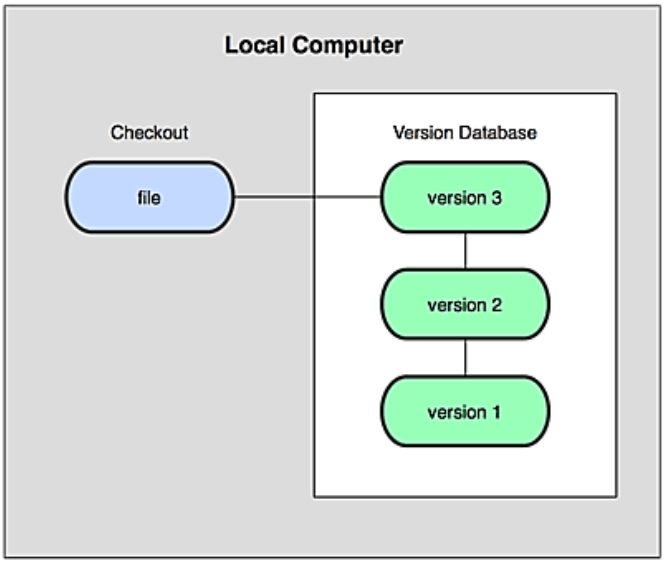
El sistema de control de versiones local es el más sencillo y básico, es utilizado para personas que desarrollan individualmente. Originalmente, este

[9] Control de versiones. *Acerca del control de versiones*. https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones. Consulta: 11 de octubre de 2018.

tipo de versionamiento se basaba en llevar el control de los cambios realizados en un archivo a través de colocarle la fecha y hora a cada archivo modificado. Este método tiene muchas deficiencias ya que, por la agilidad y premura, puede llegar a olvidarse el generar un archivo con una nueva fecha. Por ello fue necesario la adopción de una aplicación que se encargue de estos cambios la cual puede ser instalada localmente, sin necesidad de tener un servidor centralizado para llegar este control.

El esquema general se puede ver en la siguiente figura, la cual muestra como primera instancia un *checkout*, que no es más que descargar una versión que se requiera modificar del repositorio de código fuente, por definición se descarga la última versión estable.

Figura 16. **Sistema de control de versiones local**




Fuente: Control de versiones. *Acerca del control de versiones*. https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones. Consulta: 11 de octubre de 2018.

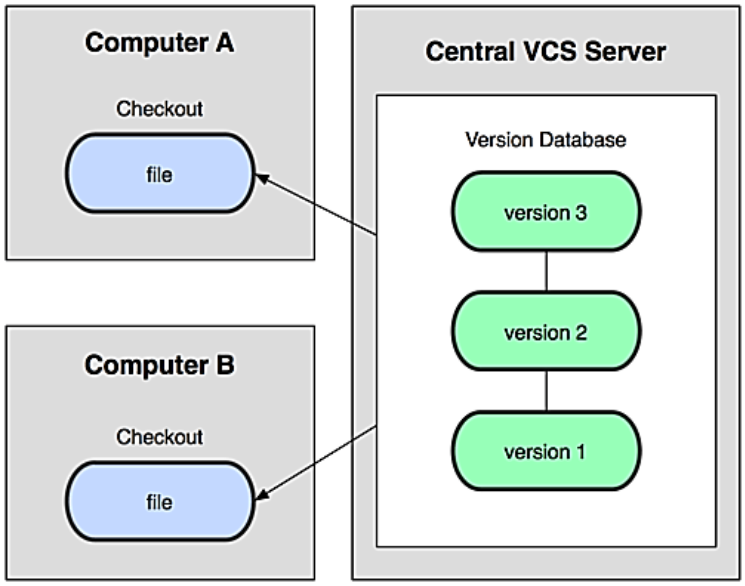
Este método no es utilizado en las organizaciones ya que está basado en un versionamiento local para un desarrollo en específico. La intención es dar a conocer cómo se gestiona un controlador de versiones en una organización o empresa con equipos de trabajo de más de una persona.

- Sistema de control de versiones centralizados

Este sistema de control de versiones para código fuente está basado en un servidor centralizado que almacenará todas las versiones que se han realizado por los distintos miembros del equipo de desarrollo. Este servidor es el que contiene todos los archivos versionados, y cada miembro del equipo, realiza un *check out* de la versión sobre la cual trabajará. Al finalizar el desarrollo, cada integrante que ha trabajado las modificaciones realizará un *commit* creando una nueva versión, en el repositorio centralizado.

Cuando el siguiente integrante del equipo de desarrollo necesite realizar su *commit*, el gestor de control de versiones le indicará que ya hay un cambio sobre la versión que el descargó originalmente, por lo que le solicitará que tome la decisión de dejar la versión del repositorio, la versión nueva o que se realice un control de versiones (merge) de los cambios realizados sobre la versión del repositorio. Los sistemas de control de versiones tienen la capacidad de realizar este merge automáticamente, pero si por alguna causa no lo pudiera hacer, genera algo que se le llama conflicto y debe ser resuelto por el último desarrollador que está realizando su versionamiento.

Figura 17. **Sistema de control de versiones centralizado**




Fuente: Control de versiones. *Acerca del control de versiones*. https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones. Consulta: 11 de octubre de 2018.

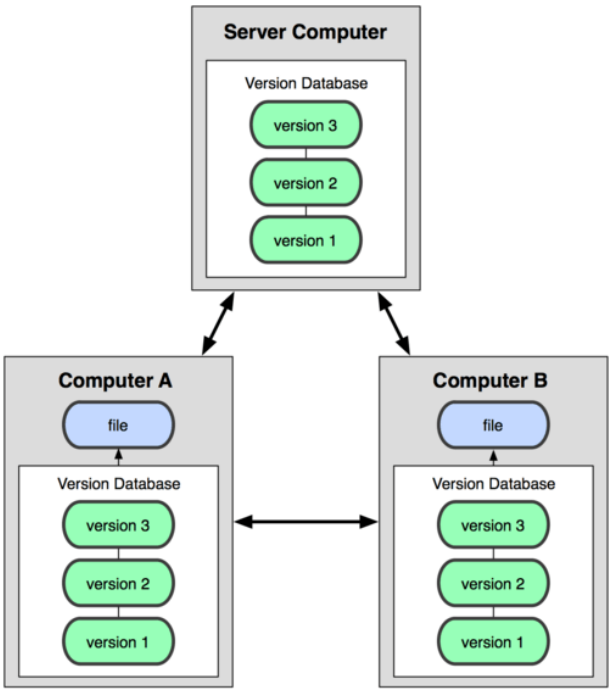
Este método es uno de los más utilizados, ya que posee muchas ventajas al momento de su gestión, ya que se tiene solamente un repositorio centralizado del código fuente. También, tiene el riesgo de que si se corrompe el servidor, se pierde todo el trabajo realizado por todos los desarrolladores. Para este tipo de casos se utilizan procedimientos para realizar *backups* continuos de este servidor y si falla el servidor, no se pierde todo el trabajo realizado.

- Sistema de control de versiones distribuidos

En este tipo de sistemas para el control de versiones, se tiene la capacidad de que cada desarrollador, al realizar el *check out*, descargue toda la base de datos de versiones y de esta manera si el servidor central falla, la

información permanece con uno o varios de los desarrolladores que están trabajando sobre el mismo proyecto.

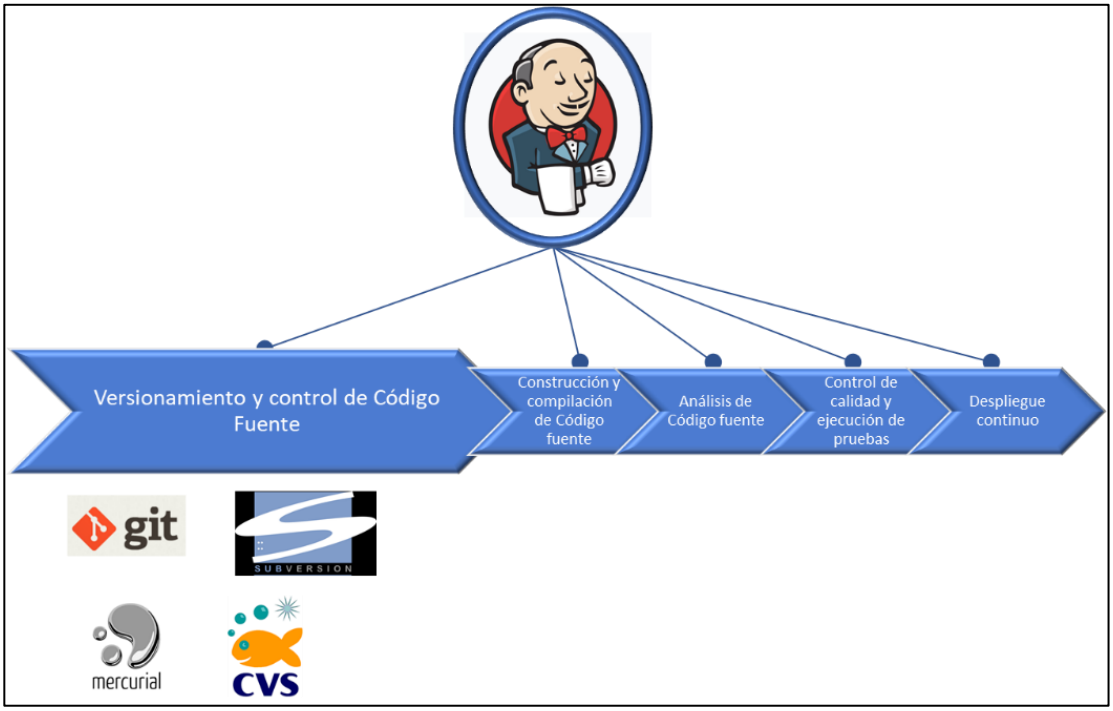
Figura 18. **Sistema de control de versiones distribuido**




Fuente: Control de versiones. *Acerca del control de versiones*. https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones. Consulta: 11 de octubre de 2018.

Ya que se describieron los tipos de sistemas con los que se pueden realizar un control de versiones de código fuente, se iniciará con la propuesta de una herramienta de control de versiones distribuida como lo es Git. A través de este documento se trabajará con esta versión para los ejemplos que se irán mencionando; pero de igual manera se puede utilizar cualquier herramienta de control de versiones que exista en el mercado no importando si es centralizado o distribuido, como Git, SVN, Mercurial, CVS, entre otros.

Figura 19. **Control de versiones, integración con Jenkins**




Fuente: elaboración propia.

### 3.2.2. Construcción y compilación de código fuente

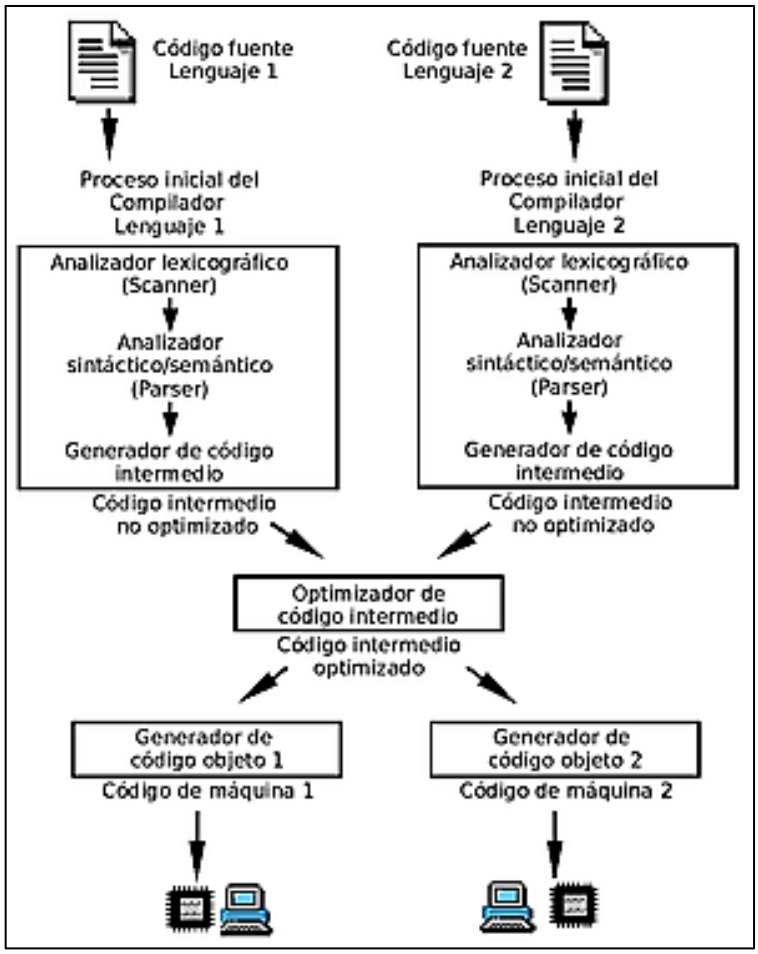
De igual manera que el control de versiones, la construcción y compilación del código fuente para la generación de código objeto, es necesario que haya herramientas que puedan automatizar este proceso; posteriormente, integrarlas a Jenkins, para que el flujo de la integración continua sea automático de igual manera.

Se sabe bien que existen varias fases que hacen que una compilación sea exitosa y las herramientas que se utilizan van a depender del lenguaje de programación. Muchos de los lenguajes de programación tienen integrado su

compilador, lo que hace más sencillo la generación del código objeto. En este caso se utilizarán compiladores para el lenguaje de programación en Java.

A continuación, se describen de forma breve lo que el compilador realiza al momento de generar el código objeto.

Figura 20. **Fases de un compilador**




Fuente: Wikipedia. *Compilación de esquema*. https://upload.wikimedia.org/wikipedia/commons/thumb/e/ef/CompilationScheme-Spanish.png/450px-CompilationScheme-Spanish.png. Consulta: 11 de octubre de 2018.

Básicamente, tiene que completar de manera exitosa todas las fases del compilador basándose en el código fuente y el lenguaje de programación. A continuación, se describirán de forma muy breve los pasos que el compilador debe completar solo como manera de referencia para el lector.

- Análisis lexicográfico

En esta fase, el compilador analiza secuencias de caracteres llamadas *tokens* para identificar que estén bien escritas y tengan un significado dentro del orden del lenguaje de programación utilizado. En esta etapa se revisa que las palabras clave del lenguaje de programación estén bien escritas y que no hayan sido utilizadas para elementos que no le correspondan dentro del código de programación. En este momento se optimiza la escritura, que quita espacios en blanco, líneas innecesarias en blanco, etc., para que al momento de generar el código objeto, el archivo sea más liviano. Este análisis realiza un análisis para la previa traducción de lo que el programador haya escrito. Este también es la entrada para el analizador sintáctico.

- Analizador sintáctico semántico

Este analizador se basa en expresiones regulares para identificar la jerarquía de los componentes léxicos ya analizados, para agruparlos y crear frases gramaticales entendibles por el compilador; de esta manera se simplifica el código que se utilizará para la salida.

El compilador ya cuenta con un conjunto de reglas que utilizará para esta generación de código, basada en la entrada del analizador léxico, ya que en conjunto se va armando la secuencia de frases que serán utilizadas por el compilador para la generación del código de salida.

Mientras que el analizador semántico, al igual que cualquier idioma, realizará un análisis para encontrar los errores de semántica y llevar un orden adecuado sobre el código generado por el desarrollador y generar el código intermedio previo a la generación del código objeto. Acá es donde observa el compilador el correcto uso de las variables y literales y que los tipos de variable estén correctamente definidos y utilizados.

- Generador de código intermedio

Este es un paso previo a la generación del código objeto, el cual lo baja a un lenguaje muy cercano a este, y es comúnmente llamado código de 3 direcciones, ya que este código generado utiliza, por cada lectura, tres apuntadores que se van ensamblando unos con otros y llevar una secuencia de lo que el desarrollador ha querido indicar que la máquina realice. Este paso tiene dos objetivos importantes; debe ser fácil su generación y debe ser fácil la creación de código objeto en base al código intermedio generado.

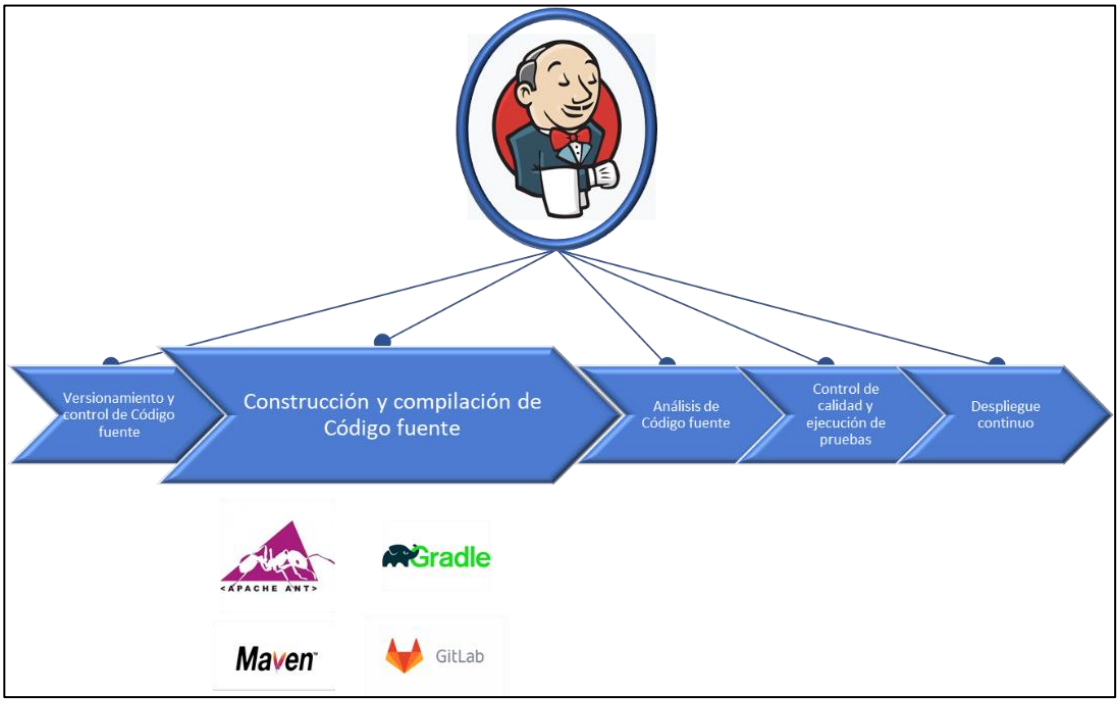
Este proceso lleva a cabo una última optimización, que reduce los pasos para las operaciones internas a nivel de procesamiento, para agilizar la generación de resultados en dichas operaciones.

- Generador de código objeto o máquina

El último paso del compilador es la generación del código objeto o código máquina, que no es más que la generación de un archivo ejecutable entendible por la computadora. En el caso de Java, se genera un archivo ejecutable que pueda ser leído e interpretado por la máquina virtual de Java, comúnmente con extensiones JAR o WAR o EAR.

Existen varias herramientas para la compilación de un código fuente en Java que se puede utilizar para automatizar el proceso e integrarlo a Jenkins como Ant, Maven, Gradle, GitLab, entre otros.

Figura 21. **Compilación de código fuente, integración con Jenkins**




Fuente: elaboración propia.

Se debe poner especial atención en la configuración de las herramientas para la compilación del producto, para que al ser llamadas por Jenkins automáticamente realice el proceso, y genere la salida para la siguiente fase para la integración continua, que en este caso es el control de calidad.

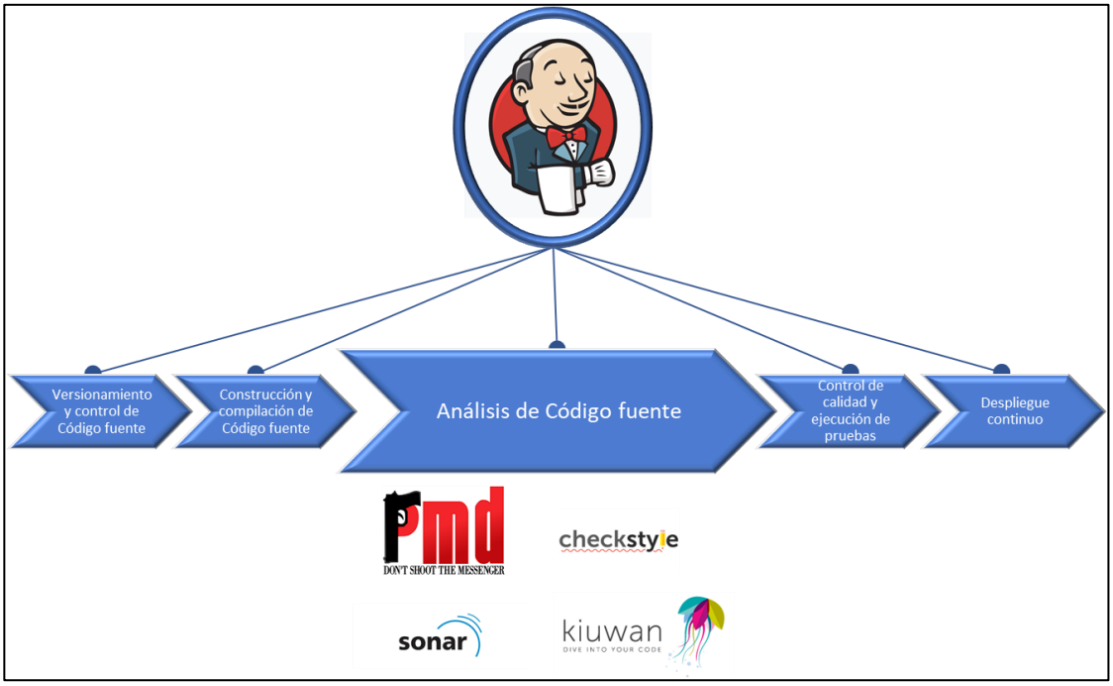
### 3.2.3. Análisis de código fuente

Existen buenas prácticas en el desarrollo de software, pero en la mayoría de los casos, los ingenieros de software se inclinan más por la generación de código acelerado que por la construcción de líneas de código que utiliza los estándares de desarrollo. Así como un editor de textos tiene un sistema de revisión de sintaxis, se han desarrollado herramientas que analizan el código fuente para validar el correcto uso de su escritura y más aún, la detección temprana de errores que podrían ocasionar bucles o recursividad infinita, código duplicado, código que está escrito, pero nunca se utilizará como variables que fueron declaradas, pero nunca fueron utilizadas, entre otros.

Previo a la generación de código objeto, es necesario hacer revisión sobre el código fuente, para validar estáticamente que este sea correcto y adecuado. Para ello utilizaremos estas herramientas que automatizarán este proceso de manera rápida y efectiva, y se gestionará a través de Jenkins.

Algunas de las herramientas que podemos utilizar para este análisis son las siguientes: PMD, Check Style, SONAR, Kiuwan, entre otros.

Figura 22. **Análisis de código fuente**




Fuente: elaboración propia.

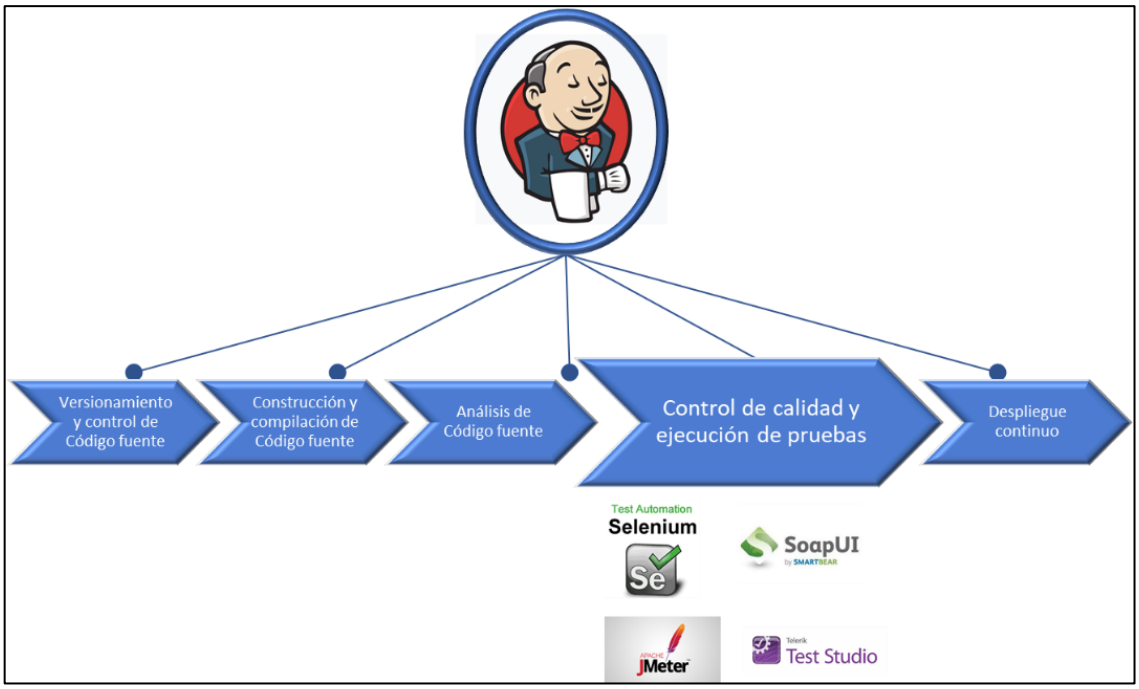
### 3.2.4. Control de calidad y ejecución de pruebas

En este punto es muy importante mencionar que el desarrollador es el responsable de realizar las primeras pruebas, la ejecución de pruebas unitarias, para que el equipo de control de calidad y sus pruebas automatizadas generen informes específicos sobre fallas no observadas en las pruebas del desarrollador.

Las pruebas de control de calidad son enviadas cuando ya se cuente con un programa ejecutable ya compilado correctamente, por lo que este sería el siguiente paso posterior a la compilación de código fuente y la generación de código objeto o máquina.

En el capítulo cuarto de este documento se estará profundizando esta fase del control de calidad y ejecución de pruebas, siempre siguiendo los mismos lineamientos de la integración continua, que es automatizar cada uno de sus procesos y fases. Pero se adelantarán algunas de las herramientas que podemos utilizar para realizar la ejecución de las pruebas de control de calidad: Selenium, SoapUI, JMeter, Test Studio, entre otros.

Figura 23. **Control de calidad, integración con Jenkins**




Fuente: elaboración propia.

### 3.2.5. Control de despliegues (despliegue continuo)

El objetivo del proceso de integración continua es agilizar cada una de las etapas del desarrollo de software, pero también es la generación de un producto final y colocarlo en un ambiente de producción lo más pronto posible

con la menor cantidad de fallas. Para ello se tiene el control de despliegues o despliegue continuo, que es el proceso que ayudará a llevar un producto con buena calidad, analizado en el ambiente de pruebas, hacia el ambiente de producción con la menor afectación posible.

Existe una gran cantidad de formas de cómo realizar un despliegue a producción de un producto, dependiendo de las necesidades de las empresas o de cómo esté diseñada la arquitectura de sus equipos físicos y servicios. Muchas de las ocasiones cuando una empresa está cambiando de desarrollar de una forma tradicional, cambiando en las fases de desarrollo manualmente a utilizar la metodología de integración continua, la arquitectura suele cambiar para adaptarse a la agilidad que se quiere ganar.

Se hablará un poco sobre algunos de los tipos de despliegue que las empresas, organizaciones o grupos de desarrolladores utilizan comúnmente, probablemente algunos de ellos se les harán conocidos a los lectores de este documento.

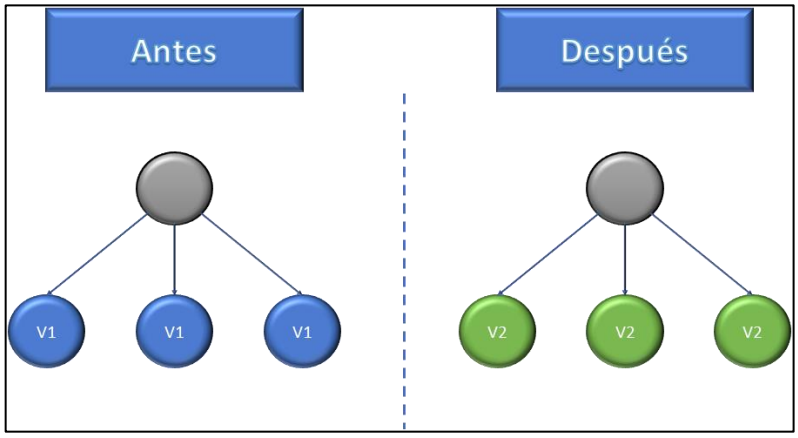
- Despliegue inmediato

El despliegue inmediato es uno de los modelos de despliegue más rústicos y con mayor riesgo para los ambientes de producción, ya que el cambio es de alto impacto ya que la forma del cambio es apagar la versión existente y colocar la nueva versión y activar nuevamente. Todo esto se realiza en cualquier momento. Este modo de despliegue es muy utilizado para los ambientes de desarrollo o escenarios de producción de bajo impacto.

Para aplicaciones que cuentan con una gran cantidad de módulos y con una base de datos centralizada, es de alto riesgo utilizar este método. Las

empresas que utilizan este método comúnmente ejecutan los despliegues en horarios no laborales o de menor impacto al usuario final.

Figura 24. **Despliegue Inmediato**




Fuente: elaboración propia.

- Despliegue escalonado

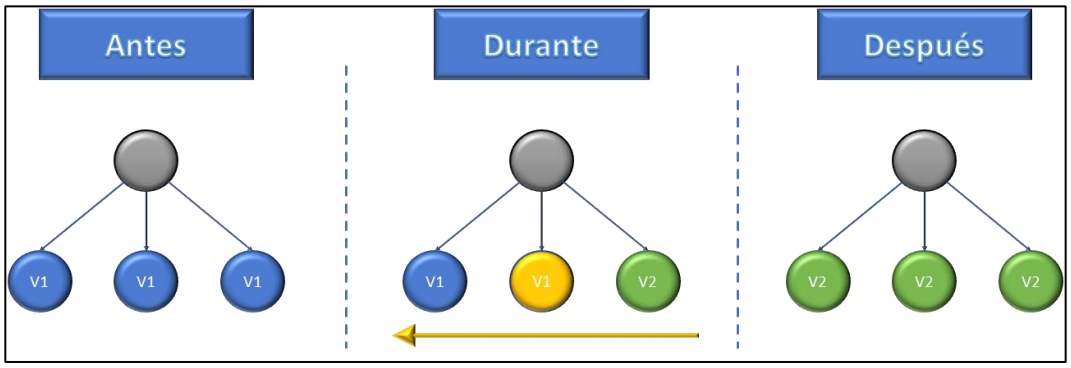
En este modelo, el despliegue se realiza de manera escalonada, es decir de servidor en servidor, o de servicio en servicio, por lo que el control de fallas en producción es menor. Es muy importante saber que este modelo puede ser utilizado solamente cuando los servidores o servicios funcionan independientemente, es decir, que la carga de trabajo sobre la aplicación está distribuida y que, si la base de datos también requiere cambios, no esté centralizada o que el cambio sobre la base de datos no afecte la versión anterior.

Una de las características más importantes de este modelo, es que se puede tener monitoreado el cambio durante el proceso del despliegue, con la

capacidad de decidir dejar el cambio y continuar con el resto o deshacer el cambio inmediatamente.

Este método puede ser muy útil para sistemas que cuenten con múltiples módulos dentro de la aplicación; de esta manera, no se afectarían todos los servicios sino solamente los que se están cambiando o sobre lo que se realizó el desarrollo. Este tipo de cambios tiene un bajo impacto sobre el usuario final.

Figura 25. **Despliegue escalonado**




Fuente: elaboración propia.

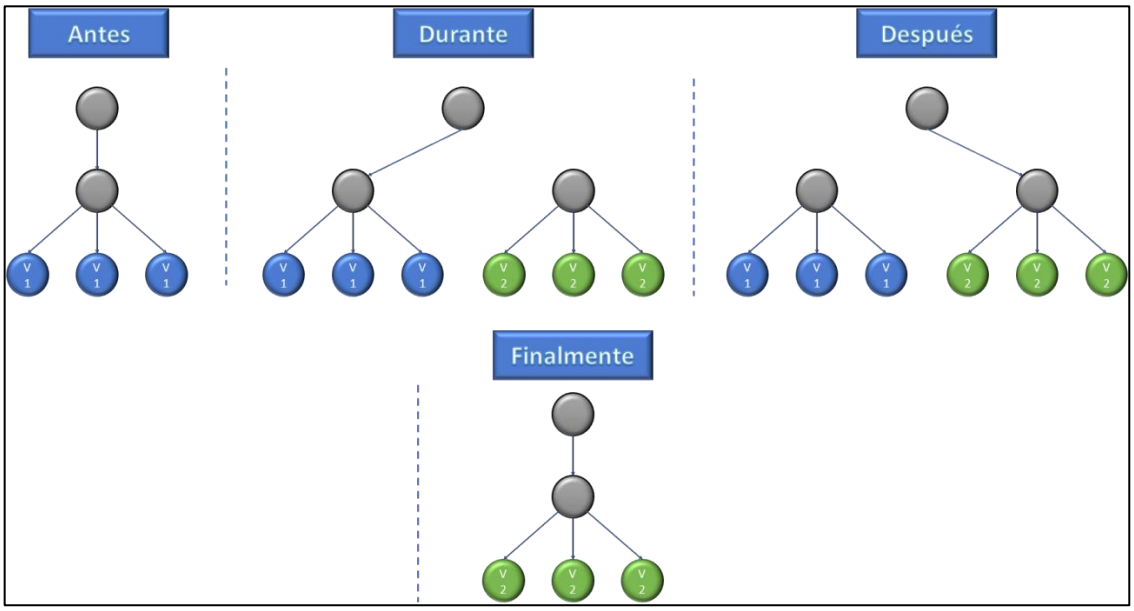
- Despliegue *blue/green*

Este método para la realización de despliegues se basa en replicar un ambiente completamente igual al de producción con la nueva versión. Este es un método muy rápido y seguro, pero requiere que la arquitectura del sistema esté diseñada con un alto nivel de escalabilidad, introduciendo a este método conceptos como el uso de contenedores, virtualización, entre otros.

Es muy importante contar con un servidor que se encargue de balancear la carga entre distintos equipos para esta arquitectura, ya que esto es lo que proveerá la movilidad entre los distintos ambientes con las distintas versiones.

Al momento que se realice el cambio en el balanceo de la carga, las aplicaciones automáticamente cambiarán hacia la nueva versión; de esta manera se puede monitorear y controlar el nuevo ambiente, tomando la decisión de dejar la aplicación allí o si se observan fallas; inmediatamente realizar el cambio de regreso al ambiente con la versión anterior, y restableciendo el sistema inmediatamente. De esta manera, la afectación al cliente final es mínima. Este modelo es muy utilizado para ciertas empresas que utilizan integración continua para sus aplicaciones.

Figura 26. **Despliegue blue/green**



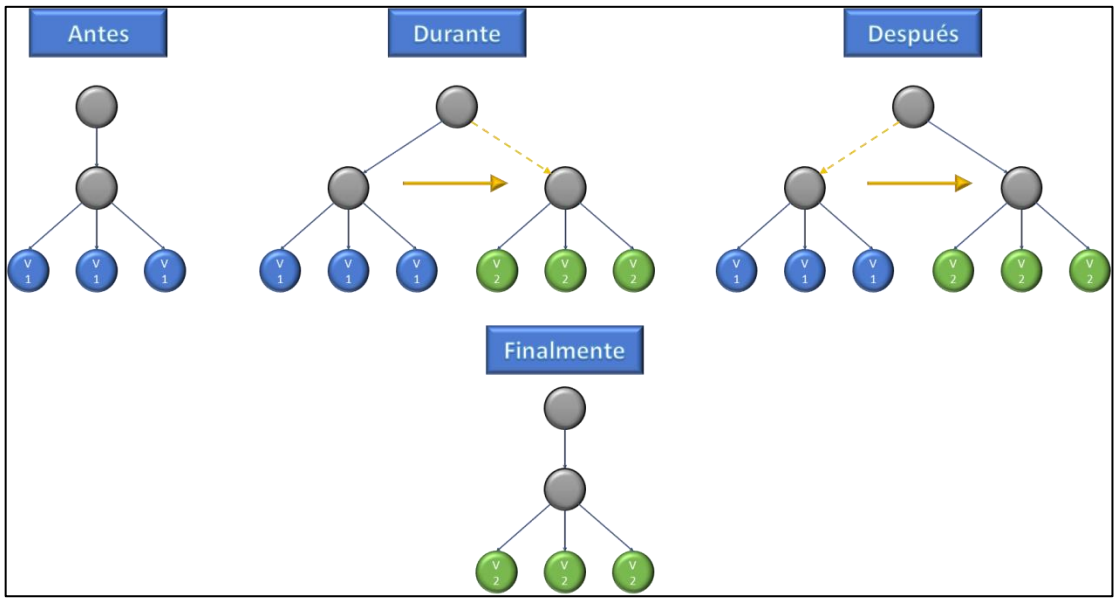

Fuente: elaboración propia.

- Despliegue Canary

En este tipo de despliegues también se necesita la replicación completa del ambiente de producción, pero a diferencia de que el método de blue/green, la transaccionalidad del uso de la nueva versión de la aplicación se va habilitando controladamente poco a poco, trasladando un porcentaje pequeño de usuarios a la nueva versión. Por ejemplo, se puede indicar que pase primero el 5 %, luego el 15 %, luego el 40 %, luego el 80 %, hasta llegar al 100 % de los usuarios utilizando la nueva versión; pero para definir adecuadamente el porcentaje, es necesario hacer un análisis de usabilidad de la aplicación la que puede variar dependiendo el tipo de empresa.

Con este método también es necesario contar con un servidor de balanceo de tráfico, el que será el encargado de ir habilitando a cierta cantidad de usuarios la nueva versión de la aplicación. Como se ve, en este método y en el anterior, la arquitectura del sistema debe ser preparado para realizar este tipo de despliegues; nuevamente, se hace mención del uso de contenedores y virtualización. Todo esto se puede realizar utilizando infraestructura física, pero los costos son más elevados, tanto por el incremento de recursos financieros, como el tiempo para la preparación de los equipos; aunque si se habla de automatizar todo el flujo, también, existen herramientas que sirven para automatizar las configuraciones desde cero de un servidor sin intervención humana, para dar un ejemplo podría ser el uso de Ansible.

Figura 27. **Despliegue Canary**



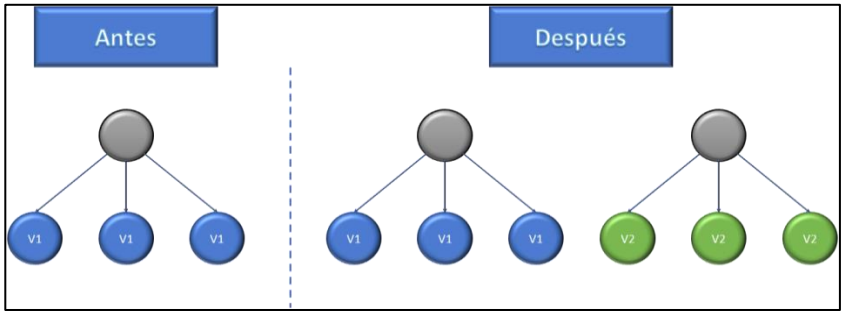

Fuente: elaboración propia.

- Despliegue por versiones

Otro método para el despliegue de software a un ambiente de producción es el despliegue por versiones. Este tipo de despliegue es muy utilizado por aplicaciones para dispositivos móviles, ya que se mantienen todas las versiones publicadas hasta que todos los usuarios hayan migrado a la nueva versión. Una versión desaparece cuando esta ya no cuenta con clientes activos.

El único inconveniente es que, con este tipo de despliegue, no se realiza un cambio de versión forzado para el usuario, sino que el usuario cambia cuando él así lo requiera, y pueden ir quedando muchas versiones publicadas, y las más antiguas pueden ser muy ineficientes por lo tanto poder presentar una mala experiencia en el cliente al respecto de la aplicación utilizada.

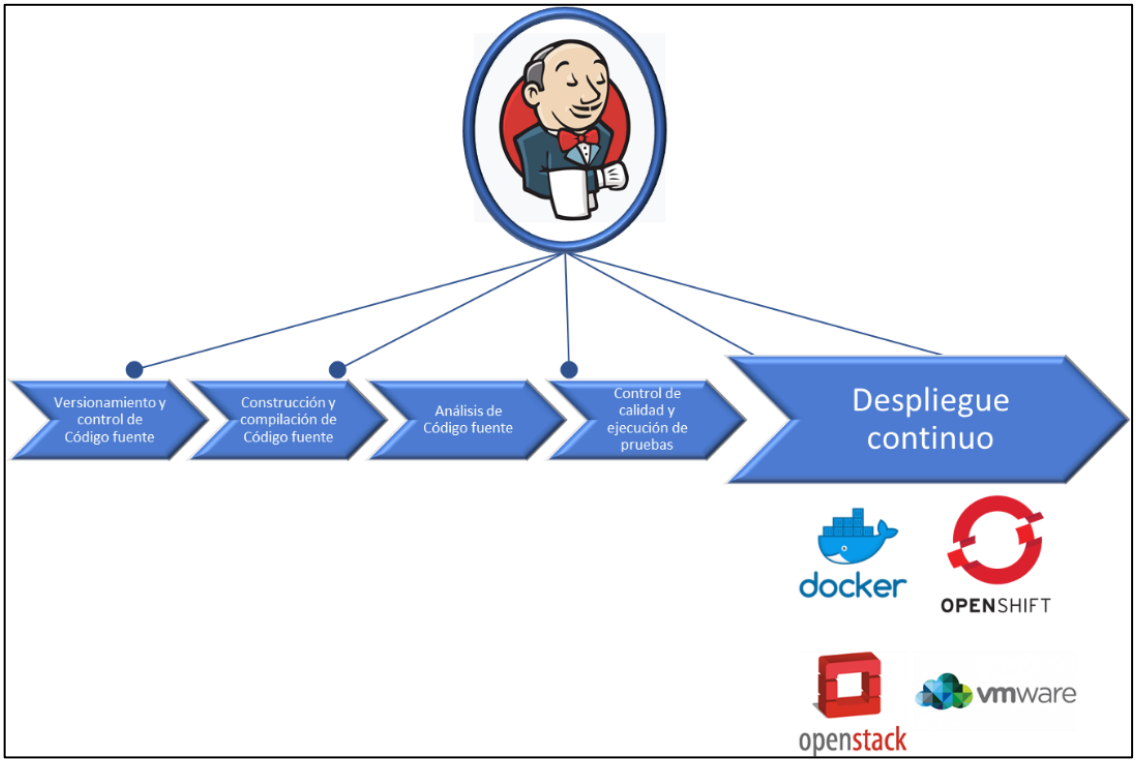
Figura 28. **Despliegue por versiones**




Fuente: elaboración propia.

Como ya se ha visto, cada una de las fases que conforman la integración continua y el despliegue continuo, requieren de ciertas herramientas que ayudarán a automatizar, con la ayuda de Jenkins, todo el proceso desde el desarrollo hasta la puesta en producción pasando por el ambiente de control de calidad.

Figura 29. **Despliegue continuo, integración Jenkins**




Fuente: elaboración propia.

# 4. LA IMPORTANCIA DE UN AMBIENTE DE CONTROL DE CALIDAD (QA) EN EL PROCESO DE DESARROLLO DE SOFTWARE

## 4.1. Gestión del control de calidad en desarrollo de software

A continuación, se describe la gestión del control de calidad en desarrollo de software.

### 4.1.1. Control de calidad en el desarrollo de software

El control de calidad del software es una etapa muy importante, si interesa entregar un producto con la menor cantidad de fallas posibles en el ambiente de producción. Esta etapa en el proceso de entrega de un producto de software varía dependiendo del tipo de software que se esté desarrollando, por lo que el equipo de control de la calidad debe iniciar con la preparación de las herramientas para las pruebas desde la concepción del proyecto y no esperar hasta que el equipo de desarrollo entregue la aplicación para pruebas.

Antes de iniciar con el desarrollo, en la etapa de análisis y diseño de un producto, se debe incluir el alcance de las pruebas a realizar en esta fase, ya que esto implica una inversión monetaria adicional. Entre más exactas sean las pruebas previo a poner el desarrollo terminado en el ambiente de producción, mayor será la inversión sobre el tipo, la cantidad y la calidad de las pruebas que se tengan que realizar, ya que, aunque existan una gran cantidad de herramientas que automatizan el proceso de integración continua, incluyendo la

etapa del control de calidad, siempre es necesario que un equipo de personas esté preparando dichos ambientes de pruebas.

Existen una serie de atributos que pueden verificarse para tener un muy completo análisis en las pruebas de calidad. Según el documento de estudio de arquitectura de software, se describen estos atributos de calidad basados en en dos categorías, observables y no observables en tiempo de ejecución, los cuales describiremos a continuación:

- Observables

Son aquellos atributos que se obtienen del comportamiento del sistema, los cuales se describen a continuación:

  - Disponibilidad

Es la capacidad que tiene la aplicación para poder usarse en el momento que el usuario así lo requiera. Este atributo debe ser definido por el encargado del desarrollo en conjunto con el cliente solicitante del software ya que, en la mayoría de las ocasiones, entre más alta es la disponibilidad mayor inversión sobre el sistema y su arquitectura hay que realizar.

  - Confidencialidad

Es la clasificación del acceso hacia la aplicación dependiendo del nivel de autorización que el usuario tenga sobre los mismos. Para ello se crean roles y niveles de servicio a nivel de la aplicación. Siempre debe de existir una persona encargada de brindar los niveles de autorización o privilegios sobre la aplicación ya que, en muchas ocasiones, las aplicaciones para grandes empresas,

manejan información muy sensible, como financiera y personal que no todas las personas deben conocer.

- Funcionalidad

Es la habilidad que tiene la aplicación para poder realizar sus funciones de acuerdo con la solicitud realizada por el cliente. Es muy importante establecer los límites y alcances de la aplicación y los criterios del negocio desde el principio ya que, en la mayoría de los casos, el cliente asume que se realizarán ciertas acciones dentro de la aplicación. Es por ello por lo que se debe definir qué y qué no realizará la aplicación. Más adelante se revisarán los tipos de pruebas que se deben realizar para mantener un control de calidad sano.

- Desempeño

Este atributo es muy importante y casi nunca ser realizan pruebas específicas para asegurar el desempeño de la aplicación. En su mayoría, las aplicaciones que pasan todas las pruebas fallan en el ambiente de producción debido a la falta de las pruebas de desempeño o el incorrecto análisis de estas pruebas.

El desempeño básicamente ser refiere a la capacidad de la aplicación para poder responder las peticiones y/o los requerimientos al usuario en un tiempo prudencial o definido por el cliente solicitante. El desempeño también incluye el análisis de las capacidades de los equipos de infraestructura, que soporten la transaccionalidad que la aplicación recibirá, tanto como sus configuraciones óptimas.

- Confiabilidad

Es la habilidad de un sistema en mantenerse operativo a través del tiempo, esto implica que no solo la aplicación esté disponible, si no que pueda usarse adecuadamente. El nivel de confiabilidad será dado a través del tiempo, este atributo puede ser medido a través de indicadores automáticos de pruebas continúas automatizadas en tiempo de ejecución.

- Seguridad externa

Este nivel de seguridad se refiere a cualquier intrusión o afectación de los servicios de la aplicación por consecuencias ambientales, o intrusiones físicas no autorizadas a alguno de los elementos que conforman la infraestructura de la aplicación. Para mitigar la afectación sobre este atributo, se instalan ambientes de infraestructura de contingencia en lugares distintos físicamente para evitar que un servicio o sistema esté fuera por mucho tiempo al ser afectado o invadido.

Dentro de los elementos que pueden afectar la seguridad de manera externa podemos mencionar, terremotos, incendios, inundaciones, dependiendo de la ubicación de los mismos, tanto como cortes de energía eléctrica pública, cortes de los cables de transmisión de datos que están expuestos a la intemperie, entre otros.

- Seguridad interna

Es la capacidad del sistema de defenderse de accesos no autorizados al sistema, o ataques de negación de servicio al mismo, mientras usuarios autorizados realizan sus operaciones dentro de la aplicación. Este tipo de

ataques es muy común, pueden ser invasores tratando de implantar virus en los sistemas, o lo que ha sido muy común en grandes corporaciones, ataques de *hackers* para extraer información sensible y privilegiada dentro de una organización.

También, existen empresas dedicadas a la creación de herramientas tanto físicas y de software que impidan que estos ataques lleguen hasta la aplicación interna o la infraestructura sobre la cual está instalada, como antivirus, antispyware, malware, firewalls, DPI, IPS, HIDS, NIDS, entre otros.

- No observables

Son aquellos atributos que se obtienen desde el momento del desarrollo del software. Es por ello que es de gran importancia que el ingeniero de software encargado del desarrollo del mismo conozca estos atributos para la generación de producto final de calidad.

  - Configurabilidad

Posibilidad que se brinda a través de privilegios especiales a uno o un grupo de usuarios con el conocimiento adecuado del sistema a realizar ciertos cambios que modifiquen o mejoren el comportamiento de la aplicación en base al análisis de uso del sistema por parte de los usuarios.

  - Integrabilidad

Es la capacidad que tienen los distintos componentes de la aplicación sobre los que los desarrolladores del software han trabajado

independientemente de integrarse al resto de la aplicación sin causar un efecto negativo sobre la misma.

- Integridad

Se conoce como integridad del software cuando este no puede ser modificado inapropiadamente, para no causar efectos indeseados sobre el comportamiento de la aplicación. Esto brinda un nivel de confianza sobre la información que en ella se encuentre. Tales como no permitir la modificación de valores financieros antiguos sobre los que ya se cerraron las operaciones, no permitir modificar información personal sensible dentro de la organización.

- Interoperabilidad

Esta es una parte complementaria de la integrabilidad, que permite a la aplicación o sus componentes poder trabajar con otro sistema, por ejemplo, un módulo de caja pueda trabajar correctamente con un módulo contable.

- Modificabilidad

Es la habilidad que tiene un software de realizar cambios sobre la aplicación más adelante, tanto para corrección de errores como para mejoras e innovaciones sobre la misma.

- Mantenibilidad

Este atributo se refiere a la capacidad del software para que el equipo de soporte pueda realizar modificaciones en las configuraciones de manera inmediata para realizar o mejorar las operaciones sobre las mismas, o realizar

reparaciones inmediatas por el equipo de desarrollo de software sobre un componente con falla.

- Portabilidad

La portabilidad permite a la aplicación ser ejecutada desde distintos equipos de hardware o software. Por ejemplo, la aplicación puede ser ejecutada desde Windows, Linux, MacOS o Android, entre otros, sin necesidad de realizar muchas modificaciones sobre la misma. Java es un lenguaje de programación que permite ser ejecutado en múltiples plataformas.

- Reusabilidad

Es la capacidad de la aplicación para utilizar uno o más componentes para otras partes de la aplicación, sin necesidad de volver a generar nueva codificación.

- Escalabilidad

Lo que cualquier empresa u organización espera, es que esta crezca con el tiempo, por lo que este atributo indicará la capacidad de que la arquitectura en donde se ejecute esta aplicación pueda ser adaptada a las nuevas necesidades del mercado sin tener que cambiar el desarrollo realizado.

- Capacidad de prueba

Este atributo indicará la capacidad que tiene la aplicación para ser sometido a un adecuado y completo control de calidad, sin necesidad de realizar tantas configuraciones o modificaciones a los ambientes de pruebas.

### 4.1.2. Tipos de pruebas en el ambiente de control de calidad

En el capítulo segundo se trataron algunas herramientas para el control de calidad en un modelo de integración continua; por lo tanto, a continuación se describirán cuáles son algunas de las pruebas que podrían ser necesarias para la realización adecuada del control de calidad de un producto de software.

- Pruebas funcionales de software

Son aquellas pruebas que están diseñadas para validar la aplicación vista desde el comportamiento que este debe realizar según lo definido por el cliente requirente. Si se está utilizando una metodología de software, como se mencionó al inicio del documento como ejemplo Scrum, el cliente, a través del método llamado User Story, definirá los *test cases*, que son las pruebas válidas de como la aplicación debe funcionar; indicando los parámetros de entrada y una salida esperada de los resultados de las operaciones. Es por ello que el ingeniero de control de calidad puede iniciar la configuración de las pruebas desde antes que el software esté desarrollado.

Dentro del grupo de las pruebas funcionales hay que incluir aquellas pruebas que el ingeniero de desarrollo proponga como genéricas con base en lo desarrollado por él, que el cliente no haya tomado en cuenta dentro de los *test cases*.

Un buen control de calidad de pruebas funcionales incluiría las pruebas sobre componentes que interactúan con la parte de la aplicación que será instalada, y no solamente analizar y probar la parte de la aplicación que acaba de finalizar su desarrollo.

- Pruebas no funcionales de software

Estas pruebas comúnmente son omitidas, ya que requiere tiempo adicional dentro del proceso de control de calidad, ya que se deben tomar en cuenta valores erróneos o equivocados del comportamiento normal de la aplicación.

Estas pruebas son muy importantes ya que en muchas ocasiones el usuario final comete errores ingresando por ejemplo caracteres no numéricos en campos que permiten valores numéricos, valores inadecuados en campos con una estructura determinada, como número de teléfono (que en Guatemala es de 8 dígitos), el DPI, el NIT, entre otros.

Tanto para las pruebas funcionales como no funcionales, si tienen las pruebas de caja blanca y caja negra, que no son más que pruebas realizadas, observando el sistema con pruebas desde adentro del módulo que acaba de ser desarrollado como desde afuera del módulo que incluye pruebas punto a punto para analizar el comportamiento de este.

- Pruebas de seguridad

Las pruebas de seguridad indicarán al ingeniero de control de calidad, la facilidad con que un usuario puede o no realizar operaciones permitidas según su rol específico dentro de la aplicación. Para ello es necesario revisar que las configuraciones de los roles sean las correctas, y las pruebas con los distintos roles permitan el acceso adecuado solamente a los segmentos de la aplicación para los cuales fueron diseñados.

Con las pruebas de seguridad también se asegura de que el usuario no pueda cambiar de un módulo a otro por ningún enlace o conexión, aunque internamente la aplicación almacene información que le será útil a otro módulo para realizar sus operaciones o desempeñar su trabajo adecuadamente.

- Pruebas estructurales de software

Las pruebas estructurales garantizarán que el código generado sea el correcto y haya sido bien utilizado, esto para que no ocasione problemas de rendimiento o funcionalidades inadecuadas de la aplicación. Con las pruebas estructurales se identifica si el módulo escrito responde las solicitudes adecuadamente según lo esperado, para este análisis se pueden utilizar las pruebas de caja blanca.

- Pruebas de regresión

Es la capacidad de ejecutar nuevamente un conjunto de pruebas posterior a haber cambiado el código fuente debido a una solicitud de cambio al equipo de desarrollo por parte del equipo de control de calidad por causa de una falla identificada en una de las corridas de las pruebas.

- Pruebas de carga y rendimiento

Esta es una prueba muy importante para el análisis del control de calidad, aunque en la mayoría de las ocasiones no se ejecutan debido a la insuficiencia de recursos o la capacidad para realizarla.

Las pruebas de carga y rendimiento se basan en la capacidad de la aplicación para trabajar y responder adecuadamente en momentos de alto

tráfico o transaccionalidad por parte de los usuarios; por lo que comúnmente se utilizan herramientas que simulen dicho tráfico mientras se ejecutan pruebas normales.

Estas pruebas también son utilizadas para medir o delimitar la capacidad máxima de la infraestructura instalada y decidir si esta capacidad es suficiente o es necesario incrementarla para atender las necesidades de los usuarios. Recuérdese que al final, lo que se requiere es tener una buena respuesta y atención en todos los puntos del negocio.

Estas pruebas también pueden abarcar las distintas áreas de la aplicación, haciendo pruebas de carga y rendimiento sobre el módulo desarrollado y sobre toda la aplicación, para validar que el nuevo módulo a instalar no afecte el rendimiento que ya se tiene sobre la aplicación ya instalada.

## 4.2. Automatización de pruebas en el ambiente de control de calidad

Ya que se conocen los atributos y los tipos de pruebas para el control de calidad con el que se trabajará, y que se ve que pueden llegar a ser demasiadas y en muchos de los casos muy complejas, se tiene que ver cómo reducir el tiempo para cubrir la mayor parte o todas las pruebas disponibles a través del tiempo, y esto solo se logrará con su correcta automatización.

Anteriormente, ya se hablaba de ciertas herramientas con las que automatizar las pruebas para el control de calidad y que con muchas de esas se pueden integrarlas a Jenkins para el uso de integración continua y de esta manera acercarse más a la metodología de DevOps, generando los informes necesarios para dar seguridad al área de operaciones que el software generado sea el adecuado para el cliente final.

Dentro de las herramientas que se puede utilizar para un ciclo adecuado del control de calidad se tiene, por ejemplo:

- Sonar
- PMD
- Check Style
- JUnit
- qVision
- Selenium
- JMeter
- SoapUI
- Test Studio

Se pueden utilizar una o más herramientas según sea la necesidad de las pruebas.

# 5. DISEÑO DEL DESPLIEGUE AUTOMATIZADO DE SOLUCIONES DE SOFTWARE PARA SU PUESTA EN PRODUCCIÓN, PASANDO POR UN AMBIENTE DE CONTROL DE CALIDAD

## 5.1. Diseño del despliegue de desarrollo hacia el ambiente de control de calidad

Las grandes empresas gestionan su información y sus operaciones de distintas maneras, algunas empresas lo realizan adquiriendo herramientas y aplicaciones de software que son conocidas en el mercado, y otras empresas crean sus propias áreas de desarrollo para crear sus propias herramientas y aplicaciones.

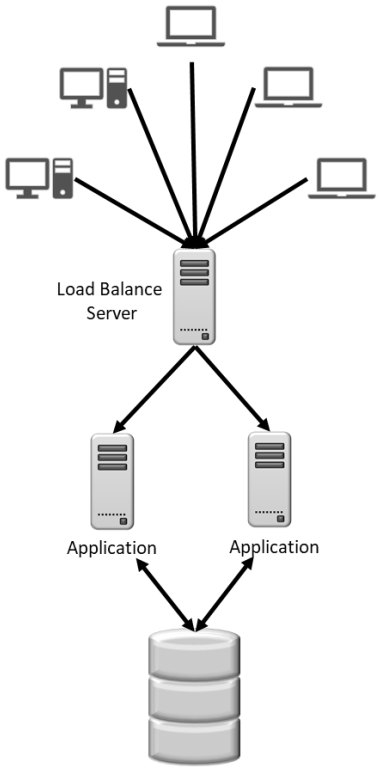
A través de los años estas aplicaciones se vuelven cada vez más complejas para realizar desarrollos nuevos o cambios sobre los módulos ya existentes, y también más críticas, hablando comercialmente, ya que la empresa depende del correcto funcionamiento de estas aplicaciones para llevar a cabo sus operaciones.

Un tema que no se trató en este documento, pero es muy importante mencionarlo, es que al principio del proyecto es necesario que el nuevo producto o servicio, o si es un cambio sobre la aplicación ya existente, tenga un análisis de impacto tanto para la infraestructura que incluya el impacto de colocarlo en el ambiente de producción, con esto se podrá definir qué tipo de despliegue se realizará, y los recursos necesarios para su realización.

El diseño será sobre una empresa ficticia, del sector de las telecomunicaciones, y se realizar sobre las premisas de que el módulo ya se encuentra operando y que se cuenta con la infraestructura y recursos necesarios para su realización, en este caso será sobre el módulo de contabilidad de la empresa, y su necesidad de cambios a través del tiempo.

La arquitectura actual del módulo de contabilidad cuenta con dos servidores de aplicaciones con un contenedor de *servlets* que reciben peticiones de los usuarios a través de un balanceador y se tiene una sola base de datos centralizada, estructurado de la siguiente manera:

Figura 30. **Arquitectura del módulo de contabilidad**




Fuente: elaboración propia.

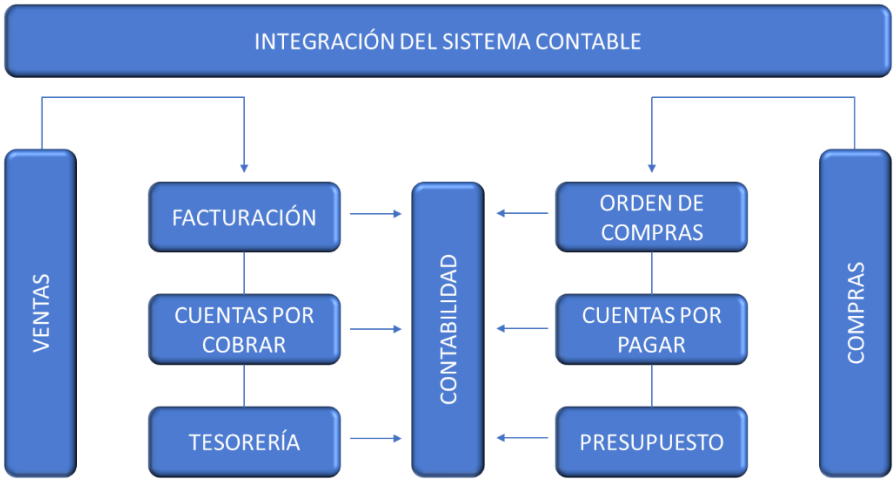
Cada servidor cuenta con las siguientes características:

- Hardware
    - Procesador: 8 núcleos de procesador
    - Memoria RAM: 32 GB RAM
    - HDD:  1 Unidad 100 GB para SO y aplicación
    - HDD:  1 Unidad 200 GB para almacenamiento de Logs, reportes y registros

- Software
    - Sistema operativo Linux
    - Apache Web Server
    - Apache Tomcat
    - Java JDK
    - MySQL Database

El sistema contable que se estará trabajando consta, de las áreas de ventas y compras, gestionados a través del módulo de contabilidad por los módulos de ventas, facturación, cuentas por cobrar, tesorería, compras, orden de compras, cuentas por pagar y presupuesto; todos estos interactúan con una base de datos centralizada.

Figura 31. **Sistema contable**




Fuente: elaboración propia.

Los usuarios tienen la capacidad de interactuar en cualquiera de los sistemas al mismo tiempo, por lo que el diseño de la aplicación cuenta con procesamiento para multiples *threads*, y una sola base de datos relacional que está diseñada para manejar la concurrencia de las solicitudes de los usuarios.

El diseño consiste en realizar una modificación sobre el módulo de contabilidad, ya que será necesario modificar la nomenclatura de las cuentas. Actualmente, las cuentas permiten configurar cuentas contables con dos niveles, pero la necesidad de la empresa es la creación de un tercer nivel, ya que habrá nuevas cuentas contables que necesitan un desglose más detallado, por lo que se le pide al equipo de desarrollo que realice este cambio. La solicitud es generada a través del área financiera quien lo solicita formalmente al equipo de desarrollo a través de una *User Story*, la cual será analizada e ingresada al *product backlog*, para definir la prioridad y el impacto de este cambio.

Es muy importante dar a conocer al lector que el alcance de este documento es el diseño de todo el modelo de desarrollo hasta la puesta en producción, haciendo referencia al control de calidad necesario para poder entregar un producto o servicio de calidad y en el tiempo establecido, con un equipo de trabajo que sigue el método de trabajo que indica DevOps, utilizando integración continua y despliegue continuo, con herramientas que automatizarán todo el flujo de trabajo, se asume que tanto el grupo de desarrolladores como de control de calidad, cuentan con el equipo físico necesario y los conocimientos del uso de las herramientas que se estarán mencionando.

Este documento no indica las configuraciones de cada herramienta que se debe de utilizar ni el desarrollo ni modificación del módulo de contabilidad sobre el cual se está trabando. Tampoco incluye el análisis y diseño de las modificaciones a la aplicación ni la codificación de dicho cambio. La intención de este trabajo es que el lector entienda y conozca un diseño general de cómo utilizar herramientas que automaticen todo este flujo de integración continua.

La metodología de desarrollo que se utiliza es Scrum, como se ha venido viendo desde el inicio del documento, pero no se realizará un manual, estaré utilizando la terminología de dicha metodología en cada una de las fases de desarrollo desde la creación de la solicitud por el usuario interesado, la cual está definida en el glosario de este mismo documento.

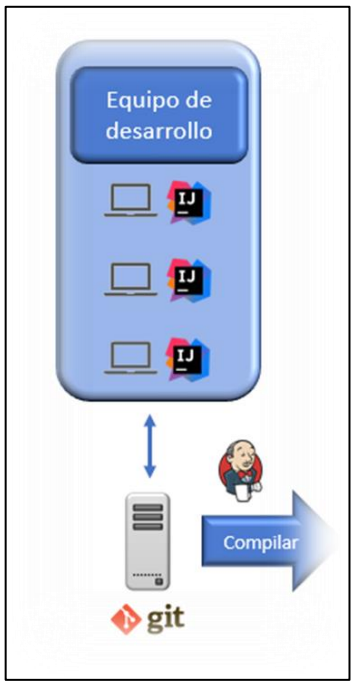
El área financiera necesita que este cambio esté publicado en los próximos dos meses. La *User Story* ya tiene definido los test cases que el área financiera necesita para que el producto funcione adecuadamente, el *scrum master* inicia con la primera reunión para la planificación de *sprint* para esta modificación.

Se aprueba el cambio, se realiza un análisis de impacto y se conforma el equipo de desarrollo que estará trabajando en la modificación de la aplicación definiendo correctamente el Sprint Backlog. La herramienta de desarrollo será IntelliJ IDEA, con el lenguaje de programación Java, que es en el que está desarrollado toda la aplicación.

Se cuenta con un repositorio versionado en GIT, para lo cual se brinda acceso a la última versión a los desarrolladores para que la descarguen y puedan trabajar localmente sobre ella. Se utiliza un sistema de control de versiones distribuido, con el cual cada desarrollador contará con todo el proyecto en cada uno de sus equipos locales de trabajo.

El equipo de desarrollo trabaja a través de los *sprints*, realizando los *Daily Scrum*, para conocer los avances del proyecto y los inconvenientes encontrados a través del desarrollo, hasta que sea finalizado el desarrollo, todos los integrantes suben la última versión al repositorio de GIT, ejecutando los *merge* y resoluciones de conflicto necesarios para crear la nueva versión del producto desarrollado e iniciar con las pruebas estáticas de código y las pruebas unitarias de los cambios desarrollados.

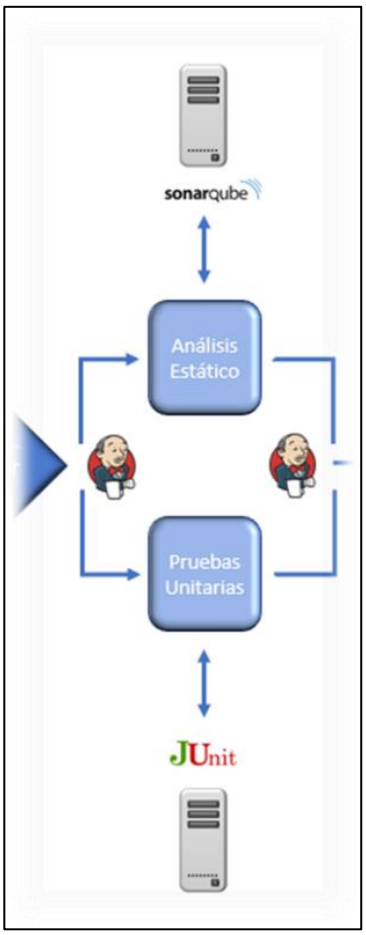
Figura 32. **Versionamiento y control de código fuente**




Fuente: elaboración propia.

Para el análisis estático del código se utilizará SonarQube, y para las pruebas unitarias Junit, las cuales serán ejecutadas automáticamente por Jenkins al finalizar sus configuraciones respectivas. Si Jenkins identifica que los resultados de las herramientas son fallidos, genera un informe a los desarrolladores para que realicen las modificaciones respectivas y vuelvan a crear una nueva versión para ser analizada nuevamente. Pero si los resultados de las herramientas son positivos, Jenkins procederá a ejecutar el paso siguiente, que es la generación del código objeto de la aplicación, a través de la herramienta dedicada para esto que es Maven.

Figura 33. **Análisis de código fuente**




Fuente: elaboración propia.

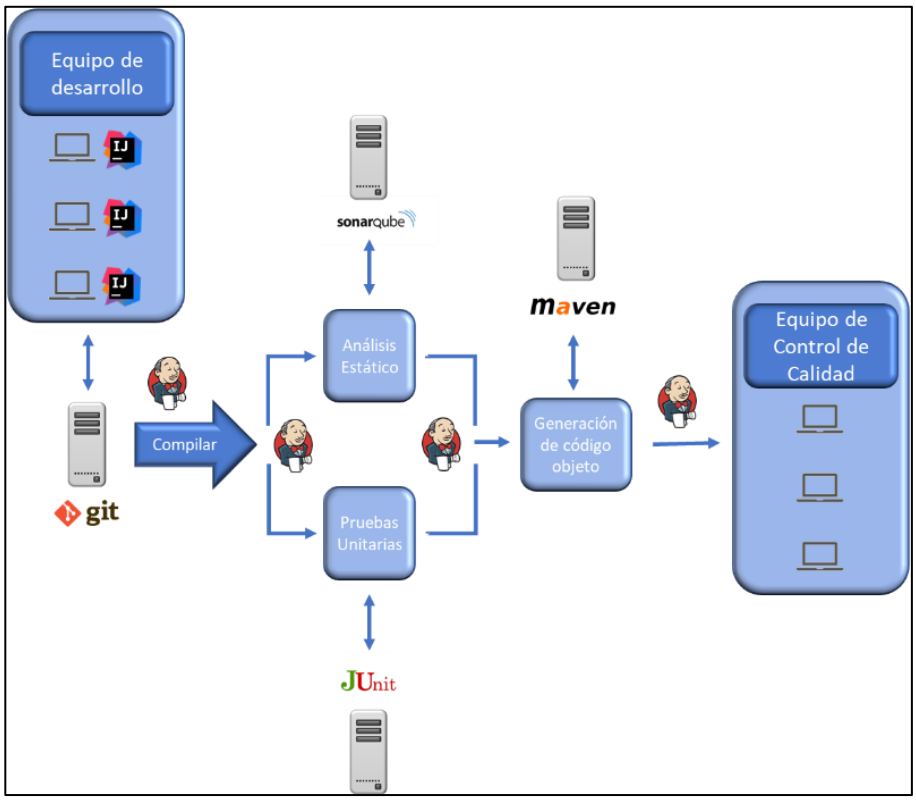
Cuando Jenkins reciba la información de que se generó el código satisfactoriamente, procede a hacer un despliegue al equipo de control de calidad, este lo hará a través de Ansible a un ambiente similar al de producción. Este diseño de pruebas lo veremos en el siguiente punto de este capítulo.

Figura 34. **Generación de código objeto**


Fuente: elaboración propia.

En la siguiente figura, se muestra el diagrama del desarrollo de la modificación solicitada y la generación del código objeto para su análisis por el equipo de control de calidad.

Figura 35. **Diagrama del desarrollo de la modificación contable**




Fuente: elaboración propia.

## 5.2. Automatización de pruebas orquestado por Jenkins

Al igual que la automatización realizada para el equipo de desarrollo, es necesario que el equipo de control de calidad cuente con las herramientas y el flujo adecuado para el análisis de las pruebas de una manera automatizada.

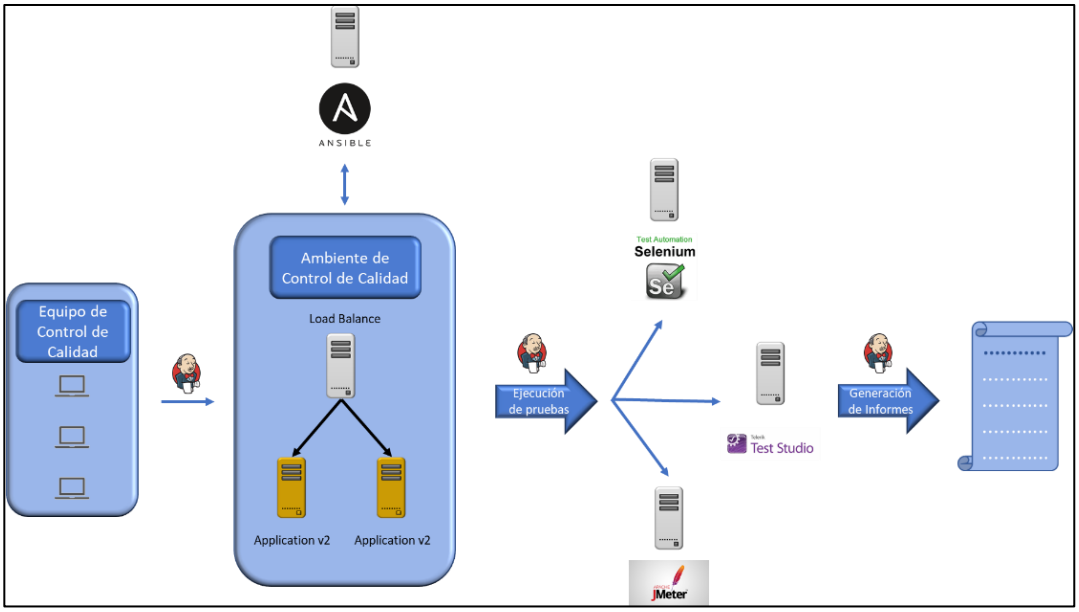
Como se indicó inicialmente, se tiene la premisa de que ya se cuenta con la infraestructura necesaria para realizar el despliegue en los distintos ambientes, incluido el ambiente de pruebas; por lo que las pruebas serán realizadas utilizando una infraestructura igual a la que se encuentra en

producción. Las configuraciones y el despliegue de la información y las configuraciones serán realizadas con Ansible, a través de la llamada de Jenkins, colocando los archivos necesarios para su ejecución automáticamente.

Con la configuración y el ambiente de pruebas listo, y si los resultados de los informes de dicho despliegue son satisfactorios, Jenkins continuará con el flujo, que en este momento son las pruebas de control de calidad; en este caso, se envían una serie de pruebas, como se vio en el capítulo cuarto de este documento, utilizando las herramientas Selenium, Test Studio y JMeter, generando informes al finalizar las pruebas.

Si las pruebas fueron insatisfactorias, Jenkins generará un informe para el equipo de calidad y el equipo de desarrollo para que procedan con los cambios sugeridos respecto a las fallas identificadas; al finalizar, nuevamente el equipo de desarrollo deberá generarse una nueva versión e iniciar el flujo completo desde las pruebas unitarias y pruebas estáticas de código. Si las pruebas fueron satisfactorias, Jenkins procederá automáticamente con el siguiente paso del flujo que será el despliegue del continuo.

Figura 36. **Diagrama del ambiente de control de calidad y sus respectivas pruebas**




Fuente: elaboración propia.

## 5.3. Diseño del despliegue de control de calidad a producción

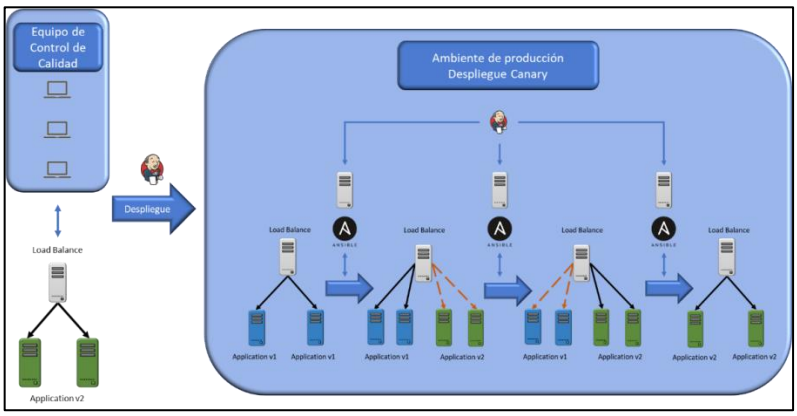
Esta fase del flujo es muy importante, ya que la decisión de cómo realizarlo al ambiente de producción va a depender de cómo esté diseñada la arquitectura de la aplicación, y del análisis de impacto que se realizó al inicio del documento.

Para este caso, se asume que se cuenta con los recursos de infraestructura necesarios para realizar el despliegue de tipo Canary, que como se indicó en el capítulo cuarto, se va colocando en producción paso a paso, haciendo un análisis de la cantidad de usuarios y transacciones de la aplicación.

Si las pruebas son satisfactorias, se da inicio con este paso del flujo. Se cuenta ya con los servidores ya instalados, configurados y revisados como se muestra en la siguiente figura; por lo que Jenkins identifica que los informes de las pruebas fueron satisfactorios e inicia con el despliegue a producción utilizando el despliegue Canary, habilitando los servidores, con el balanceador de producción e iniciando con el proceso de habilitación de tráfico, enviando solamente a un pequeño porcentaje de usuarios.

En este caso, por ser un módulo utilizado por usuarios internos, no hay una gran cantidad de demanda en la transaccionalidad de estas, por lo que se habilita para que lo puedan empezar a utilizar un par de usuarios durante un período establecido, que, para nuestro caso, como ejemplo, se puede poner el sistema a prueba durante dos horas. Al finalizar las dos horas, se habilitará la aplicación para todos los usuarios, mientras el equipo de soporte está monitoreando el comportamiento del uso de la nueva versión de la aplicación, dejándolo a prueba hasta el día siguiente, que es cuando se toma la decisión de deshabilitar por completo la versión anterior.

Figura 37. **Diseño del despliegue a producción**




Fuente: elaboración propia.

De este modo se completa un flujo completo de modificación, que deja un período de *baby sitting* la aplicación para validar que no exista afectación en los días siguientes. En este período de prueba, la versión anterior no se desecha aún. Si se encuentra una falla de menor impacto en la aplicación, el equipo de soporte indicará al equipo de desarrollo los hallazgos de la falla para que inmediatamente se proceda con un cambio de emergencia. Si la falla es de alto impacto, se toma la decisión de realizar un *rollback* (si es factible aún) o realizar un *rollforward* inmediato para mitigar y corregir la falla.

Asumiendo que los resultados fueron satisfactorios, el equipo de desarrollo procede con su reunión de *sprint retrospective*, para aprender de los errores y de las buenas prácticas identificadas al realizar este *sprint*.

# CONCLUSIONES

1. Se ve la necesidad de que las grandes corporaciones que manejan la generación de sus propias aplicaciones de software utilicen una metodología como DevOps, a través de procesos de integración continua ya que la necesidad de cambios debido a que los clientes cada vez son más y más exigentes, es alta, utilicen estas metodologías dentro de la gestión de la generación de software, tanto externo cómo interno.

2. En el mercado, existen muchas herramientas que nos pueden ayudar a integrar las soluciones de todo el proceso de desarrollo de software a través de la integración continua y despliegue continuo, en donde Jenkins puede ser de gran utilidad; ya que es un orquestador fácil de utilizar y que cuenta con una gran cantidad de *plugins* para comunicarse con las distintas herramientas que servirán para automatizar cada parte del proceso de desarrollo.

3. La importancia de un ambiente de control de calidad para que un producto de software sea liberado al ambiente de producción con la menor cantidad de fallas posibles y la automatización de las fases del control de calidad para agilizar la puesta en producción.

4. Se finaliza con un diseño de todo el flujo del proceso de integración continua y despliegue continuo para una empresa de telecomunicaciones para el módulo de contabilidad.

# RECOMENDACIONES

1. Tomar en cuenta que, al momento de querer iniciar con la adopción de la metodología de DevOps, se cuente con el apoyo de las altas autoridades de la empresa; ya que se trata no solo de adaptar la tecnología de esta metodología sino de un cambio cultural en el trabajo entre los equipos de desarrollo, operaciones y control de calidad.

2. Adaptar el uso de las herramientas que servirán para la automatización para cada una de las fases de la integración continua y despliegue continuo que esté de acorde con la arquitectura instalada.

3. Transformar la arquitectura física actual a sistemas virtualizados que permitan el uso de contenedores para el crecimiento y escalabilidad de los servicios de manera más ágil para poder ser adaptados al modelo de integración continua y despliegue continuo.

4. Dentro del flujo de desarrollo de software, es necesario la integración de un equipo de control de calidad que cuente con las herramientas necesarias para realizar las pruebas, al menos según los tipos de pruebas analizados en este documento, automatizándolos para que este ambiente no sea un cuello de botella en la gestión del flujo del desarrollo de software para su puesta en el ambiente de producción.

# BIBLIOGRAFIA


1. Amazon.com. *What is DevOps*. [en línea]. <https://aws.amazon.com/es/devops/what-is-devops/>. [Consulta: 4 de diciembre de 2018].

2. Aprende a tu ritmo. *DevOps Masters Program*. [en línea]. <https://www.udemy.com/learn-devops-scaling-apps-on-premise-and-in-the-cloud/learn/v4/overview>. [Consulta: 4 de diciembre de 2018].

3. Arsys.es. *Automatiza las tareas de desarrollo e integración continua con Jenkins*. [en línea]. <https://www.arsys.es/blog/programacion/jenkins-cloud-integracion-continua/>. [Consulta: 3 de enero de 2019].

4. Blog itaysk.com. *Estrategias de despliegue*. [en línea]. <http://blog.itaysk.com/2017/11/20/deployment-strategies-defined>. [Consulta: 3 de enero de 2019].

5. CIOPERÚ. *8 herramientas para éxito de DevOps*. [en línea]. <https://cioperu.pe/fotoreportaje/20463/8-herramientas-para-el-exito-dev ops/?foto=8>. [Consulta: 3 de enero de 2019].

6. GARZAS, Javier. *¿Despliegue continuo, que implica?.* [en línea]. <https://www.javiergarzas.com/2015/06/implicaciones-despliegue-continuo.html>. [Consulta: 3 de enero de 2019].

7. _____. *¿Qué es Jenkins?.* [en línea]. <http://www.javiergarzas.com/2014/05/jenkins-en-menos-de-10-min.html>. [Consulta: 3 de enero de 2019].

8. _____. *Una lista de herramientas para pruebas software (funcionales y de software libre)*. [en línea]. <https://www.javiergarzas.com/2012/03/herramientas-para-pruebas-software.html>. [Consulta: 3 de enero de 2019].

9. Git scm.com. *Control de versiones*. [en línea]. <https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones>. [Consulta: 3 de enero de 2019].

10. Jenkins. *DevOps with AWS CodePipeline, Jenkins and AWS CodeDeploy*. [en línea]. <https://www.udemy.com/ci-and-cd-with-aws-codepipeline-jenkins-and-aws-codedeploy / learn / v4 / over view>. [Consulta: 3 de enero de 2019].

11. _____. *Jenkins continuous integration and delivery server*. [en línea]. <https://hub.docker.com/r/jenkins/jenkins/>. [Consulta: 3 de enero de 2019].

12. _____. *Jenkins Ecosystem*. [en línea]. <https://awesome-tech.readthedocs.io / images / jenkins_is_the_hub_CD_Devops. png>. [Consulta: 3 de enero de 2019].

13. _____. *Learn continuous integration with Jenkins all in one guide*. [en línea]. <https://www.udemy.com/learn-continuous-integration-with-jenkins-all-in-one-guide/>. [Consulta: 3 de enero de 2019].

14. KIM, Gene; BEHR Kevin; SPAFORD, George. *The phoenix project*. Portland, Oregon, USA: IT Revoluton Press, 2013. 170 p.

15. KIM, Gene; HUMBLE, Jez; DEBOIS, Patrick; WILLIS, John. *The DevOps handbook*. USA: IT Revoluton Press, 2016. 480 p.

16. Learn DevOps. *Scaling apps on-premises and in the cloud*. [en línea]. <https://www.udemy.com/learn-devops-scaling-apps-on-premise-and-in-the-cloud/learn/v4/overview>. [Consulta: 4 de noviembre de 2018].

17. Panel.es. *Tipos de pruebas de software*. [en línea]. <https://www. panel.es / blog / software-qa-cuales-son-los-tipos-de-pruebas-software/>. [Consulta: 3 de enero de 2019].

18. Paradigma digital. *Qué es DevOps (y sobre todo qué no es DevOps)*. [en línea]. <https:// www.paradigmadigital.com/techbiz/que-es-devops-y-sobre-todo-que-no-es-devops/>. [Consulta: 4 de diciembre de 2018].

19. PARI, Julio. *Integración continua en proyectos ágiles de software*. [en línea]. <http://blog.juliopari.com/integracion-continua-en-proyectos-agiles-de-software/>. [Consulta: 3 de enero de 2019].

20. Pinimg.com. *Agile DevOps*. [en línea]. <https://i.pinimg.com/originals/ 83/e7/2d/ 83e72d5d9144e7e0cc3295226b9ac974.jpg>. [Consulta: 3 de enero de 2019].

21. Qvision.com. *Tipos de pruebas funcionales de software*. [en línea]. <http://www.qvision.com.co/pruebas-funcionales-manual-y-automatizada/>. [Consulta: 3 de enero de 2019].

22. RIBAS, Ester. *Herramientas imprescindibles para desarrolladores Java*. [en línea]. <https://www.iebschool.com/blog/herramientas-desarrolladores-java-tecnologia/#codigo_arbierto>. [Consulta: 3 de enero de 2019].

23. Testeando software. *Las mejores herramientas para realizar pruebas de software*. [en línea]. <https://testeandosoftware.com/las-mejores-herramientas-para-realizar-pruebas-de-software/>. [Consulta: 3 de enero de 2019].

24. Tutoriales. *Docker Crash Course for busy DevOps and Developers*. [en línea]. <https://www.udemy.com/docker-tutorial-for-devops-run-docker-containers/learn/v4/overview>. [Consulta: 4 de noviembre de 2018].

25. Udemy.com. *DevOps Mastering*. [en línea]. <https://www.udemy.com/mastering-devops/learn/v4/overview>. [Consulta: 4 de noviembre de 2018].

26. Wikipedia. *DevOps*. [en línea]. <https://es.wikipedia.org/wiki/DevOps>. [Consulta: 4 de diciembre de 2018].

27. _____. *Jenkins*. [en línea]. <https://es.wikipedia.org/wiki/Jenkins>. [Consulta: 3 de enero de 2019].