ESTUDIO DE UNA ALTERNATIVA DE INTEGRACIÓN CONTINUA QUE SOPORTE EL DESARROLLO DE UNA INFRAESTRUCTURA TI DE SERVICIOS DE INFORMACIÓN DEL TRANSPORTE PÚBLICO DE PASAJEROS

ANGEL DE JESUS VALBUENA NAVARRO

ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
UNIVERSIDAD INDUSTRIAL DE SANTANDER
BUCARAMANGA
2017

ESTUDIO DE UNA ALTERNATIVA DE INTEGRACIÓN CONTINUA QUE SOPORTE EL DESARROLLO DE UNA INFRAESTRUCTURA TI DE SERVICIOS DE INFORMACIÓN DEL TRANSPORTE PÚBLICO DE PASAJEROS

ANGEL DE JESUS VALBUENA NAVARRO

Trabajo de Grado para optar al título de
Ingeniero de Sistemas

Director
GABRIEL PEDRAZA FERREIRA, PhD

ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
UNIVERSIDAD INDUSTRIAL DE SANTANDER
BUCARAMANGA
2017

DEDICATORIA

Dedico este trabajo a mis padres Ángel Valbuena Gómez y Luz Mary Olmos Pérez,
A mi familia que ha sido el motor de mi vida,
A la familia Cotte Gamboa que me acogió como uno de los suyos,
A mis profesores que con sus conocimientos contribuyeron en mi formación,
A Angélica Basto que ha estado animándome en esta culminación de etapa.

Angel Valbuena Navarro

CONTENIDO

LISTA DE FIGURAS

GLOSARIO

Extreme programming(XP): Es una metodología de desarrollo software que se enfoca en la adaptación a los cambios de requisitos en cualquier punto del proyecto, con el fin de mejorar la productividad del mismo y garantizando la calidad del software desarrollado.

Metodologías Ágiles: Son una serie de técnicas para la gestión de proyectos software que se caracteriza por el seguimiento de principios establecidos en un manifiesto con el fin de aumentar la eficiencia de las personas involucradas en el proyecto y minimizar el coste y el impacto de las tareas que no son totalmente imprescindibles para conseguir el objetivo del proyecto.

Feedback: Es un método de control en el cual los resultados obtenidos de una actividad en el desarrollo del proyecto son introducidos nuevamente en el sistema.

Build: Es una actividad del desarrollo software en la que se lleva a cabo la construcción de código,con el fin de convertir el código en artefactos autónomos que se pueden ejecutar en una computadora.

Commit: Es la actividad de registrar cambios en un repositorio de un sistema de control de versionesde tal forma que es posible colocar visible todos los cambios a otros usuarios.

Package: Son una serie de programas precompilados que se distribuyen conjuntamente, con el fin de complementar o requerir a otros programas de una aplicación proporcionando de esta forma que se pueda distribuir en paquetes.

Jar: Es un tipo de archivo que permite ejecutar aplicaciones escritas en lenguaje java, están comprimidos en formato ZIP y su extensión es cambiada a .jar.

RESUMEN

TÍTULO: ESTUDIO DE UNA ALTERNATIVA DE INTEGRACIÓN CONTINUA QUE SOPORTE EL DESARROLLO DE UNA INFRAESTRUCTURA TI DE SERVICIOS DE INFORMACIÓN DEL TRANSPORTE PÚBLICO DE PASAJEROS*

AUTOR: ANGEL DE JESUS VALBUENA NAVARRO**

PALABRAS CLAVE: Integración continua, Automatización, Caso de Estudio.

DESCRIPCIÓN: En la actualidad los equipos de desarrollo de software utilizan herramientas tecnológicas y metodologías que les facilitan la ejecución de diferentes actividades, entre las cuales cabe mencionar la verificación, compilación, las pruebas, la construcción, el empaquetado y el despliegue de aplicaciones con el fin de conseguir una buena calidad de software. Algunas de estas tareas son desarrolladas de forma manual o semiautomatizada, contradiciendo un paradigma del desarrollo software moderno como lo es, la entrega frecuente al cliente y al equipo de desarrollo software, dificultando la obtención de productos software de buena calidad en el menor tiempo y coste posible.

En este trabajo se diseña, se implementa y se analiza una alternativa de integración continua en un caso de estudio relacionado con una infraestructura TI que soporta servicios de información al transporte público colombiano, haciendo posible evidenciar una mejora en el desarrollo de un proyecto software al llevar a cabo esta alternativa.

Para la implementación de dicha alternativa, se contempla la identificación y ejecución de las herramientas primordiales que permiten la integración continua. En las diferentes etapas de este proceso se muestra la ventaja de automatización, desde la puesta en marcha del proyecto hasta su puesta en producción. Adicionalmente se llevarán a cabo el despliegue de la aplicación del caso de estudio teniendo en cuenta la portabilidad, usabilidad y el poco consumo de recursos de la máquina.

*Trabajo de grado.

** Facultad de Ingenierías Fisico-Mecanicas, Escuela de Ingeniería de Sistemas e Informática. Director Gabriel Pedraza Ph.D.

ABSTRACT

TITLE: STUDY OF A CONTINUOUS INTEGRATION ALTERNATIVE THAT SUPPORTS THE DEVELOPMENT OF AN IT INFRASTRUCTURE OF PUBLIC TRANSPORT INFORMATION SERVICES FOR PASSENGERS*

AUTHOR: ANGEL DE JESUS VALBUENA NAVARRO**

KEYWORDS: Continuous Integration, Automation, Case Study.

At present, software development teams use technological tools and methodologies that facilitate the execution of different activities, among which the verification, compilation, testing, construction, packaging and deployment of applications in order to Get a good quality software. Some of these tasks are developed manually or semi-automated, contradicting a paradigm of modern software development as it is, frequent delivery to the client and the software development team, making it difficult to obtain good quality software products in the shortest time and cost possible. In this paper, an alternative of continuous integration is designed, implemented and analyzed in a case study related to an IT infrastructure that supports information services to Colombian public transport, making it possible to demonstrate an improvement in the development of a software project by carrying Out this alternative. For the implementation of this alternative, it is contemplated the identification and execution of the primordial tools that allow the continuous integration. In the different stages of this process the advantage of automation is shown, from the start up of the project until its production. In addition, the deployment of the application of the case study will be carried out taking into account the portability, usability and the low consumption of resources of the machine.

*Bachelor Thesis.

** Faculty of Physical-Mechanical Engineering. Shool of Engineering and Computer Science.. Gabriel Pedraza Ph.D.

INTRODUCCIÓN

El desarrollo de software moderno exige que los equipos de desarrollo cumplan con objetivos como el desarrollo de productos de calidad, la generación adecuada de los artefactos de despliegue y la capacidad de entrega continua de los productos (*continuous delivery).* Contrario a una percepción generalmente aceptada en la cual el desarrollo de software solamente es una tarea de programación, el desarrollo de software requiere un conjunto de actividades que incluyen entre otras las pruebas, la generación de artefactos, el despliegue de los mismos, etc. Estas actividades son generalmente realizadas de forma manual, lo que conlleva a un aumento de costos, susceptibilidad a los errores humanos además de la incapacidad de llevar los productos a producción en el tiempo deseado.

Por otra parte el desarrollo software es una tarea que en equipos, cada miembro del equipo puede llevar a cabo parte de estas actividades, pero se necesita la realización de las mismas de forma que integre la labor realizada por cada miembro del equipo. Por esta razón han surgido nuevas prácticas que facilitan al equipo de desarrollo la ejecución de sus tareas de manera eficiente y óptima, entre ellas una de las más importantes es la noción de integración continua. El proceso integración continua tiene dos objetivos principales, el primero es integrar el trabajo realizado por los diferentes miembros del equipo, el segundo es tratar de automatizar en lo posible el conjunto de actividades involucradas en un proceso de desarrollo software. La automatización de algunas actividades proporciona diversas ventajas entre ellas se pueden citar: menos errores, generación rápida de artefactos, construcción de forma frecuente, testing, compilación, empaquetado y distribución de forma eficiente, deployment, pruebas automáticas, gestión de dependencias.

En este sentido, el presente trabajo plantea el estudio de una alternativa de un entorno de integración continua, de un caso de estudio que está basado en el desarrollo de una infraestructura TI. Para lo cual expone el planteamiento de un prototipo de diseño, la selección de las herramientas para su implementación y ejecución, y los resultados de su experimentación. Con el fin de mostrar la ventaja de automatización de esta práctica, así como las ventajas mencionadas anteriormente, permitiendo una mayor confianza y seguridad en el equipo de desarrollo y por consiguiente facilitando la elaboración y la integración de los componentes del caso de estudio.

1. PLANTEAMIENTO Y JUSTIFICACIÓN DEL PROBLEMA

Cuando un equipo de desarrollo software lleva a cabo un proyecto es inevitable que se presenten errores en sus diferentes etapas, es por ello que en los últimos años han surgido nuevas prácticas y herramientas que permiten la mejora continua en los procesos de desarrollo software. La integración continua ha sido una de las prácticas más importantes, esto se debe a que permite automatizar algunas actividades de desarrollo de software que son fundamentales, entre las cuales podemos mencionar: la compilación, la gestión de dependencias, las pruebas, el empaquetado, el despliegue y la integración del código de los miembros del equipo de desarrollo. Todos de un nivel de complejidad alto, pero sin duda alguna, es en el despliegue donde se presenta la mayor complejidad, debido a que para desplegar un nuevo servicio a través de un conjunto de componentes, es necesario analizar el estado actual de la plataforma, determinar la dependencia y compatibilidad entre los componentes a instalar y los existentes, la compatibilidad de los componentes con el sistema operativo y con los recursos disponibles.

Por otra parte, hoy en día el *Internet de las Cosas* o IoT (por su sigla en inglés *Internet of Things*) es una tendencia que ha tenido un gran auge, ya que estamos rodeados de dispositivos con capacidades de cómputo y comunicación que pueden ser utilizados para diferentes tipos de aplicaciones y que han permitido avances en diversos dominios como la medicina, la biología, la astronomía y hasta en la satisfacción de necesidades cotidianas como el entretenimiento, siendo sus principales campos de acción los objetos cotidianos conectados con internet y Smart Cities. Smart cities son aquellas ciudades que utilizan las tecnologías de la información y la comunicación con el propósito de alcanzar un desarrollo sostenible que permita una mejora en la calidad de vida de los ciudadanos.

En este contexto, con el fin de mejorar la calidad de vida de las personas las Smart Cities utilizan nuevas tendencias tecnológicas entre ellas IoT y una posible aplicación apunta a la mejora del servicio de transporte urbano de pasajeros en las ciudades de Colombia.
Sin embargo, desarrollar software con IoT no es una tarea sencilla debido a la necesidad de un ambiente tecnológico unificado y de mecanismos adecuados que permitan el desarrollo de sistemas intuitivos y autónomos que involucren automatización.

Por esta razón, este proyecto pretende realizar el estudio de una alternativa de integración continua en el marco de la implementación de una infraestructura TI, que permita la reducción de errores y de procesos repetitivos manuales. Logrando así, una mayor confianza y seguridad en el equipo de desarrollo para mejorar los componentes software desarrollados y por consiguiente el servicio de transporte público de pasajeros en Colombia.

2. OBJETIVO GENERAL Y ESPECÍFICOS

2.1. OBJETIVO GENERAL

Diseñar y evaluar una alternativa de integración continua para el desarrollo de los componentes de una infraestructura TI que ofrece servicios de información al transporte público de pasajeros.

2.2. OBJETIVOS ESPECÍFICOS

- Realizar el estudio de una alternativa de integración continua en el marco de la implementación de una infraestructura TI.
- Caracterizar un caso de estudio a utilizar en la experimentación del proceso de integración continua.
- Definir un proceso de integración continua para un caso de estudio de la infraestructura.
- Determinar los criterios para validar el proceso de integración continua en el caso de estudio.

3. METODOLOGÍA

La metodología se desarrollará en cuatro fases, las cuales son:

Fase 1: Exploración Tecnológica

En la primera fase se explorara el ambiente tecnológico del proyecto. Las actividades que se llevarán a cabo en esta fase incluyen:

1. Identificar las herramientas tecnológicas pertinentes en el caso de estudio.
2. Interactuar con las tecnologías identificadas.

Fase 2: Diseño caso de estudio

La segunda fase propone el diseño de una infraestructura que soporte procesos de integración continua. Para el desarrollo de esta fase se deben llevar a cabo las siguientes actividades:

1. Identificar las características del caso de estudio.
2. Diseño de un proceso de integración continua para el caso de estudio.
3. Validación del prototipo de diseño.

Fase 3: Determinación de criterios de evaluación

En la tercera fase se analizará y determinará los criterios para la validación del caso de estudio. En esta fase se desarrollaran las siguientes actividades:

1. Análisis de las diferentes etapas realizadas en la infraestructura en estudio.
2. Observación de la interacción de los diferentes componentes en el caso de estudio.
3. Establecer criterios para validar la arquitectura en estudio.

Fase 4: Experimentación

En la cuarta fase se realizará la experimentación usando el caso de estudio, cabe aclarar que en esta instancia se puede volver a la tercera fase. Para desarrollar la tercera fase se requiere de las siguientes actividades:

1. Interactuar con los diferentes componentes del caso de estudio.
2. Ejecución de pruebas en cada etapa del proceso desarrollado en la infraestructura.

4. MARCO TEÓRICO

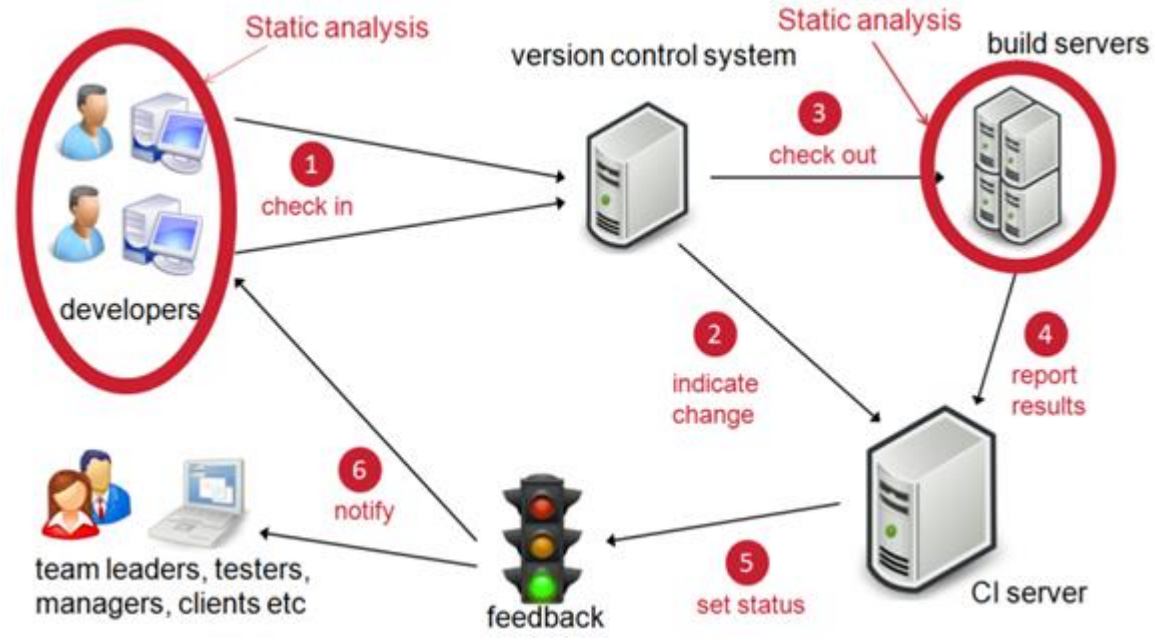
4.1. INTEGRACIÓN CONTINUA

La integración continua es una práctica de extreme programming(XP) que fue concebida y publicada alrededor del año 2001 por Martin Fowler, quien es un desarrollador de software especializado en análisis y diseño orientado a objetos, UML, patrones de diseño, y metodologías de desarrollo ágil, incluyendo programación extrema. Hay varias definiciones de integración continua pero una de las más aceptadas es la de su creador, en el 2006 Martin Fowler la definió de la siguiente manera:

> *"**Práctica de desarrollo software donde los miembros del equipo integran su trabajo frecuentemente, al menos una vez al día. Cada integración se verifica con un build automático (que incluye la ejecución de pruebas) para detectar errores de integración tan pronto como sea posible."*** [1]

En base a esta definición, la integración continua tiene como objetivo realizar comprobaciones automáticas y periódicas en las diferentes etapas de un proyecto software para detectar errores lo antes posible y mejorar la calidad del mismo. Para dicho objetivo cuenta con herramientas que al interactuar constituyen un entorno de integración continua.

Figura 1. Entorno de Integración Continua



Fuentes: https://hyperneon.wordpress.com/

En la figura se observa que los desarrolladores alojan su código en un sistema de control de versiones, el cual es analizado y enviado a un servidor de integración continua donde previamente han sido desarrolladas las pruebas (funcionales y no funcionales) por un tester, finalmente el servidor de integración continua envía el estado de la prueba por medio de feedback al tester y a los desarrolladores. Es importante resaltar aunque no es notoria en la figura que desde la puesta en marcha del proyecto se hace uso de una herramienta para la construcción y gestión del mismo.

La implementación de un entorno de integración continua trae consigo ventajas como:

- Reducir procesos repetitivos y manuales.
- Reducir riesgos y tiempos de desarrollo.
- Lograr una mayor autoconfianza y seguridad en el equipo de desarrollo.
- Generación de análisis y presentación de informes sobre el estado del código.
- Reducción de costes en el proceso de desarrollo y despliegue.
- Detectar errores con mayor rapidez y antelación. Esto provocará que sea más sencillo de corregir y por lo tanto más barato. [2][3]

Uno de los objetivos centrales de la integración continua es la automatización de las pruebas en las diferentes etapas de un proyecto, debido a esto se hablará en el siguiente apartado de las pruebas que se llevan a cabo en un proyecto software.

4.2. PRUEBAS DE SOFTWARE

Entre las etapas que se llevan a cabo en el ciclo de vida de un producto software, está el análisis, el diseño, la implementación, el desarrollo y las pruebas. Todas de gran importancia pero la fase de pruebas es una fase decisiva, pues en base a los resultados obtenidos en esta, el equipo de desarrollo tiene insumos para decidir si el sistema software tiene un nivel de calidad que permita para su puesta en producción.

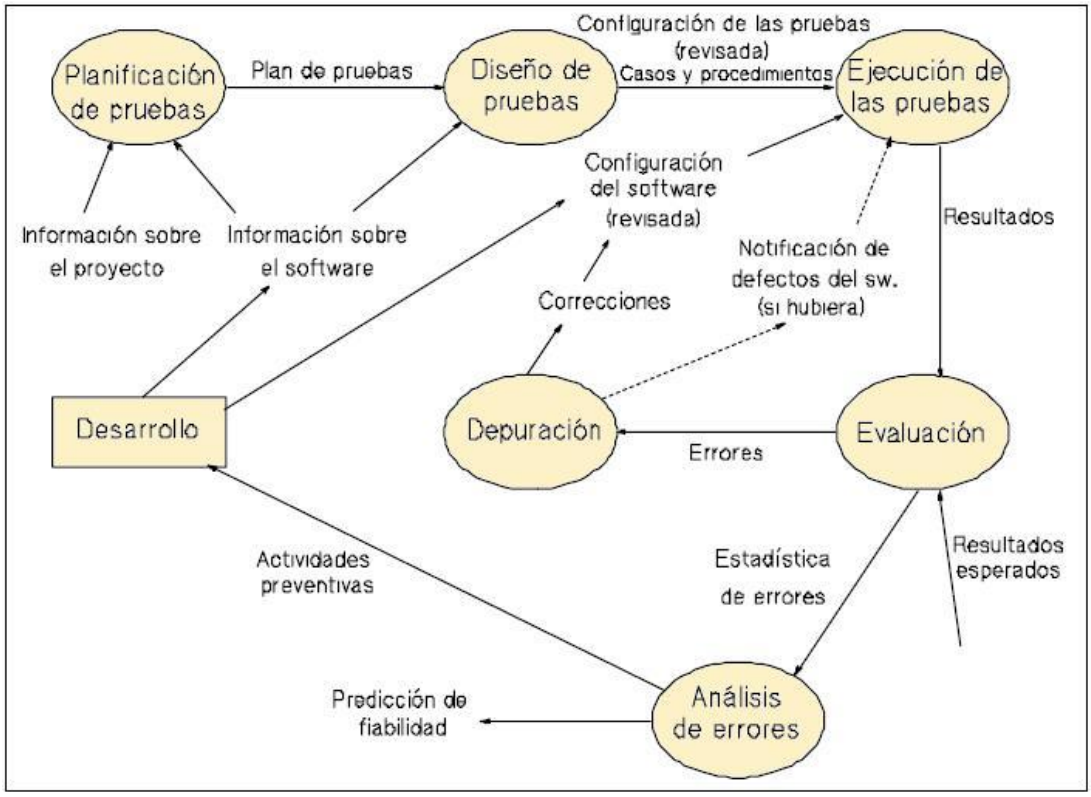
Las pruebas del software básicamente consisten en ejecutar un proceso con el fin de encontrar errores con circunstancias previamente especificadas, y con el fin de observar, registrar y analizar los resultados obtenidos. El éxito de las pruebas radica en descubrir por lo menos un error, y traducido a una empresa de desarrollo el descubrimiento de un error significa un éxito para la mejora de la calidad del producto.

Las pruebas de software se basan en tres aspectos importantes:

- Que el software no haga lo que debe hacer.
- Que el software haga lo que debe hacer.
- Que el software cumpla con las propiedades no funcionales.

Para la ejecución de las pruebas previamente se desarrollan los casos de prueba, que son las condiciones de ejecución y los resultados esperados desarrollados para un proyecto en específico.

Figura 2. Ciclo de vida de las pruebas de software



Fuente: http://adsig7.blogspot.com.co/

Para el diseño de una prueba existen varios enfoques, dentro de los cuales se pueden resaltar:

- El enfoque estructural o de caja blanca, que se centra en la estructura interna del programa.
- El enfoque funcional o de caja negra, que se centra en las funciones, entradas y salidas, sin acceso al código fuente de las aplicaciones.

En base a estos enfoques en el desarrollo software se realizan varias pruebas como son las pruebas unitarias, las pruebas de integración, las pruebas de regresión, las pruebas de humo, las pruebas de carga y stress, pruebas del sistema, entre otras. En esta alternativa de integración continua se llevarán a cabo pruebas unitarias, pruebas de carga y stress, y pruebas de integración por medio de pruebas a servicios web.

Figura 3. Niveles de pruebas

Niveles de pruebas				
Test	**Objetivo**	**Participantes**	**Ambiente**	**Método**
Unitario	Detectar errores en los datos, lógica, algoritmos	Programadores	Desarrollo	Caja Blanca
Integración	Detectar errores de interfaces y relaciones entre componentes	Programadores	Desarrollo	Caja Blanca, Top Down, Bottom Up
Funcional	Detectar errores en la implementación de requerimientos	Testers, Analistas	Desarrollo	Funcional
Sistema	Detectar fallas en el cubrimiento de los requerimientos	Testers, Analistas	Desarrollo	Funcional
Aceptación	Detectar fallas en la implementación del sistema	Testers, Analistas, Cliente	Productivo	Funcional

Fuente: http://materias.fi.uba.ar/7548/PruebasSoftware.pdf

Como se ha podido observar, el solo planteamiento de las pruebas conlleva un gran trabajo por parte de los analistas y tester en un equipo de desarrollo, ahora si se añade a esto que las pruebas sean ejecutadas de forma manual, el ciclo de la prueba sería muy largo y propenso a errores, el personal encargado de esta área en la empresa le llevará mucho tiempo esta fase, sin realimentación y con baja frecuencia de los resultados obtenidos en las pruebas. Por esta razón en el desarrollo de software moderno, surge la necesidad de automatizar pruebas en las diferentes etapas del proyecto, y una posible alternativa para solventar esta problemática es la integración continua.

4.3. HERRAMIENTAS DE GESTIÓN DE PROYECTOS

La fase del desarrollo software de construcción de código es una tarea compleja, debido a que es necesario gestionar librerías que permitan el correcto funcionamiento del código que se esté realizando. Además de que las actividades que se llevan a cabo en los proyectos son realizadas por un equipo de desarrollo, haciendo necesario la integración del código, complicando un poco más el panorama. Anteriormente la solución propuesta por las empresas de desarrollo fue tener a una persona o un grupo de personas dedicada exclusivamente a realizar este proceso, pero esto tiene ciertas desventajas como pérdidas de tiempo para aprender y adaptarse a las peculiaridades de un nuevo proyecto,

compilar y generar los artefactos de despliegue, revisar e incluir las librerías que utiliza cada código y las dependencias de compilación, persistiendo en un proceso de construcción tedioso.
Para resolver esta problemática se han implementado herramientas que permiten la gestión y construcción de proyectos, de tal forma que se propicie un ambiente de trabajo organizado por medio de automatización de tareas de desarrollo como la compilación, la gestión de las dependencias, la ejecución de las pruebas, la generación de artefactos de despliegue, etc.
El uso de una herramienta de construcción y gestión de proyecto software brindan los siguientes beneficios:

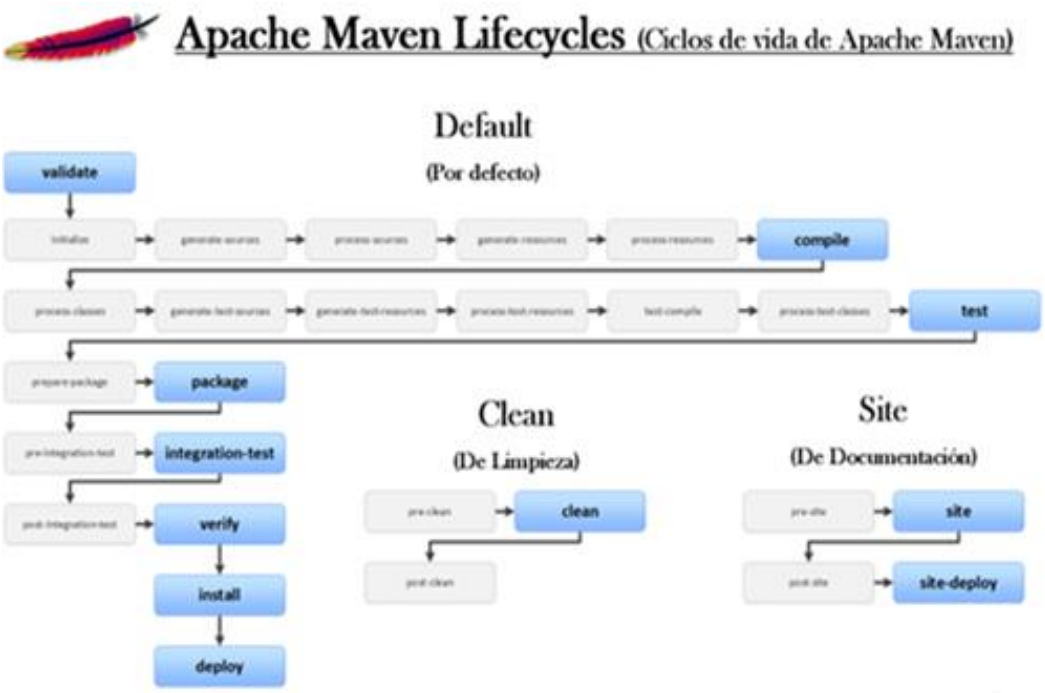
- Automatización de tareas repetitivas, como *build*, *package*, etc.
- Mejor organización del proyecto.
- Tener información sobre la construcción del proyecto de manera automática, permitiendo que el equipo esté enterado si algún *build* ha fallado.
- Mejora en la calidad del producto.

En la actualidad existen varias herramientas de este tipo como Ant, Gradle, Maven entre otras. En el siguiente ítem se habla acerca de algunas de estas herramientas, y se profundizará en Maven que fue la herramienta seleccionada en este proyecto.

4.3.1. Maven

Maven es una herramienta *open source* creada por la fundación Apache por Jason van Zyl, en 2002 para la gestión y construcción de proyectos, para ello se basa en un fichero central pom.xml, donde se definen todas la tareas que se necesitan en el proyecto.
Maven define en sucesivas fases el ciclo de vida de un proyecto, de manera que la ejecución de una de las fases de este ciclo implica la ejecución de todas las fases anteriores.

Figura 4. Ciclo de vida maven



Fuente: http://jarroba.com/maven/

Entre las ventajas que brinda maven se pueden mencionar:

- Organización de todos los proyectos de la empresa.
- Disponer de los últimos jar en un repositorio centralizado.
- Posibilidad de elección de un patrón determinado en función de los requerimientos del proyecto.

Pero Maven también tiene ciertas desventajas, como:

- Su dependencia de que haya internet o al menos red. A pesar de que Maven tiene una ejecución offline, pero no funciona del todo bien.
- A veces coinciden las actualizaciones de plugins de maven con algún tipo de problema de red o el estar off-line nos hace imposible compilar.

4.3.2. Ant

Ant es una herramienta gratuita(*Open Source*) desarrollada en java por Apache software fundation, básicamente fue creado por James Duncan Davidson, nació como un simple intérprete que cogía un archivo XML “build file” para compilar Tomcat independiente de la plataforma sobre la que operaba. Su primera versión fue lanzada oficialmente como un

producto independiente el 19 de julio de 2000. Es usada en programación para la realización de tareas mecánicas y repetitivas, en su mayor parte durante la fase de compilación y construcción, para ello ant se basa un fichero xml en el que colocamos las tareas que queremos que se ejecuten, dicho fichero se le llama comúnmente build.xml.

Ant tiene limitaciones como:

- Al ser una herramienta basada en XML todos los archivos Ant deben ser escritos en XML.
- Problema en los proyectos muy grandes, son difíciles de mantener.
- No es un lenguaje para un flujo de trabajo general, y no debería ser usado como tal.

Ant brinda ciertas ventajas a los desarrolladores, entre ellas se puede mencionar:

- Ant está escrito en XML y Java, por lo que ofrece una solución interoperable al nivel de sistema operativo (debido a Java) y configuraciones descriptivas (debido a XML).
- Es fácilmente extensible e integrable con muchas herramientas (p.e: Editores, IDE,…)
- No presenta dependencia del SO en el que trabaja(portabilidad).
- Idónea como solución multi-plataforma.

4.4. SISTEMA DE CONTROL DE VERSIONES

En el desarrollo software es necesario tener varias versiones de un proyecto, porque la aplicación o proyecto está propenso a errores y si en un determinado momento ocurre un error fatal el equipo necesitará retomar la implementación de la versión anterior.
Anteriormente las empresas guardaban los hitos (una especie de versión) del proyecto en archivos comprimidos y medios de almacenamiento como disquetes y CDs. Cuando se presentaba un error y era necesario volver a la versión anterior, tenían que abrir el archivo o los archivos necesarios para comparar las versiones, realizando un proceso tedioso, gastando tiempo y recursos en una versión estable del proyecto. Para solventar y facilitar este proceso se crearon los sistemas de control de versiones.

Un sistema de control de versiones básicamente es una herramienta que registra todos los cambios hechos en uno o más proyectos, guardando así versiones del producto en todas sus etapas del desarrollo. Las versiones son como fotografías que registran su estado en ese momento del tiempo y se van guardando a medida que se hacen modificaciones al código fuente.[4]

Los repositorios tradicionales manejan una estructura cliente-servidor: el servidor es el que mantiene y controla el versionado de los ficheros, y el cliente se descarga copias del servidor para que sean modificadas y subidas posteriormente con el fin de que sean vistas por el resto de usuarios. A mediados del 2000 se crearon los repositorios distribuidos, estos se basan en una estructura diferente. Los clientes ya no se descargan una copia del repositorio, sino que lo clonan, comportándose como cliente y servidor al mismo tiempo, es decir, que la copia podría comportarse a su vez como servidor ante otros clientes (construyendo así una estructura distribuida).
Algunos de los sistemas de control de versiones más usados son Git, Mercurial, y Subversion.

A continuación hablaremos de Mercurial y de Git que fue el sistemas de control de versión es usado en este proyecto.

4.4.1. Git

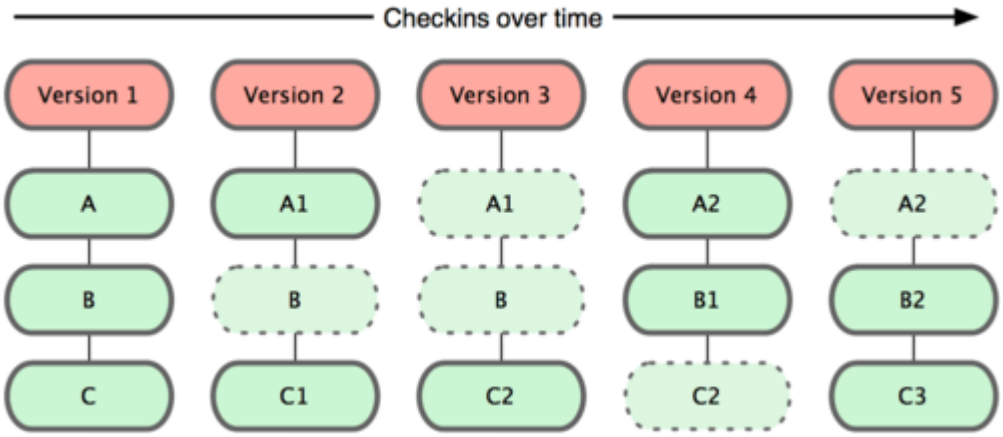
Fue desarrollado por Linus Torvalds, el mismo padre del kernel Linux, en el año 2005. Git es un sistema de control de versiones distribuido open source que surge como alternativa a un control de versiones privativo que se usaba en ese entonces para el kernel llamado BitKeeper. Git está basado en la eficiencia y confiabilidad de las diferentes versiones de aplicaciones y por ello proporciona herramientas que facilitan el trabajo en equipo. Fue liberado bajo una licencia GNU GPLv2 y su última versión estable fue publicada a inicios de Abril de este año. Se ha convertido en uno de los más usados alrededor del mundo.

Entre las características más importantes de Git se encuentran:

- Rapidez en la gestión de ramas, debido a que Git fusiona los cambios mucho más rápidamente y frecuentemente de lo que se escribe originalmente.
- Gestión distribuida, los cambios se importan como ramas adicionales y pueden ser fusionados de la misma manera como se hace en la rama local.
- Gestión eficiente de proyectos grandes.
- Realmacenamiento periódico en paquetes.

Git se diferencia de los demás VCS(Por sus siglas en inglés *sistemas de control de versiones*) en la forma como modela los datos. La mayoría de los sistemas almacenan la información en archivos o almacenan los datos como cambios de cada archivo respecto a una versión base. Git modelos los datos como un conjunto de fotografías de un mini sistema de archivos. Cada vez que se confirma un cambio, él hace una foto del aspecto de todos tus archivos en ese momento, y guarda una referencia a esa instantánea.[5]

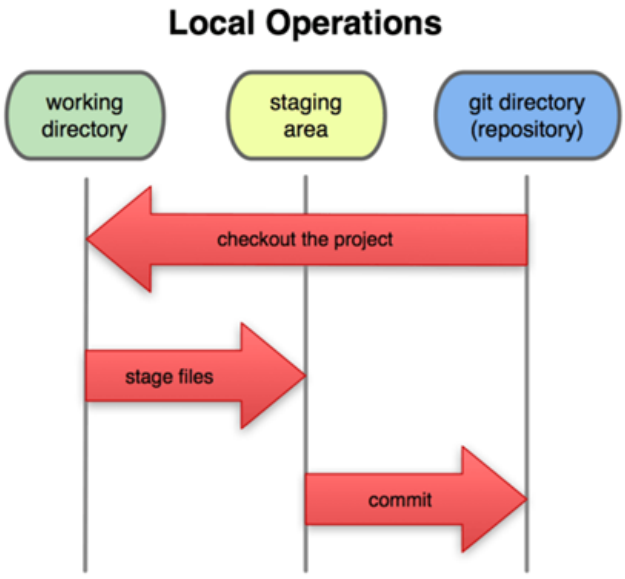
Figura 5. Proceso de modelamiento de los datos de Git



Fuente: https://git-scm.com/book/es/v1/Empezando-Fundamentos-de-Git

Git tiene tres estados principales en los que se pueden encontrar los archivos: confirmado (committed), modificado (modified), y preparado (staged). Confirmado significa que los datos están almacenados de manera segura en la base de datos local. Modificado significa que se ha modificado el archivo pero todavía no lo has confirmado a tu base de datos. Preparado significa que se ha marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación. Lo que conlleva a las tres secciones principales de un proyecto de Git: el directorio de Git (Git directory), el directorio de trabajo (working directory), y el área de preparación (staging area).

Figura 6. Estados de Git



Fuente: https://git-scm.com/book/es/v1/Empezando-Fundamentos-de-Git

El flujo de trabajo básico en Git se puede resumir en los siguientes pasos:

- Modificar una serie de archivos en un directorio de trabajo.
- Preparar los archivos, añadiendoles a un área de preparación.
- Confirmar los cambios, lo que toma los archivos tal y como están en el área de preparación, y almacena esas instantáneas de manera permanente en tu directorio de Git.

4.4.2. Mercurial

Mercurial fue desarrollado por Matt Mackall en el 2005, y al igual que Linus, Matt buscaba una alternativa a BitKeeper para el control de versiones del kernel Linux. También es liberado bajo una licencia GNU GPL v2 y su última versión estable fue publicada en Abril de este mismo año.

Está implementado principalmente haciendo uso del lenguaje de programación Python, pero incluye una implementación binaria de diff escrita en C. Mercurial fue escrito originalmente para funcionar sobre GNU/Linux pero ha sido adaptado para Windows, Mac OS X y la mayoría de otros sistemas tipo Unix.

Mercurial funciona bajo líneas de comandos, sus operaciones se invocan como opciones dadas a su programa motor, hg (hace referencia a símbolo químico del mercurio).

Dentro de las ventajas más importantes de Mercurial, podemos resaltar:

- Permite el desarrollo distribuido de un proyecto mediante a branches.
- Provee un control de versiones robusto que permite volver a cualquier revisión subida al repositorio.
- Permite documentar cada uno de los cambios realizados en el repositorio.
- Ofrece un sistema de fusión de versiones mediante el programa diff. [6]

Se han mencionado algunas herramientas para la construcción de proyectos y algunos sistemas de control de versiones, en el próximo ítem se hablara acerca de los servidores de integración continua.

4.5. SERVIDORES DE INTEGRACIÓN CONTINUA

El papel que desempeña un servidor de integración continua es vital en el desarrollo software pues permite la integración de código y la ejecución periódica de las pruebas que previamente han sido planteados por el analista o tester. En la actualidad se han implementado muchos servidores de integración continua Jenkins, Bamboo, TeamCity, Cruise Control, Go Continuous Delivery,etc. Se centra la atención en los tres primeros y se profundizará en Jenkins que ha sido el seleccionado para la implementación del entorno de integración en el caso de estudio.

4.5.1. TeamCity

Es un servidor de integración continua y un administrador de software creado por JetBrains en el 2006. Es un software comercial y está bajo licencia de propiedad. Sin embargo tiene disponible una licencia “Premium” para un número de configuraciones de builds. Los proyectos *open source* pueden solicitar una licencia gratis.

Es integrable con entornos de desarrollo como eclipse, visual studio, entre otros y con plataformas compatibles como Java, Ruby, etc. Además soporta diversos sistemas de control de versiones como Subversion, CVS, Git, Mercurial, Microsoft Visual SourceSafe, entre otros.

TeamCity permite una gestión de usuario flexible, es decir, facilita la inclusión de funciones de usuario, la clasificación de los usuarios en grupos, diferentes formas de autenticación y un registro con todas las acciones de los usuarios para la transparencia de toda a la actividad en el servidor.[7]

4.5.2. Bamboo

Bamboo es la solución software que ofrece la empresa Atlassian como un servidor de integración continua (CI) que se integra plenamente con Jira y que se paga en función del número de agentes de *build* que necesitas. Incluso hay una versión de prueba gratuita.
Cualquier cambio en el código se construye y prueba de forma automática con la posibilidad de establecer dependencias y procesos de construcción complejos. Define procesos de construcción manuales en caso de ser necesario, parametrizar los builds para simplificar la construcción sobre diferentes versiones de código y el despliegue en distintos entornos.

Bamboo se integra con las siguientes herramientas:

- Versiones de código: Subversion, Git, Mercurial, CVS y Perforce.
- Tecnologías y lenguajes: Maven (1, 2, 3), PHP, Ant, .Net, Make y scripts
- Entornos de pruebas xUnit: jUnit, Selenium, PHPUnit y Nunit
- Análisis de calidad: Sonar o Clover.

4.5.3. Jenkins

Jenkins es un servidor de integración continua *open-source* ideada por Kohsuke Kawaguchi. Está escrito en Java y, permite planificar y realizar multitud de tareas simplificando los procesos involucrados en el ciclo de vida de un proyecto.

Desde Jenkins se puede indicar que se lancen métricas de calidad y visualizar los resultados dentro de la misma herramienta. También se puede ver el resultado de los tests, generar y visualizar la documentación del proyecto o incluso pasar una versión estable del software al entorno de QA para ser probado, a preproducción o producción.[9]

Por ejemplo, se puede programar una tarea en la que se compruebe el repositorio de control de versiones cada cierto tiempo, y cuando un desarrollador quiera subir su código al control de versiones, este se compila y se ejecutan las pruebas. Si el resultado no es el esperado o hay algún error, Jenkins notificará al desarrollador, al equipo de QA, por email o cualquier otro medio, para que lo solucione. Si el build es correcto, podremos indicar a Jenkins que intente integrar el código y subirlo al repositorio de control de versiones.
Además Jenkins se ejecuta dentro de un contenedor de servlets, como Apache Tomcat; y también soporta herramientas de control de versiones como CVS, Subversion, Git, Mercurial, Perforce y Clearcase, mientras puede ejecutar proyectos basados en Apache Ant y Apache Maven, así como scripts de shell y programas batch de Windows.

Algunas de las características importantes de Jenkins son:

- Comprobación cada cierto periodo de tiempo si se ha realizado algún *commit* en el repositorio de control de versiones (GIT), y en caso de ser así, compilar el código y ejecutar las pruebas para testearlo.
- Notificación de errores que se hayan detectado tras las ejecuciones de pruebas, por ejemplo vía mail, twitter, chat, etc.
- Generación y publicación de binarios.

- Ejecución de métricas de calidad y visualización los resultados.
- Generación de documentación asociada a un proyecto. [8]

4.6. Docker

Actualmente los equipos de desarrollo después de ejecutar las tareas de compilación, empaquetado, pruebas, deben tomar decisiones para colocar en producción la aplicación. En este punto del desarrollo tienen en cuenta características que posibilitan la ejecución de la aplicación independiente de la máquina origen que se utilice (máquina virtual, servidor, computador) y que sean implementadas con el mayor ahorro de recursos de máquina posible. Se han implementado varias herramientas que permiten el despliegue de las aplicaciones y alguno de los beneficios mencionados anteriormente pero es Docker la herramienta que satisface estas necesidades con mayor precisión.

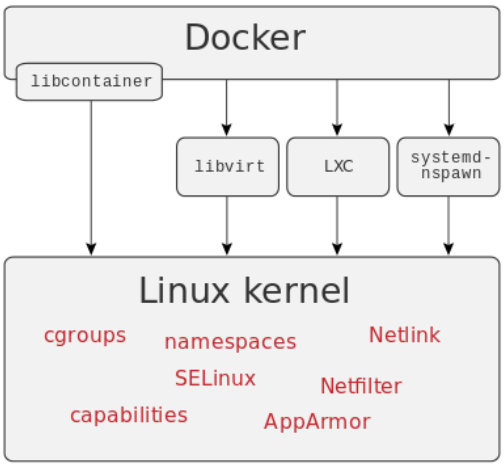
Docker es una herramienta open-source que fue liberada en marzo del 2013 dentro de la empresa docCloud por Simon Hykes, cuyo objetivo principal es automatizar el despliegue de aplicaciones por medio de contenedores que utilizan una "virtualización ligera" permitiendo así el despliegue de aplicaciones en cualquier sistema que disponga de esta tecnología.

Los contenedores docker son plataformas aisladas donde se puede alojar una aplicación que contiene todas las herramientas necesarias para su despliegue y se pueden crear a partir de una imagen que se encuentra en un repositorio público, privado o en nuestra máquina local.
Docker es un proyecto que ha tomado ideas de otras herramientas, agrupando todas en una sola idea. Desde hace años Linux tiene muchas herramientas que están comprendidas en Docker, pero por separado. Los desarrolladores de la empresa docCloud liderados por Simón Hykes se ingeniaron la forma de agrupar todas esas herramientas interactuando directamente con el kernel de linux.

Docker primeramente fue pensado en Linux pero por el gran auge que ha tomado ha sido modificado para ser adaptado a otros sistemas operativos.

La siguiente figura muestra la interacción de Docker con el kernel de linux. Además podemos observar las librerías y extensiones de docker que permiten que utilice los recursos necesarios del kernel de Linux.

Figura 7.Estructura interna de Docker



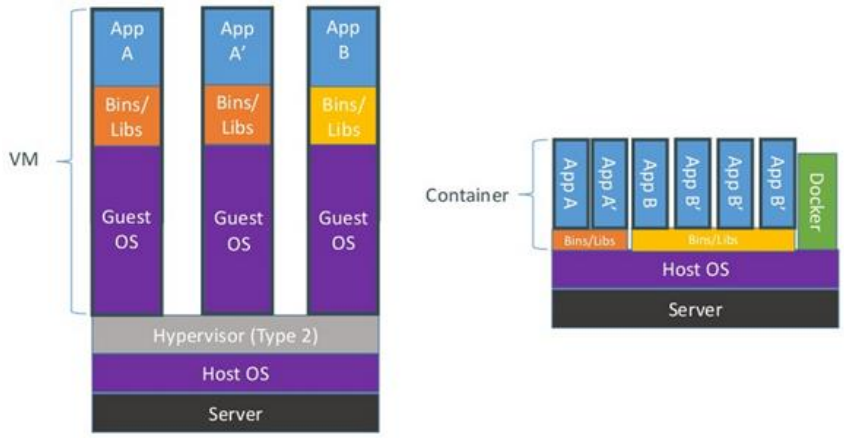
Fuente: https://blog.docker.com/2014/03/docker-0-9-introducing-execution-drivers-and-libcontainer/

Al utilizar virtualización Docker funciona de una forma similar a las máquinas virtuales, en el siguiente apartado se hablará de las diferencias y similitudes entre ellos.

4.6.1. Docker Vs Máquinas Virtuales

La principal diferencia radica en que las máquinas virtuales para ejecutar una aplicación necesita contener todo el sistema operativo mientras que un contenedor Docker al interactuar con el sistema operativo directamente utiliza solo los recursos necesarios del sistema operativo.

Figura 8. Docker vs Máquinas Virtuales



Fuente:https://www.upguard.com/articles/getting-started-with-docker-part-1-of-2

En la figura se observan los componentes que se usan para ejecutar una aplicación en máquina virtual, y en Docker. Mientras que Docker utiliza los recursos necesarios del kernel por medio de binarios y librerías, la máquina virtual utiliza por medio de un Hypervisor todo el sistema operativo minimizando el uso por parte de la aplicación de recursos de máquina como cpu, ram, disco y procesador.

Las principales ventajas de Docker con respecto a las máquinas virtuales son:

- Los contenedores son más livianos.
- Menor CPU,RAM o espacio de almacenamiento requerido.
- Más contenedores por máquina que las máquinas virtuales.
- La portabilidad de las aplicaciones, permitiendo desplegar la aplicación desde cualquier entorno sin necesidad de instalar las herramientas necesarias para ello.

En general, el uso de Docker al usar menos recursos hace posible un mejor rendimiento de aplicaciones por servidor que las máquinas virtuales.

5. ESTADO DEL ARTE

Se ha vuelto imprescindible en el desarrollo de los proyectos software disponer de una batería de test automatizado, algunas prácticas de desarrollo y metodologías ágiles permiten satisfacer esta necesidad, entre estas la integración continua. Debido a que las herramientas usadas en esta práctica de extreme *programming* están siendo actualizadas y muchos desarrolladores y contribuidores están implementando nuevas herramientas que propician un ambiente tecnológico mejorado de este entorno. Cuando el desarrollo de pruebas es manual, las empresas de desarrollo integran el código justo antes de colocar en producción el proyecto, contratan a una persona dedicada en la inspección del código y en el mejor de los casos esta tarea demoraba alrededor de un día, solo en la corrección de errores de compilación en la integración del código.

Hoy por hoy, se han elaborado varios artículos con respecto al funcionamiento de integración continua, la mayoría coinciden en la evolución y la acogida que ha tenido el entorno tanto en sus herramientas como en la asociación de varias metodologías ágiles en esta práctica, como son: la planificación, programación en parejas, refactorización y el desarrollo dirigido por pruebas. [10]
Por ello se le ha dado el título de "una buena práctica de programación" y es utilizado en varias empresas como Dell, Sony, proyectos de código abierto como Github, Ebay, Facebook, etc. [12]

La integración continua permite que un equipo de desarrollo le sea posible tener una retroalimentación instantánea, una generación de documentos automática, confianza al aplicar cambios, una reducción de errores, y demás. En este contexto, en este trabajo se estudia y se implementa un entorno de integración continua en un caso de estudio, mostrando las ventajas de esta práctica que está revolucionando el mundo del desarrollo software.[11][13]

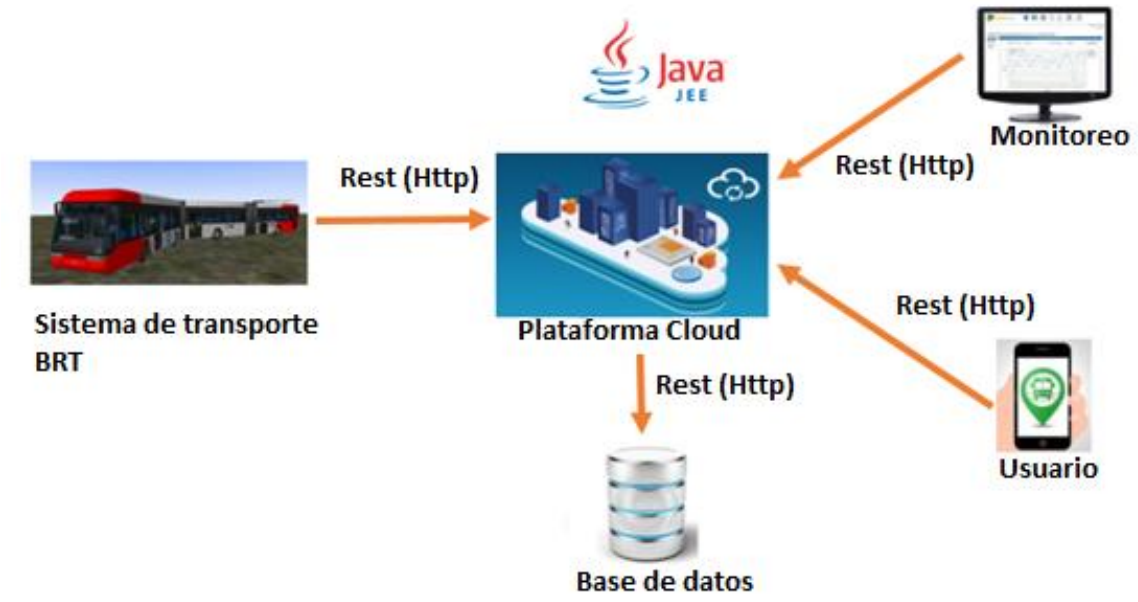
6. DESARROLLO DEL PROYECTO

En este capítulo se describe las características de un caso de estudio que se probara en un proceso de integración continua, la selección de las herramientas para la implementación del entorno de integración continua, la implementación del proceso de integración, así como la elaboración y ejecución de las pruebas a automatizar y los resultados obtenidos.

6.1. Caso de estudio

Este proyecto está asociado a un macroproyecto, el cual pretende solucionar entre otras problemáticas la falta de información en un sistema de transporte BRT (por sus siglas en incles Bus Rapid Transport)

Figura 9. Estructura de Macroproyecto



A los buses del sistema de transporte BRT se les implemento una plataforma embebida, la cual recolecta los datos y los envía a la plataforma cloud. La plataforma cloud por medio de servicios tiene a disposición los datos del bus a terceros, como son departamento de monitoreo, un usuario final (pasajero).

Del macroproyecto se seleccionó la plataforma cloud porque tiene características que hacen posible la realización de actividades en un entorno de integración continua, como son:

Funcionalidades bien definidas que permiten realizar pruebas funcionales como son las pruebas unitarias.

La plataforma tiene propiedades no funcionales que permiten realizar pruebas no funcionales.

La plataforma cloud implemento servicios Rest, permitiendo la realización de pruebas a los servicios Rest y transversalmente pruebas de integración.

La plataforma cloud utiliza un framework Java Entreprise Edition (JEE) permitiendo realizar un análisis en la generación de artefactos.

6.2. Selección de las herramientas para la implementación del entorno de integración continua

Las herramientas a implementar en el entorno de integración continua del caso de estudio fueron seleccionadas en base a criterios como coste, funcionalidad, usabilidad, madurez, documentación existente, accesibilidad, automatización, entre otros.

En la siguiente tabla se muestra la selección de las principales herramientas que constituyen la alternativa de integración continua y su motivo de elección.

Tabla 1. Selección de herramientas principales del entorno de integración continua.

Tipo de Herramienta	Selección de herramienta	Justificación
Herramienta de construcción y gestión de proyectos	Maven	Maven fue seleccionada para la construcción de las pruebas, debido a que el proyecto del caso de estudio utiliza esta herramienta de construcción.
Sistema de Control de Versiones	Git	Se ha seleccionado Git como el sistema de control de versiones debido a la flexibilidad que brinda a los desarrolladores al ser un sistema de control de versiones distribuido, por el auge que ha tomado y en base a ello su gran acogida en entornos de integración continua. Además se utiliza la plataforma de hosting github.
Servidor de Integración continua	Jenkins	Se escogió a Jenkins como el servidor de integración continua por las ventajas que ofrece frente a otros servidores de integración, entre las cuales es posible destacar, que dispone de una gran cantidad de plugins, soporta y se integra con varias tecnologías con un bajo coste, y es fácil de usar.

6.3. Selección de las herramientas para realizar las pruebas de software.

En el apartado de las pruebas de software se menciona las pruebas que se llevarán a cabo en esta alternativa. En esta sección se habla de la selección de las herramientas para la elaboración y ejecución de estas pruebas.

6.3.1. Selección de una herramienta para realizar las pruebas unitarias

El proyecto de estudio fue implementado en lenguaje java, por lo cual para realizar las pruebas unitarias se utilizara jUnit. Básicamente Junit es un framework implementado por Erich Gamma y Kent Beck que contiene un conjunto de librerías

escritas en java para el despliegue de las pruebas unitarias, se ha convertido en una herramienta fundamental en entornos de integración continua porque permite la automatización de pruebas unitarias.

6.3.2. Selección de un framework para realizar las pruebas de carga y estrés.

Existen varias herramientas open-source como (Jmeter, JWebUnit) y de pago como (LoadRunner) para realizar las pruebas de carga y estrés de una aplicación web, pero en la actualidad Jmeter se destaca entre todas por su versatilidad, estabilidad, facilidad de uso entre otras ventajas. Apache JMeter es una herramienta open-source que permite realizar pruebas de rendimiento y pruebas de carga / estrés además de otras funciones enfocadas a las propiedades no funcionales del software.

6.3.3. Selección de un framework para realizar las pruebas a los servicios rest del caso de estudio.

Para realizar las pruebas a los servicios web se utilizará el framework Rest Assured debido a que los servicios web implementados en el caso de estudio son servicios rest, hubiese sido posible escoger a Jersey u otro framework que pruebe este tipo de servicios pero la estructura de Rest Assured permite con mayor flexibilidad implementar las pruebas.

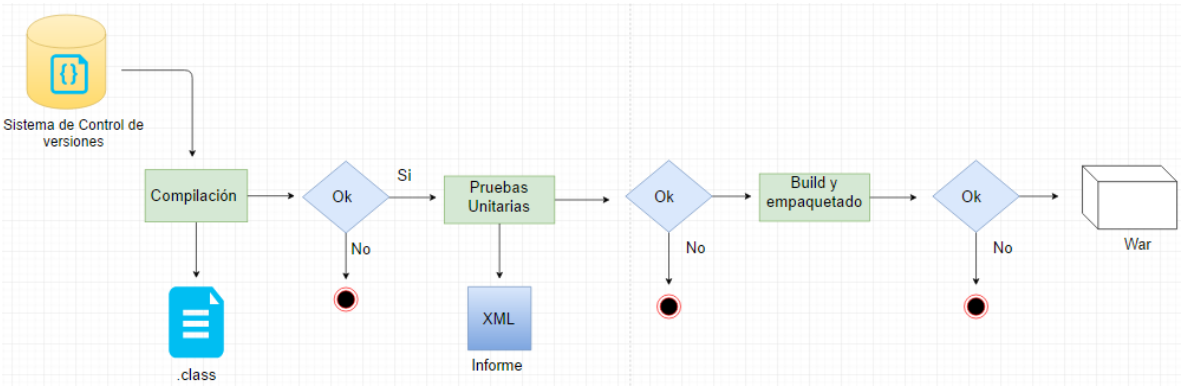
6.4. Proceso de Integración continua en caso de estudio

El proceso de integración en el caso de estudio se llevó a cabo por medio de la realización de las siguientes fases:

El proceso comienza cuando Jenkins ejecuta las pruebas unitarias y encuentra uncambio en el flujo de trabajo, es decir los desarrolladores han realizado unaactualización (commit) en el sistema de control de versiones, en este momento sedispara la fase de construcción y ejecución de pruebas unitarias, siendo posible laobtención de alguno de los siguientes resultados:

1. Se produce algún fallo en las pruebas unitarias y no se construye el proyecto notificando a las partes interesadas de que ha fallado.
2. Las pruebas son correctas, se construye el proyecto y se despliega el war, notificando a las partes interesadas de que se ha acabado la fase con éxito.

Figura 10. Primera fase (Workflow)



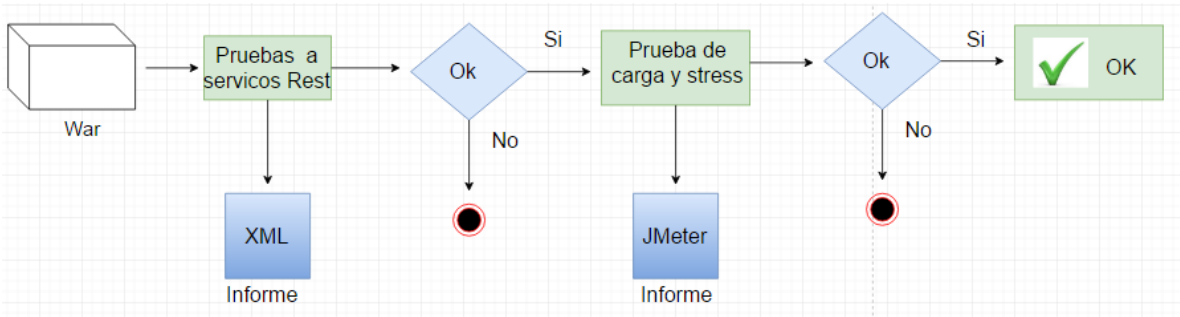
La siguiente fase será la ejecución de las pruebas de integración, para este entorno se implementaron pruebas a los servicios Rest, debido a que permiten verificar si la integración del proyecto es correcta, esta fase puede devolver dos resultados:

1. Se produce algún fallo en las pruebas, notificando a las partes interesadas.
2. Todas las pruebas se ejecutan con éxito, se notifica a las partes interesadas de los resultados de las pruebas y de que es posible pasar a la siguiente fase.

En la próxima fase se ejecutarán las pruebas no funcionales, se podrán obtener dos resultados:

1. Se produce algún fallo en las pruebas, notificando a las partes interesadas.
2. Todas las pruebas se ejecutan con éxito, se notifica a las partes interesadas de los resultados de las pruebas.

Figura 11. Workflow segunda y tercera fase



Por último, se observan los resultados obtenidos, se valora si el software está listo para su puesta en producción, de ser así se realizará el despliegue final de la aplicación.

6.4.1. Validación del proceso de integración continua en caso de estudio

Para la validación del proceso de integración continua en el proyecto de estudio, se realizaron las siguientes actividades, las cuales serán profundizadas en secciones posteriores.

- **Ejecución del proyecto en diferentes entornos**

Se ejecutan pruebas unitarias en diferentes ambientes y se examinan los resultados.

Figura 12. Ejecución de prueba en máquina local

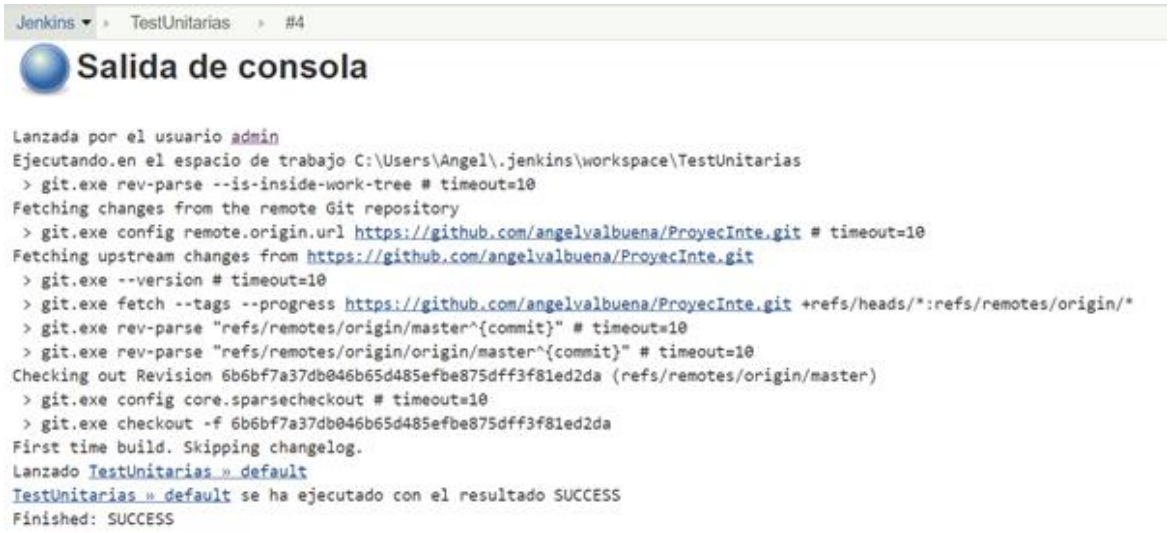
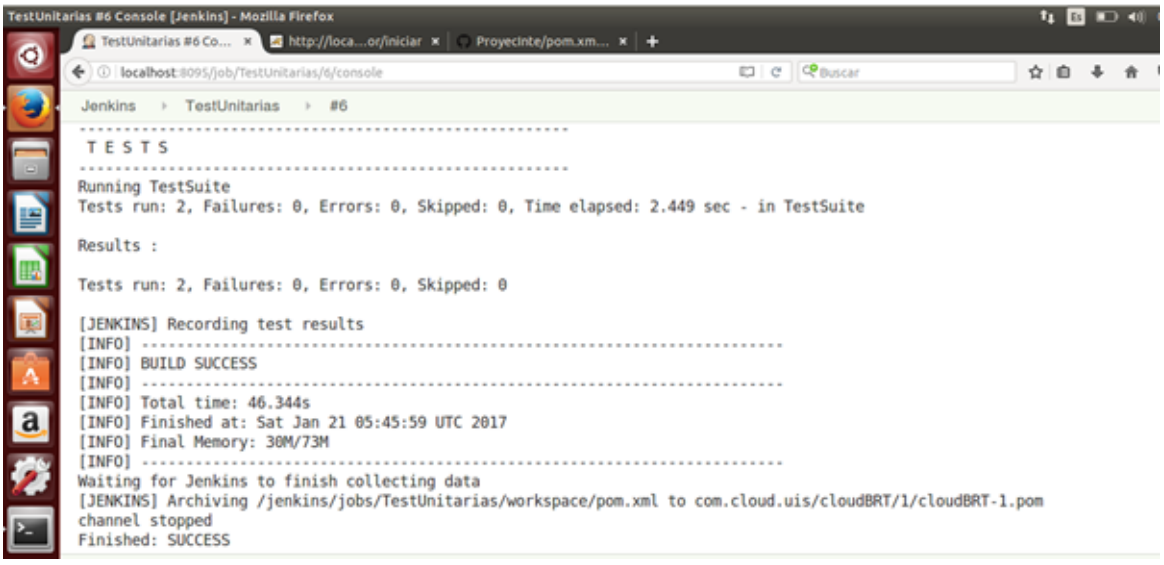


Figura 13. Ejecución de prueba en máquina virtual.



Se observa que la prueba funciona correctamente en ambos ambientes.

- **Examinar los tiempos de respuesta de las pruebas**

Por medio de la ejecución de las pruebas no funcionales se examinan los tiempos de respuesta de la prueba para verificar la rapidez en la ejecución de las mismas.

Figura 14. Ejecución de prueba de carga y stress en máquina local.

En la imagen se muestra que es posible tener un análisis detallado de los tiempos de respuesta de las pruebas a partir de su ejecución.

- **Verificar la producción de los artefactos a desplegar**

Se ejecutan pruebas unitarias y se verifica la producción de los artefactos que se deben desplegar.

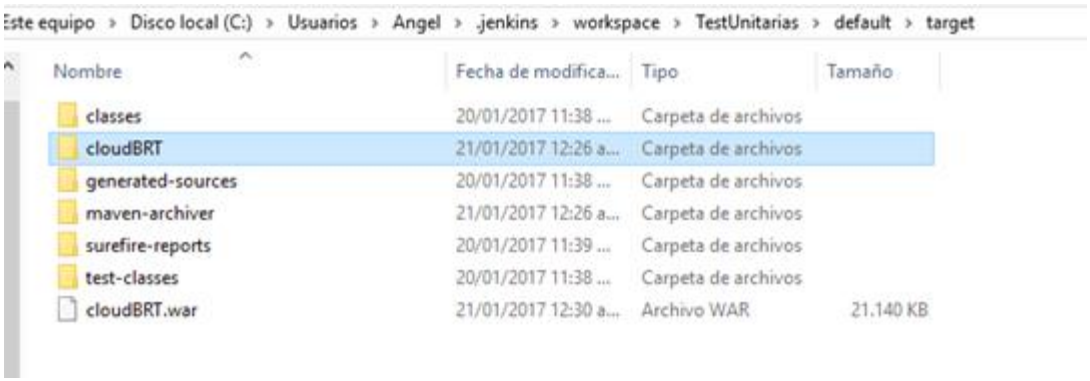
Figura 15. Ejecución de prueba unitaria en máquina local.

En la imagen se observa que los artefactos se han desplegado correctamente.

- **Verificar el correcto despliegue**

Después de ejecutar las pruebas unitarias, se examina en la carpeta target del proyecto el correcto despliegue del war.

Figura 16. Comprobación de despliegue war en carpeta target.



En la imagen se puede verificar que se ha desplegado correctamente el war.

- **Verificar la integración de las herramientas a utilizar**

Se ejecutan pruebas que requieren una integración correcta de las herramientas.

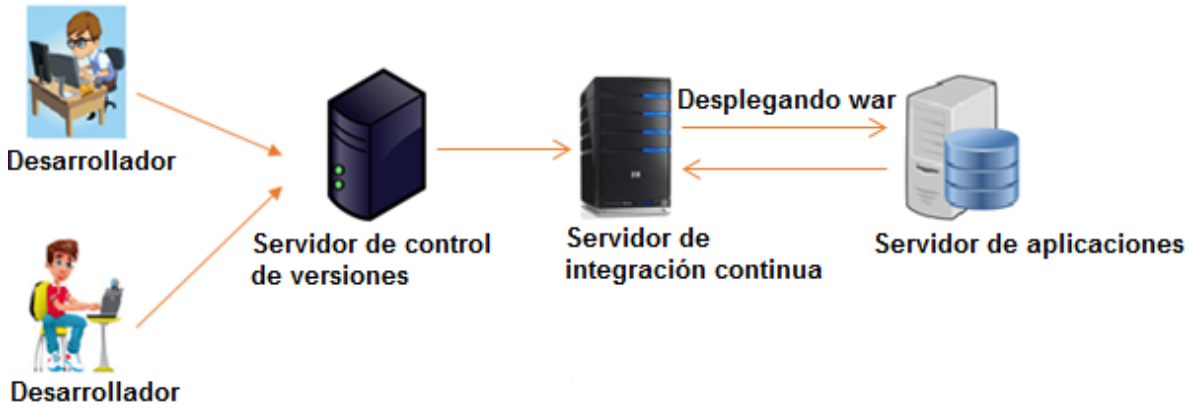
Figura 17. Ejecución de prueba a servicio web en máquina local.

En la imagen se observa que se han ejecutado las pruebas a servicios web correctamente, por lo que se puede inferir la correcta integración de las herramientas.

6.4.2. Diseño lógico del entorno de integración Continua

Para el diseño de esta alternativa se tiene en cuenta la integración de las herramientas previamente seleccionadas.

Figura 18. Diseño lógico del proceso de integración continua

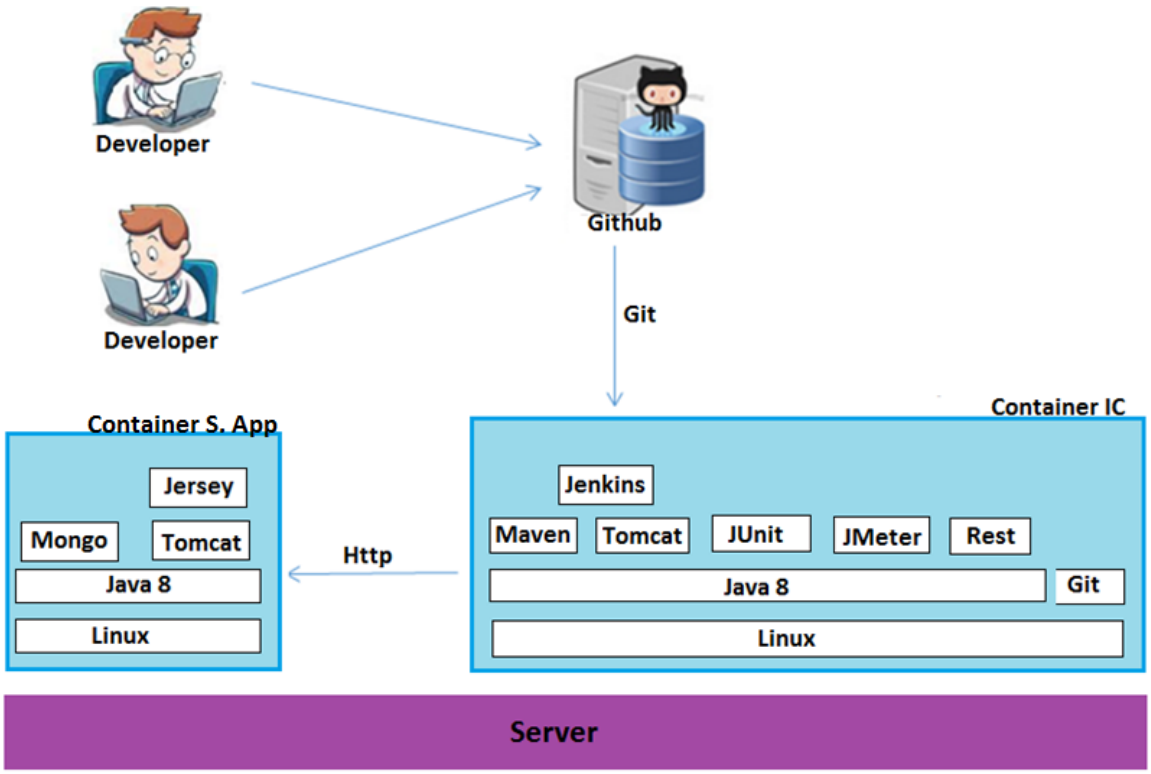


El proceso de integración continua comienza cuando Jenkins ejecuta las pruebas unitarias y encuentra un cambio en el flujo de trabajo, es decir los desarrolladores han realizado una actualización (commit) en el sistema de control de versiones, en esta fase las pruebas han sido previamente implementadas y están alojadas en Github. Mediante la configuración de las tareas creadas en jenkins, se puede establecer la integración con Github, y se podrán ejecutar las pruebas. Estas pueden ser ejecutadas de forma manual o configurando Jenkins para que sean ejecutadas periódicamente, posteriormente Jenkins muestra los resultados y los envía al tester. Si funcionan correctamente las pruebas unitarias, se despliega el war en Tomcat y se llevarán a cabo las pruebas de integración y posteriormente las pruebas que el tester haya implementado.

6.4.3. Proceso de diseño en arquitectura física

La alternativa de integración para este proyecto se implementará en un servidor facilitado por la Universidad Industrial de Santander. En la siguiente imagen se muestra el prototipo de diseño en esta infraestructura.

Figura 19. Diseño de arquitectura física del entorno de integración continua.



En la figura se puede observar que se utilizaron dos contenedores implementados en Docker, uno para Jenkins y otro para Tomcat. Se usa Docker porque se adapta a las necesidades del entorno, puesto que los recursos actuales para la elaboración de este proyecto no son suficientes para implementar una arquitectura con máquinas reales, y Docker por medio de virtualización ligera interactuando directamente con el kernel permite ahorro en los recursos. Además en esta alternativa de integración varios componentes están constantemente interactuando y es necesario un ambiente limpio sin contaminación. Docker, por medio de contenedores permite tener un ambiente aislado con estas condiciones en cada etapa del entorno adaptándose perfectamente al mismo.

6.4.4. Determinación de criterios de evaluación en la infraestructura de estudio

Fue necesario analizar las diferentes etapas de la infraestructura en estudio para establecer los criterios de evaluación (en el apartado de fases del proceso se describe detalladamente dicho análisis), en base a este análisis se establecieron los siguientes criterios:

- **Verificar las propiedades funcionales del caso de estudio**

Se ejecutan las pruebas unitarias en Jenkins para verificar propiedades funcionales como métodos y clases del proyecto de estudio.

- **Verificar las propiedades no funcionales del proyecto de estudio**

Se ejecutan pruebas de carga y stress con Jmeter para verificar propiedades no funcionales como el performance y la escalabilidad del proyecto

- **Verificar las propiedades de integración del código del proyecto de estudio**

Se ejecutan las pruebas de los servicios Rest para verificar si la integración del proyecto es correcta.

6.5. Implementación de la alternativa del entorno de Integración continua

Primeramente se realiza la codificación de las pruebas unitarias, pruebas de servicios rest y pruebas de carga y stress. Las clases y servicios seleccionados en esta fase están descritos en el plan de pruebas (El código de las pruebas y el plan de pruebas se anexan al final de este documento).

Para la siguiente parte del proceso se aloja el código de las pruebas en diferentes proyectos mavenizados en Git, según el tipo de prueba y se configura Jenkins para que interactúe con Git, permitiendo el acceso a los proyectos de prueba. En el próximo apartado se explicará la creación de tareas en Jenkins y como configurar Jenkins para que interactúe con Git.

6.5.1. Integración de Jenkins y Git

Antes de mostrar la configuración de Jenkins para que se integre con Git, es necesario establecer las variables de entorno para Java y Maven, para su configuración se deben seguir los siguientes pasos:

- Ingresar al panel principal de Jenkins.
- Click en la opción “administración del sistema”.
- Click “configurar herramientas”.

Estableciendo así las variables de entorno.

Para la integración de Jenkins y Git se debe descargar el plugin “Git Plugin”, en este paso Jenkins facilita su descarga porque tiene una opción donde se podrán descargar e instalar

los plugins requeridos. En los siguientes screen se muestra la creación de una tarea de construcción en Jenkins con su respectiva configuración.

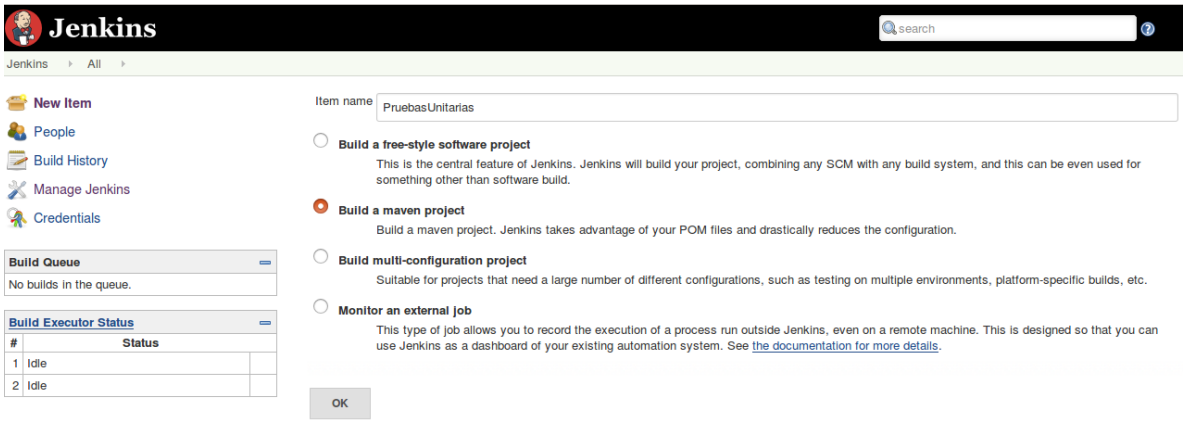
6.5.2. Creando tarea en Jenkins

Para la creación de una tarea en Jenkins se deben seguir los siguientes pasos:

- Ingresar al panel de control de Jenkins.
- Click en “nueva tarea”, aquí escogemos el tipo de proyecto que se va a crear. Para el caso de estudio será proyectos Maven.

En la siguiente figura se muestra la creación de una tarea.

Figura 20. Construcción de tarea en jenkins.

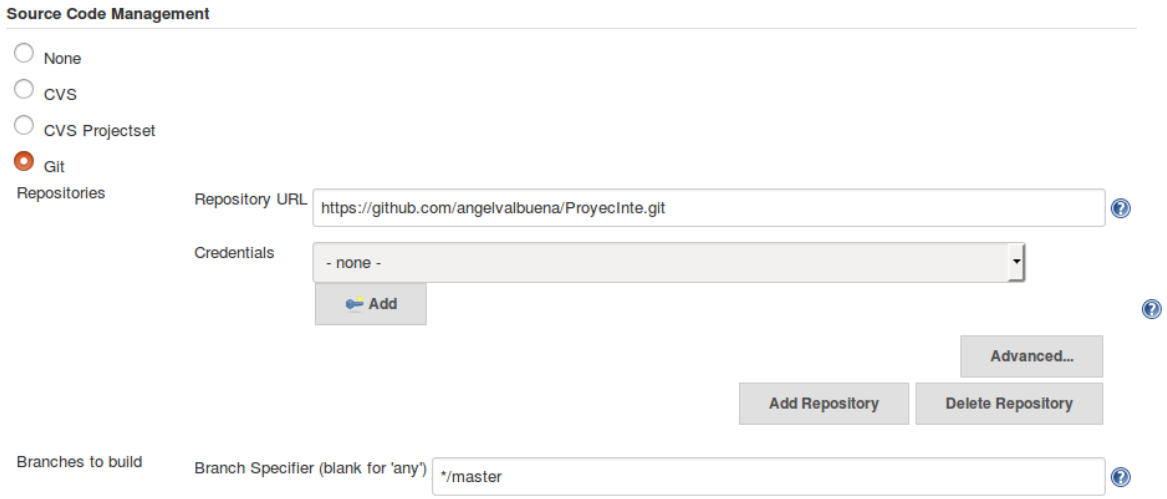


Para configurar una tarea se deben realizar los siguientes pasos:

- Ingresar a la tarea dando click sobre su nombre.
- Click en la opción “configurar”.

Se configura el proyecto según las especificaciones del entorno que se esté empleando, para este caso de estudio, Jenkins se comunicara con Git haciendo push, por lo tanto se selecciona Git y se copia la url del repositorio donde está alojada la prueba.

Figura 21. Configuración de Jenkins con Git.



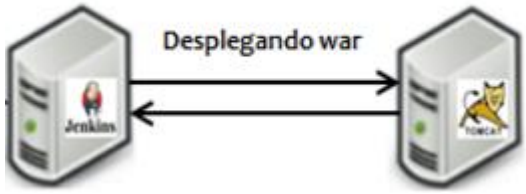
Adicionalmente Jenkins interactúa con proyectos maven y en esta misma ventana de configuración podremos especificar el goal a ejecutar en la prueba.
Con estos pasos estará configurado Jenkins para que interactúe con Git y se podrán ejecutar las pruebas para corroborar la correcta configuración.

6.5.3. Interacción Jenkins-Tomcat

Se escogió Apache Tomcat como servidor web, debido a que se adapta perfectamente a los recursos de este proyecto al ser open-source, y a las especificaciones de la arquitectura física de los servicios web siendo un servidor multiplataforma, además del buen posicionamiento que tiene en el mercado.

La siguiente imagen muestra la interacción de Jenkins con Tomcat.

Figura 22. Interacción Jenkins-Tomcat



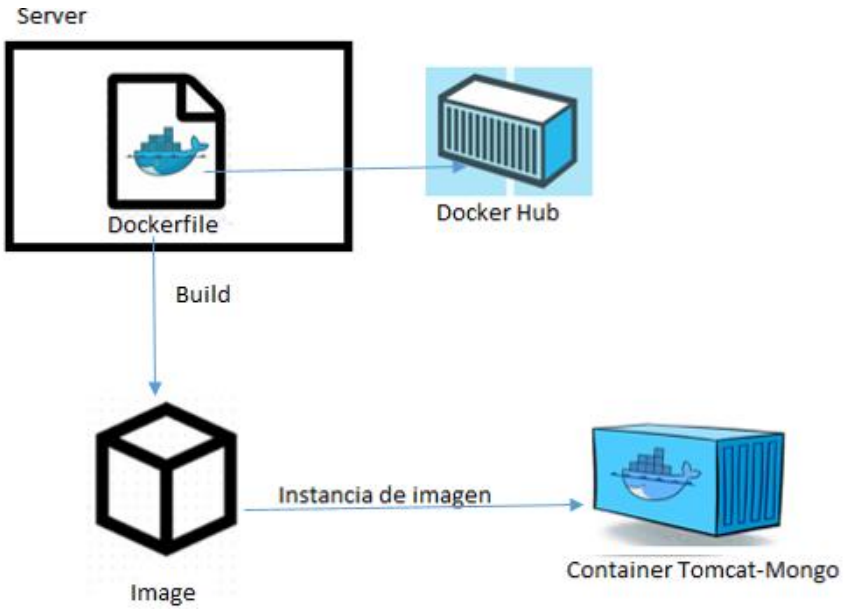
En la imagen, se observa que el servidor de integración continua Jenkins interactúa con el servidor web apache Tomcat. Después de que Jenkins da el visto bueno a las pruebas unitarias, construye el código para posteriormente desplegar el war del proyecto, dejando el servidor web apache listo para las pruebas que se realizarán a los servicios rest y las pruebas de carga y stress por medio de la url del servicio.

6.6. Implementación del entorno de integración continua en la arquitectura física

En el apartado “proceso de diseño de arquitectura física” se explica la razón de utilizar docker, y de forma general el diseño de la arquitectura física. En esta sección se hablará sobre su implementación.

Primeramente se creó el contenedor de Tomcat-Mongo. La siguiente imagen muestra el proceso de generar el contenedor de Tomcat-Mongo.

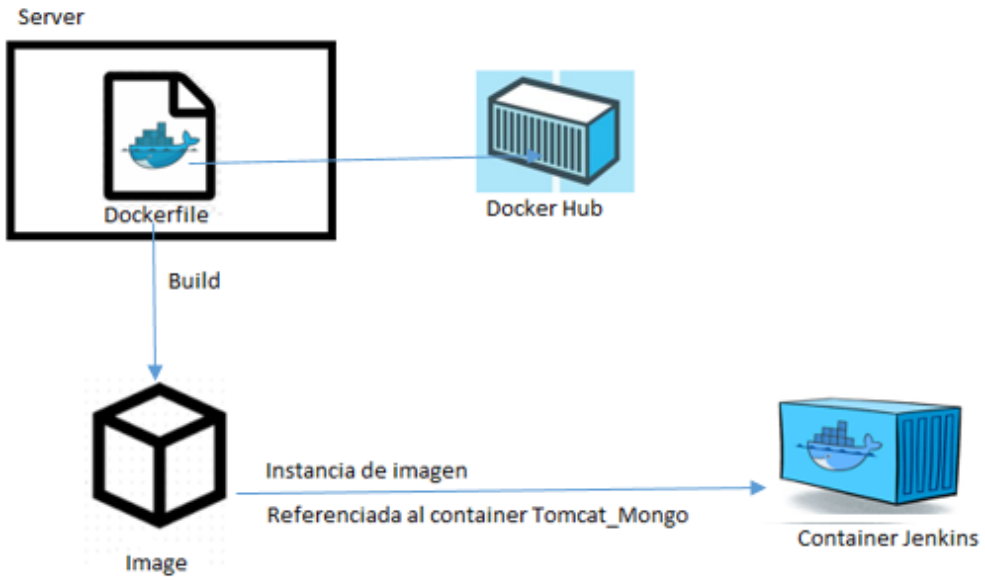
Figura 23. Proceso para generar contenedor Tomcat-Mongo



En la figura se observa, que en el servidor está alojado el Dockerfilepara construir la imagen(La url del repositorio esta anexado al final del documento). Un Dockerfile es un fichero en el cual se insertan las instrucciones de las herramientas de la imagen que se va a crear.
En la configuración del Dockerfile se especifica una imagen base, para lo cual es necesario que Docker descargue la imagen base y las imágenes requeridas en el Dockerfile utilizando el repositorio de imágenes de Docker llamado Docker Hub, posteriormente se construye la imagen haciendo un build y se instancia la imagen para la creacióndel contenedor.

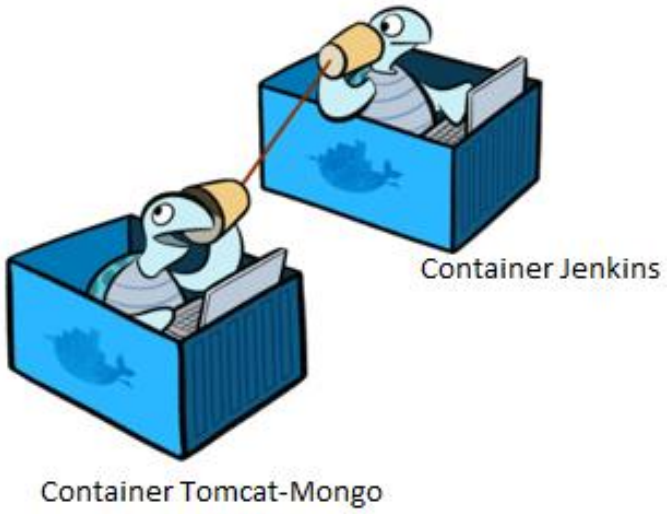
Una vez creado el contenedor Tomcat-Mongo se procede a crear el contenedor Jenkins, en esta etapa del proceso es necesario tener en cuenta la configuración de red entre los contenedores, debido a que es necesaria la comunicación entre estos para el correcto despliegue del war y para alojar las pruebas ejecutadas en la segunda y tercera fase. Docker en su documentación para plataforma de multicontenedores especifica el uso de link, link es una característica de Docker que permite configurar la red entre contenedores por medio de la referencia de nombres. Esta ha sido la configuración de red utilizada en este entorno.

Figura 24. Proceso para generar contenedor Jenkins



En la figura se observa que la creación del contenedor de Jenkins se hace de forma similar a la del contenedor Tomcat-Mongo, el servidor almacena el Dockerfile(La url del repositorio esta anexado al final del documento), este a su vez interactua con Docker Hub para descargar las imágenes especificadas en él y generar la imagen de Jenkins. Finalmente se instancia la imagen referenciando el nombre del contenedor de Tomcat-Mongo y se crea el contenedor de Jenkins en red con el contenedor de Tomcat-Mongo.
Para verificar la correcta configuración de red entre los contenedores, se ingresa al contenedor de Jenkins y se verifica los host que tiene en red.

Figura 25. Interacción entre contenedor Tomcat-Mongo y contenedor Jenkins



Fuente: http://blog.arunsriraman.com/2016/04/running-docker-containers-with-native.html

En la imagen se observa la interacción de los contenedores Jenkins y Tomcat-Mongo, lo que quiere decir que el entorno en el servidor está listo para la ejecución de las fases descritas en el proceso de integración continua.

6.7. Ejecución de las pruebas en el servidor

Para ejecutar las pruebas en el servidor se debe ingresar una url con la siguiente estructura ip_maquina-asignada:puerto_mapeado, y posteriormente ejecutar las pruebas. En los siguientes screen se muestra los resultados obtenidos después de la ejecución de las diferentes tipos de pruebas en el servidor.

6.7.1. Ejecución de las pruebas unitarias

Las pruebas unitarias tienen como principal objetivo probar una unidad de código en el proyecto (clase o método), en esta alternativa es imprescindible realizarlas debido a que permite un análisis en la parte funcional del caso de estudio y constituyen la primera fase en el momento de la ejecución de esta alternativa.

Para proceder a ejecutar las pruebas se crea y configura la tarea de construcción especificando cómo goal “test”. Se pueden ejecutar de dos formas las pruebas, manualmente o parametrizando la configuración para que se ejecute cada cierto tiempo.

Para ejecutar manualmente la prueba:

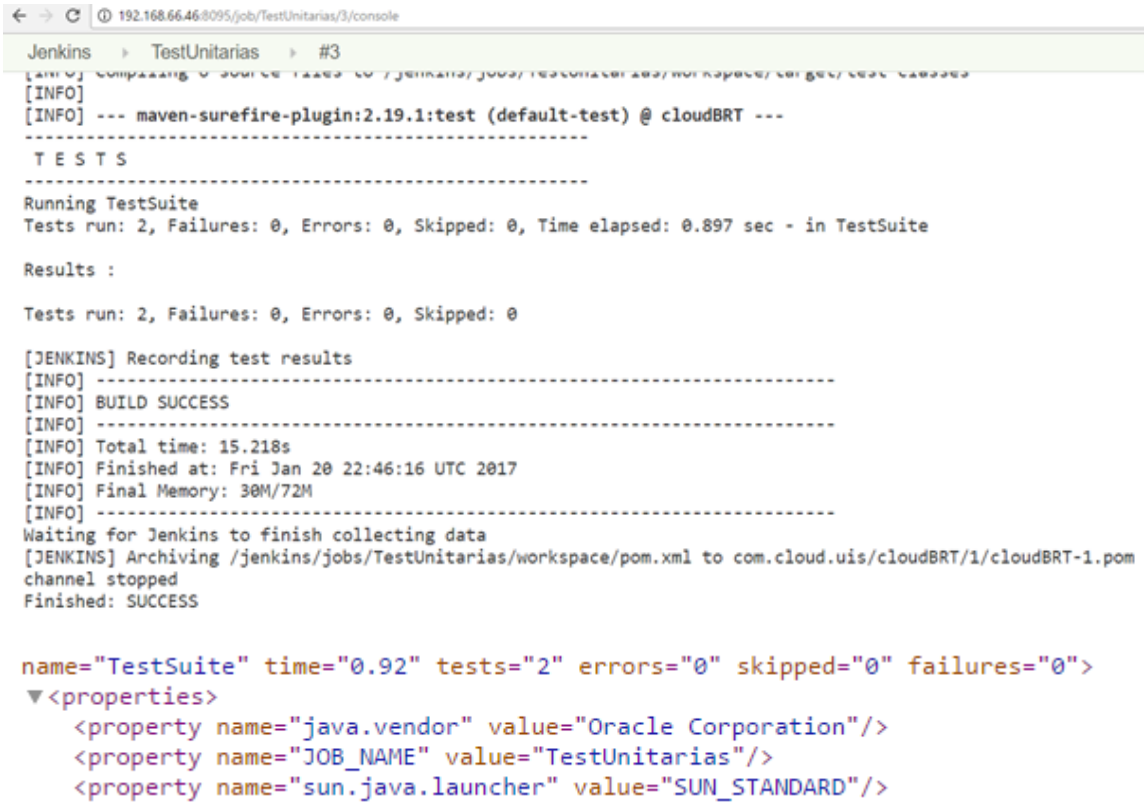
- Click en el “nombre de la tarea”

- Click en “construir ahora”.

En este caso se configuran las pruebas para que se ejecute periódicamente y de forma automatizable, configurando la tarea para que se ejecuten posteriormente los proyectos hijos.
La primera vez que se ejecutarón las pruebas unitarias en el servidor ocurrió un error en el despliegue del war, el problema radico en que no se estaba ejecutando correctamente el plugin de Maven-Tomcat, por lo cual fue necesario la creación de roles en el archivo de tomcatusers.xml.

La siguiente figura muestra la ejecución de las pruebas unitarias después de solucionar este error.

Figura 26. Ejecución de prueba unitaria en servidor.



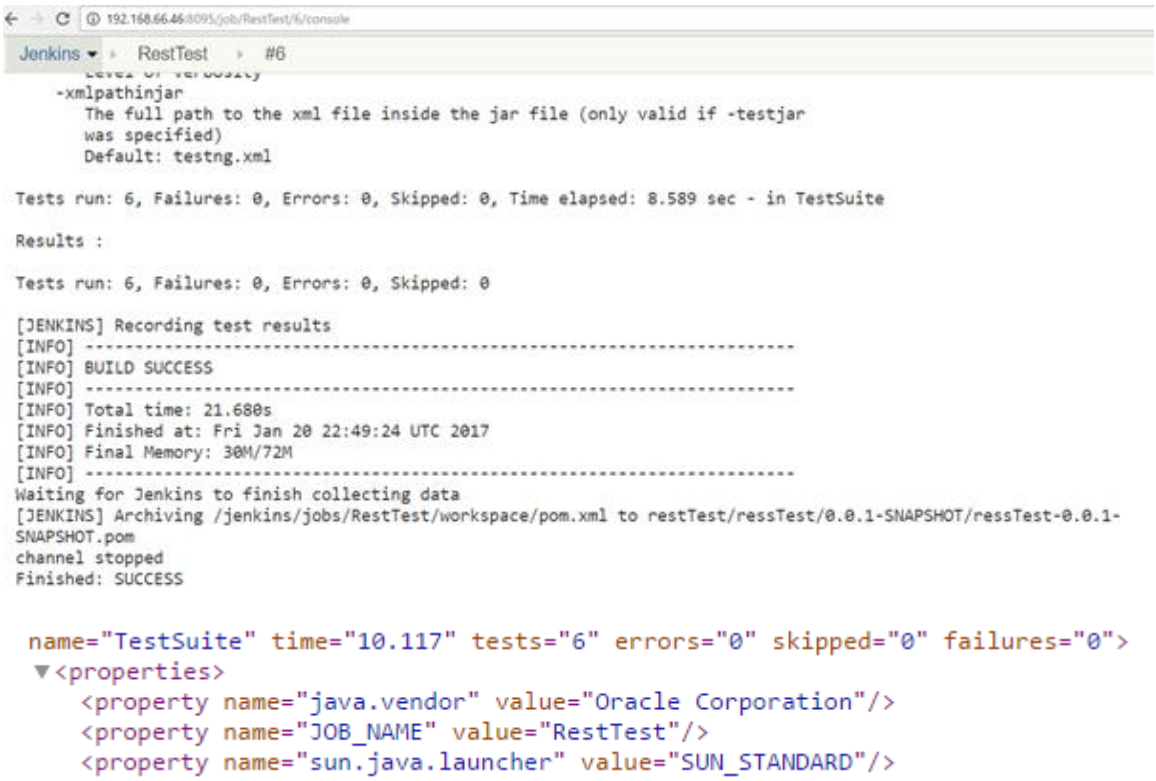
Se ejecuta el proyecto y se observa que se han realizado las pruebas unitarias y se ha construido el código del proyecto correctamente. Por lo tanto la funcionalidad de las clases probadas en el proyecto han sido bien implementadas, además el war ha sido desplegado correctamente en el contenedor de tomcat, haciendo posible pasar a la siguiente fase del proceso y llevar a cabo las pruebas de integración.

6.7.2. Ejecución de pruebas a los servicios Rest

El principal objetivo de implementar y ejecutar estas pruebas es verificar la correcta integración del proyecto en el momento en que se ha construido código integrando cambios en el flujo de trabajo, estas pruebas se ejecutan por medio de su url.
Para crear y configurar las tareas de construcción para los servicios rest se hace de forma similar a las pruebas unitarias, excepto por la configuración de la ejecución periódica, puesto que esa configuración solo se realiza al proyecto padre que en este caso son las pruebas unitarias. La tarea que contiene las pruebas de los servicios rest será un proyecto hijo de las pruebas unitarias, es decir, una vez termine la ejecución de las pruebas unitarias se ejecutarán automáticamente las pruebas de los servicios rest.

En la siguiente imagen se muestra un screen de la ejecución de la prueba a los servicios rest.

Figura 27. Ejecución de prueba a servicios Rest en servidor.

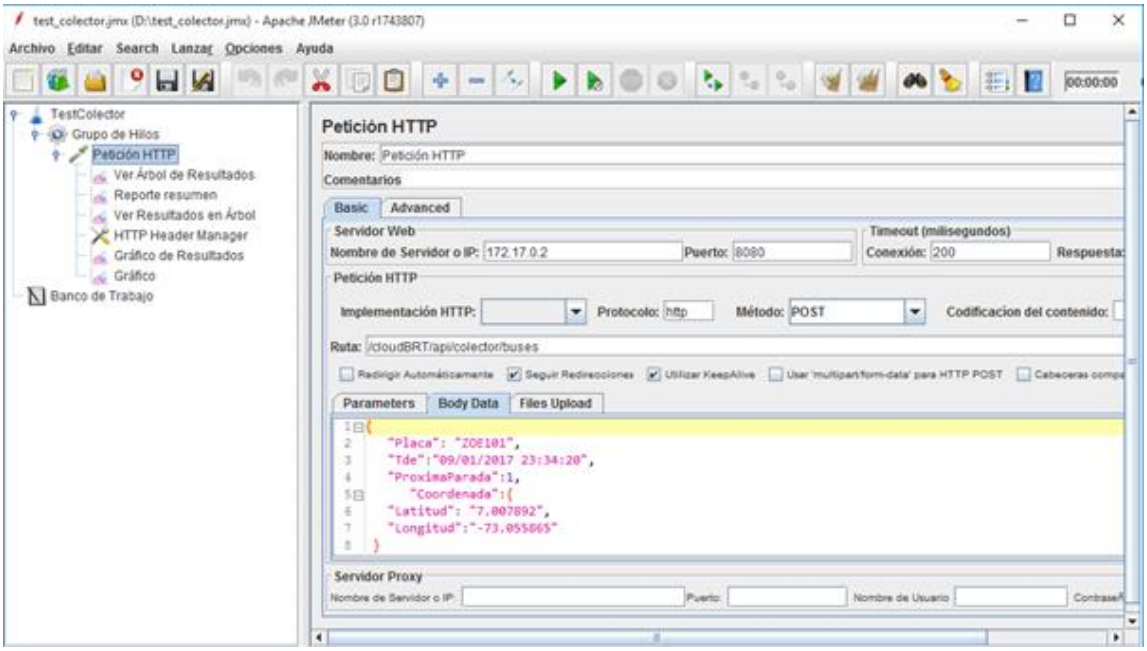


En la imagen se observa que los servicios seleccionados en el plan de prueba tuvieron un resultado positivo por lo cual se puede afirmar que estos servicios están correctamente implementados y se puede inferir la correcta integración del proyecto. Además se observa que las pruebas a los servicios web en el contexto del entorno de integración continua se están ejecutando correctamente.

6.7.3. Ejecución de las pruebas de carga y stress

Los test de carga y stress son pruebas no funcionales que han sido seleccionadas para ser ejecutadas en este entorno con el objetivo de mostrar propiedades no funcionales como la escalabilidad del proyecto, el performance, entre otras.
Para ejecutar estas pruebas se utilizó el framework Jmeter, por lo cual se debe crear previamente un archivo de extensión Jmx que contenga las pruebas con las especificaciones precisas que requiere el proyecto.

Figura 28. Especificaciones de prueba de carga y stress en framework Jmeter



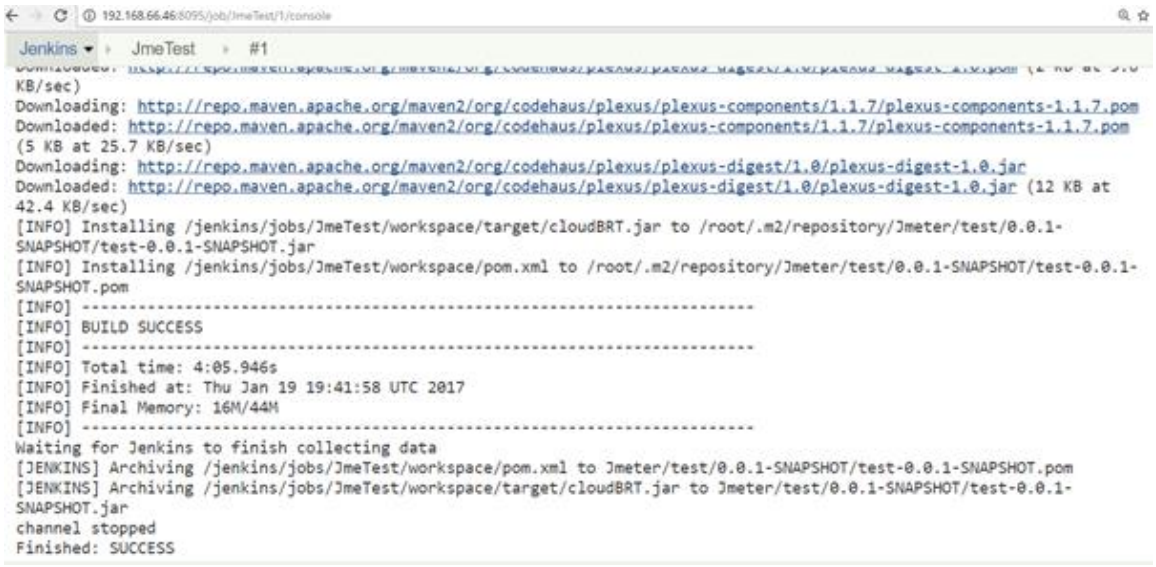
Además es necesario la configuración de un plugin que permite interacción entre jmeter y maven.

Figura 29. Plugin Maven-Jmeter

```
<plugin>
        <groupId>com.lazerycode.jmeter</groupId>
        <artifactId>jmeter-maven-plugin</artifactId>
        <version>1.10.1</version>
        <executions>
                <execution>
                        <id>jmeter-tests</id>
                        <phase>verify</phase>
                        <goals>
                                <goal>jmeter</goal>
                        </goals>
           </execution>
        </executions>
</plugin>
```

Se ejecuta la prueba de carga como las pruebas anteriores pero en el goal se coloca “verify”.

Figura 30. Ejecución de prueba de carga y stress en servidor




```
<testsuite tests="1" failures="0" name="Jmeter.test.AppTest" time="0.003" errors="0" skipped="0">
▼<properties>
   <property name="java.runtime.name" value="Java(TM) SE Runtime Environment"/>
   <property name="sun.boot.library.path" value="C:\Program Files\Java\jdk1.8.0_102\jre\bin"/>
   <property name="java.vm.version" value="25.102-b14"/>
   <property name="java.vm.vendor" value="Oracle Corporation"/>
   <property name="maven.multiModuleProjectDirectory" value="C:/Users/PC/Documents/GitHub/test"/>
```

El servicio seleccionado para esta prueba es un cliente de la plataforma implementada en el caso de estudio y ha sido probado con 1500 peticiones con especificaciones de tiempo de respuesta y de conexión de 200 ms.
En la imagen se observa que el servicio no pasó la prueba con las especificaciones antes mencionadas, pero el test de carga y stress se ha ejecutado de forma correcta. Permitiendo por medio de los datos obtenidos establecer un análisis delas propiedades no funcionales como la escalabilidad, el performance en este servicio y de forma general en el caso de estudio.

En general, a partir de los resultados obtenidos se observa que las pruebas se ejecutaron correctamente en el servidor, permitiendo verificar diferentes propiedades del proyecto de estudio, como son las propiedades funcionales, no funcionales y de integración.

7. CONCLUSIONES

Los resultados de las pruebas realizadas permiten concluir que:

- Se definió el diseño de una alternativa de integración que se adaptó a las especificaciones de un caso de estudio, haciendo posible examinar sus fases y entender por medio de su funcionamiento las características e interacciones de las herramientas que lo componen.

- Por medio del diseño del entorno y la ejecución de pruebas, se desarrollaron los componentes de forma eficiente y posteriormente en la implementación fue posible corroborar su correcta integración.

- Por medio de la definición, implementación y ejecución de la alternativa de integración fue posible establecer criterios que permitieron la validación del mismo, permitiendo verificar algunas características del proyecto como son sus propiedades funcionales y no funcionales.

- A nivel personal pienso que la integración continua es un entorno que debería ser adoptado por la mayor parte de las empresas de desarrollo software de status medio-alto, ya que inicialmente tiene un esfuerzo y un costo elevado pero el ahorro de recursos y tiempo lo compensa, además de la facilidad de ejecución de las pruebas, con solo un “click” se podrán ejecutar las tareas en el ciclo de vida del proyecto de forma automatizada.

REFERENCIAS BIBLIOGRÁFICAS

[1] Crespo,Roberto.(Agosto 17,2015).Aprende a montar un entorno de integración continua (I). Descargado de: http://www.robertocrespo.net/kaizen/aprende-a-montar-un-entorno-de-integracion-continua-i/

[2] Lopez,Jesus.(Abril 30,2013).Ventajas y/o puntos fuertes de la integración continua. Descargado de:https://jesuslc.com/2013/04/30/ventajas-yo-puntos-fuertes-de-la-integracion-continua/

[3] Garzás,Javier.(Septiembre 25,2014).Beneficios de la integración continua. Descargado de:http://www.javiergarzas.com/2014/09/beneficios-integracion-continua.html

[4] Que es un sistema de control de versiones y por qué es tan importante.(Abril 30,2014). Descargado de: https://hipertextual.com/archivo/2014/04/sistema-control-versiones/

[5] Empezando-Fundamentos de Git. Descargado de: https://git-scm.com/book/es/v1/Empezando-Fundamentos-de-Git

[6] Mello,Martín.(Septiembre 10,2015).Mercurial,sistema de control de versiones.Descargado de: https://labi.fi.uba.ar/sites/default/files/cursos/mercurial/presentation_1.pdf

[7] Marco,Richard.(2004).Sistema de gestión de pagos electrónico, Mantenimiento y actualizaciones.Descargado de: http://www.biblioteca.unlpam.edu.ar/rdata/tesis/i_ricsis448.pdf

[8] Crespo,Roberto.(Agosto 26,2015).Aprende a montar un entorno de integración continua (IV). Descargado de: http://www.robertocrespo.net/kaizen/aprende-a-montar-un-entorno-de-integracion-continua-iv-jenkins/

[9] Garcia,Ana Maria.(Mayo 9,2014). ¿qué es jenkins?. Descargado de: http://www.javiergarzas.com/2014/05/jenkins-en-menos-de-10-min.html

[10] Rodriguez,Pilar.(Septiembre,2008).Estudio de la aplicación de metodologías ágiles para la evolución de productos software.Descargado de: http://oa.upm.es/1939/1/TESIS_MASTER_PILAR_RODRIGUEZ_GONZALEZ.pdf

[11] Procedimiento de implantación de un entorno de integración continua.(Enero 23,2016). Descargado de: https://www.ecured.cu/Procedimiento_de_implantaci%C3%B3n_de_un_entorno_de_Integraci%C3%B3n_Continua

[12] Ortiz,David.(Enero 28,2014).Integración continua y jenkins. Descargado de: http://www.agiliacenter.com/blog/integracion-continua-y-jenkins/

13] Camacho,Yeray.(s.f).Integración continua. Descargado de: https://emanchado.github.io/camino-mejor-programador/html/ch08.html

BIBLIOGRAFÍA

- http://mmorejon.github.io/blog/integracion-continua-jenkins-ios9-xcode/
- https://technologyconversations.com/2014/06/18/build-tools/
- http://www.thubanwiki.vivatia.com/index.php?title=Gu%C3%ADa_de_Integraci%C3%B3n_Continua_-_Jenkins
- https://camilorada.wordpress.com/2014/06/22/integracion-continua-con-jenkins-en-aplicativos-java/
- https://webxico.blogspot.com.co/2013/05/instalancion-y-configuracion-manual-de.html
- http://www.vogella.com/tutorials/JUnit/article.html
- http://www.javiergarzas.com/2014/05/control-versiones-scrum.html
- http://www.javiergarzas.com/2014/08/implantar-integracion-continua-2.html
- https://openwebinars.net/blog/docker-que-es-sus-principales-caracteristicas/
- https://docs.docker.com/engine/getstarted/
- https://www.genbetadev.com/metodologias-de-programacion/buenas-practicas-la-integracion-continua
- http://es.ccm.net/faq/4815-el-concepto-de-la-integracion-continua
- http://bibing.us.es/proyectos/abreproy/90879/fichero/TFG_JEnriqueRomero.pdf

ANEXOS

Anexo 1: Código del proyecto en estudio
Se anexa el link del código del proyecto en estudio

https://github.com/Giovanni2293/cloudBRT

Anexo 2: Código de las pruebas realizadas al caso de estudio
Se anexa los link del código de las pruebas realizadas

https://github.com/angelvalbuena/ProyecInte
https://github.com/angelvalbuena/RestTest
https://github.com/angelvalbuena/JmeTest

Anexo 3: Plan de pruebas
Se anexa el link del plan de pruebas
https://drive.google.com/open?id=1yXYo9e8mtdCU_oPYLxOBDLwcsrOOYjwlNMH2y0pg4rk

Anexo 4: Links de los repositorios donde están alojados los dockerfile de las imágenes utilizadas en Docker.

Repositorio de dockerfile de imagen de Tomcat
https://github.com/andynight/DockerfileProyect

Repositorio de dockerfile de imagen de Jenkins
https://github.com/angelvalbuena/DockerJenkins