ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"DISEÑO E IMPLEMENTACION DE UN SERVIDOR DE INTEGRACION CONTINUA DE CODIGO MULTIPLATAFORMA PARA UNA PEQUEÑA Y MEDIANA EMPRESA"

INFORME DE MATERIA INTEGRADORA

Previa a la obtención del Título de:

LICENCIATURA EN REDES Y SISTEMAS OPERATIVOS

FRANCISCO CRESPO CUESTA

WELLINGTON MENDOZA CALDERON

GUAYAQUIL – ECUADOR

AÑO: 2016

AGRADECIMIENTOS

Nuestro agradecimiento más importante a Dios por que nos ha privilegiado con grandes oportunidades. Agradecemos a nuestros padres que nos han apoyado. Y agradecemos a nuestros maestros quienes nos han sabido guiar por el camino del conocimiento.

DEDICATORIA

Dedicamos este proyecto a los desarrolladores de software, quienes trabajan para tener un mundo más eficiente, automatizando procesos.

Dedicamos este proyecto también a todas las personas que buscan contribuir con el conocimiento del mundo.

TRIBUNAL DE EVALUACIÓN

Ing. Roberto Patiño

PROFESOR EVALUADOR

Ing. María Angélica Santacruz

PROFESOR EVALUADOR

DECLARACIÓN EXPRESA

"La responsabilidad y la autoría del contenido de este Trabajo de Titulación, nos corresponde exclusivamente; y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Francisco Crespo

Wellington Mendoza

RESUMEN

En el presente trabajo se muestra como el Servidor de Integración Continua forma parte importante del desarrollo de software colaborativo eficiente.

Se toma como referencia, los valores y principios de desarrollo de software que dicta el Manifiesto Agile, y en base a estos se estudian las metodologías de desarrollo, las cuales necesitan de un Servidor de Integración Continua para la automatización de detección de errores en las actualizaciones de software, evitando en gran porcentaje los errores en ambiente de producción.

Para el estudio de este Servidor se realiza su implementación, la cual se documenta, para su análisis, se exponen los requerimientos y dependencias necesarios.

La aplicación con la que se trabaja para realizar la integración continua en nuestra implementación es Jenkins, se realizan pruebas con códigos sencillos hechos en java, y se escribe la programación de las pruebas en groovy. Con estas pruebas se demuestra cómo funciona el sistema, y como se logra analizar y encontrar los errores de lógica en cada porción de código que se sube, sin importar quien lo suba, por medio de cada test que se realiza.

ÍNDICE GENERAL

INDICE DE FIGURAS

INDICE DE TABLAS

INTRODUCCIÓN

Empresas de desarrollo de software que siguen el “Modelo de desarrollo en Cascada”, tienden a cometer errores comunes dentro del ciclo de desarrollo y entrega del producto al cliente.

Un claro ejemplo es la entrega de código estable en ambientes de producción que a su vez causan daños colaterales en las partes ya funcionales del sistema.

En la actualidad se promueve el desarrollo moderno basando sus principios y prácticas de acuerdo a metodologías de “Desarrollo Agile”, como Scrum, Extreme Programming, Lean, Kanban, etc. (Manifiesto Agile, 2001)Una de las prácticas que el desarrollo agile promueve es el uso de “Servidores de Integración Continua”, que luego podrían pasan a ser “Servidores de Entrega Continua”, y posteriormente a “Servidores de Despliegue Continuo”.

En el transcurso del tiempo a través de estas metodologías agiles, surge una cultura llamada “DevOps” [1], un perfil intermedio entre Desarrollador y Operaciones (tareas por lo general asignadas al administrador de sistemas).

Este es el personal que finalmente se encarga de hacer que los procesos, cumplan con una serie de políticas y estándares para asegurar la funcionalidad y estabilidad, utilizando herramientas modernas de automatización.

CAPITULO 1

1. MARCO TEÓRICO

Día a día la realidad de un departamento de sistemas, que por lo general consta de administradores y programadores; es que ambos perfiles realizan sus tareas totalmente independientes uno del otro, y casi siempre llegan a un extremo de “no trabajar colaborativamente”, quizás el trabajo colaborativo no es necesario en todos los escenarios, pero cuando el administrador debe mantener un sistema empresarial que usa bases de datos, servidor web, servidor de aplicaciones, y más; debe existir total colaboración entre ambos equipos, o al menos ambos deberían estar completamente consciente de las herramientas, y metodologías que usa cada equipo.

Aun si pareciera que ambos perfiles deberían trabajar de forma independiente porque sus áreas son diferentes, lo cierto es que en la actualidad la tecnología está tan intrínsecamente ligada entre sí que es casi imposible pensar en un proyecto tecnológico sin tener en consideración algunos factores comunes que este involucra, ejemplo: requerimientos mínimos de hardware para soportar el software, seguridad en la transmisión de datos, persistencia de datos, arquitectura de software, frameworks que se van a usar (sean de código abierto o cerrado), etc. Adicionalmente se suma la administración del proyecto, calcular tiempos estimados, designar responsabilidades y en fin.

A lo largo del tiempo las prácticas que antes eran buenas, en algunos casos suele suceder que dejan de ser buenas prácticas, y de la misma manera cambian las formas de planificación, de análisis, y diseño. Pero es gracias a todos estos cambios, que actualmente hay una comunidad grande de científicos revisando, cuestionando y probando cada cambio que hay en el mundo tecnológico, muchas veces ellos son quienes aprueban ciertos movimientos como norma o standard.

Un claro ejemplo de todo esto, en cuanto a proyectos de software se refiere y que es relativamente nuevo en el campo de la tecnología, es que actualmente todo converge en las “Metodologías Ágiles”, difundidas en gran manera por su eficacia y eficiencia.

1.1. OBJETIVO

Objetivo General

Minimizar el riesgo de cargar código que pueda afectar la disponibilidad de los servidores de producción, y agilitar la depuración de código inestable, a través de un Servidor de Integración Continua.

Objetivos Específicos

Preparar un ambiente de desarrollo con pruebas unitarias para satisfacer la necesidad del servidor de Integración Continua.

Integrar código inestable con el Servidor de Integración y mostrar los errores de código al desarrollador a través de las pruebas unitarias.

Presentar las causas de error a los desarrolladores a través de una interfaz amigable, para que haya retroalimentación al desarrollador y pueda realizar los cambios respectivos que corrigen el error.

1.2. JUSTIFICACIÓN

La integración continua como pieza importante del desarrollo agile, busca automatizar la detección de errores de código de software en un equipo de desarrolladores, mejorando la retroalimentación, para su posterior solución, evitando así problemas en los ambientes de producción.

Según un artículo escrito en febrero del 2013 por Mike Cohn, fundador de “Mountain Goat Software” acerca del éxito de las metodologías Ágiles por sobre las metodología en cascada, en donde se hace referencia al informe del 2011 CAOS del Standish Group, nos indica que el proceso agile es el remedio universal para las fallas de los proyectos de desarrollo de software, el desarrollo de software a través de procesos ágiles tiene tres veces más casos de éxito que el método tradicional.

En los métodos tradicionales de desarrollo encontramos que hay un 57% de fallo y un 14 % de éxitos, sin embargo con los métodos ágiles encontramos que hay un 9% que falla y un 42% que son exitosos.

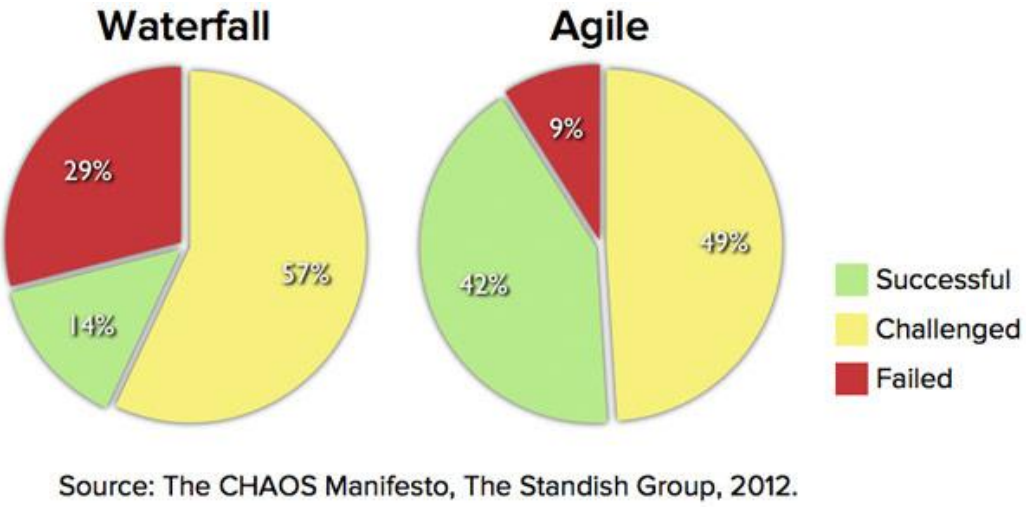

Figura 1.1 Estadística Agile vs Cascada

En el rol de los administradores de sistemas se tiene la obligación de mantener la estabilidad y disponibilidad de los servicios que se entregan a los clientes.

En temas de aplicaciones web, se debe asegurar que las actualizaciones de estas aplicaciones hayan sido probadas antes de llegar al servidor de producción, pero en muchos casos a pesar de haber pasado por varias pruebas los sistemas siguen llegando con errores.

Cuando esto sucede, los administradores son responsables de regresar el sistema a un estado estable, pero no es suficiente solo con revertir los cambios, ya que este problema involucra tanto a administradores de sistemas como desarrolladores, es importante dar una buena retroalimentación de la causa del problema, y prevenirlo en la medida en que ambas partes (administradores y desarrolladores) sean comprometidas. Por ello es necesario conocer el equipo de desarrollo y las metodologías que se siguen, a fin de proponer una solución común.

Este proyecto trata como tema principal, el demostrar las razones de uso de un servidor de integración continua, que servirá como filtro para asegurar a los desarrolladores que su código está cumpliendo con las pruebas específicas para satisfacer las necesidades del cliente, sin causar daños colaterales en la actualización del código al ambiente de producción.

Las tecnologías con las que se realizará el análisis dependen del ambiente de producción que aloja el código.

Cabe destacar que para cada lenguaje existe un framework de pruebas unitarias que facilita la retroalimentación de errores en el código.

1.3. METODOLOGÍA

La metodología se llevará a cabo con el desarrollo de las siguientes fases:

Fase 1:

Recolectar información sobre las herramientas y metodologías que dan solución al problema.

Fase 2:

Diseñar una metodología para adaptar el uso del Servidor de Integración Continua al ciclo de desarrollo web.

Fase 3:

Implementación del Servidor de Integración Continua.

Fase 4:

Pruebas de integración con diferentes versiones de código inestable, y retroalimentación al equipo de desarrollo por parte del servidor.

1.4. Metodologías para el desarrollo de software

El desarrollo de software no siempre fue el resultado de un proceso o metodología definida.

En los años cincuenta no existían metodologías descritas para el desarrollo de software. A finales de la década de los sesenta se empieza hablar de la ingeniería de software.

La ingeniería de software tiene como objetivo que el desarrollo sea un proceso formal, de modo que se tengan los lineamientos de cómo recibir los requerimientos, cómo realizar el diseño, pruebas, implementación y mantenimiento.

Con esto nacen herramientas, metodologías, procesos y tecnologías, los cuales serían solución a los problemas de planificación, presupuestos, y calidad en el desarrollo de software.

Una metodología de desarrollo de software establece un proceso con el cual se puede tener mejor comunicación entre las partes involucradas en la construcción de un sistema.

1.4.1. Metodología PSP/TSP

Entre algunas de las metodologías tradicionales que fueron surgiendo está el TSP/PSP, PSP (Proceso Personal de Software), TSP (Proceso en Equipo para el software).

El PSP dicta el proceso para que cada profesional del software sea responsable de la calidad del producto que desarrolla, tiene como objetivos:

Mejorar las habilidades de planeación del proyecto

Establecer compromisos reales

Mejorar la calidad de los procesos del desarrollador

Minimizar la cantidad de errores en el trabajo realizado.

El TSP ofrece un entorno que apoya el trabajo en equipo, establece ciclos de desarrollo.

Los ingenieros de software necesitan conocer PSP, para trabajar en proceso de software en equipo TSP.

El TSP ayuda a los ingenieros de software, gerentes, y administradores de proyecto, creando un marco de trabajo de procesos definidos para el desarrollo de software, el objetivo de TSP es mejorar la calidad y productividad de un equipo de desarrollo de software.

1.4.2. Metodologías Ágiles

Los negocios y empresas trabajan en un entorno cambiante, y deben responder a nuevas oportunidades, mercados, productos, competidores. Siendo el software de un negocio parte fundamental de cada uno de los procesos internos de cada empresa, es necesario que este se vaya adaptando a cada cambio.

Cuando el proceso de desarrollo de software se basa estrictamente en las especificaciones de los requerimientos y diseño, éste no se ajusta a un desarrollo rápido.

Normalmente los requerimientos cambian o se descubren nuevos problemas, por lo que se debe incurrir en un proceso de rediseño, lo cual conlleva mucho tiempo, prolongando la entrega del software definitivo.

La metodología Agile busca la satisfacción del cliente y la entrega continua de software [2], por lo que se establecen procesos en los que se involucran a los desarrolladores usuarios y administradores en las especificaciones, diseño, desarrollo, y las pruebas del software.

1.4.3. Integración Continua en Metodologías Ágiles

La necesidad de usar un servidor de CI (Continuous Integration) surge cuando se quiere entregar de manera iterativa e incremental el desarrollo de un software [3]. Y es en esta situación que equipos de desarrolladores que trabajan individualmente diferentes porciones de

código, cuando deciden integrar entre sí el proyecto el proceso se vuelve un conflicto, si surgen errores luego de la integración esto puede retrasar la entrega iterativa del software por la resolución de errores.

Las razones de hacer una entrega iterativa de un proyecto de software tienen sus principios en las metodologías ágiles, que por lo general cumplen como una de sus prácticas entregar un desarrollo incremental del proyecto en periodos de tiempo establecidos con el cliente, satisfaciendo la necesidad inmediata del cliente. Todo esto con la finalidad de poder retroalimentarse a tiempo del criterio del cliente, es decir; mientras aún sigue en desarrollo el software.

1.5. Integración, Despliegues y Entrega Continua

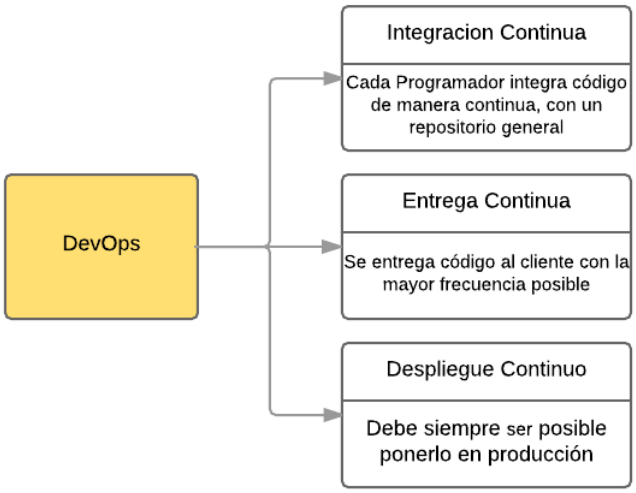

Figura 1.2: Integración, Entrega y Despliegue Continuo

Continuando con el propósito del servidor de CI. Generalmente esta necesidad del servidor surge con aquellos equipos que practican metodologías ágiles ya que las entregas iterativas los obligan a integrar constantemente código [4][5][6]. Pero para metodologías ágiles, existen 3 soluciones similares:

1.5.1. Integración Continua

Es el más popular y el primero en conocer, su uso es poder alertar al equipo sobre errores de integración en el código a través de una interfaz intuitiva.

1.5.2. Entrega Continua

Realiza las mismas funciones que el servidor de CI, pero adicionalmente hace la construcción (build) del proyecto generando un compilado, que luego está disponible para llevarlo a cualquier ambiente deseado por el cliente.

1.5.3. Despliegue Continuo

Llegar a implementar un servidor así es muy complejo, ya que hace las mismas funciones que integración continua y entrega continua, pero su

valor principal está en su automatización para actualizar este código de forma desatendida en cualquier ambiente deseado.

En este proyecto nos enfocaremos en un servidor de Integración Continua.

1.6. Consideraciones para implementar el servidor de Integración Continua

Las consideraciones de implementación de un Servidor de CI generalmente se basan en dos factores principales:

Personal capacitado

Complejidad de software

1.6.1. Personal capacitado en Desarrollo y Operaciones.

Empresas que quieran implementar esta solución en su infraestructura, necesitan de un perfil que cumpla principalmente el rol de administrador de servidores, para mantener consistencia en la infraestructura de red y todas las políticas actualmente establecidas.

Normalmente lo primero a considerar es la ubicación del servidor en el diseño de la infraestructura, deben decidir si el servidor debe estar alojado internamente o en la nube y sea este virtual o físico, además deben asegurarse de que el servidor tiene suficiente acceso para descargar o actualizar las dependencias de software que tiene el código.

Luego deben tomando decisiones teniendo en cuenta que el diseño podría contar con una zona desmilitarizada, políticas para asignación de direcciones ip públicas y privadas, restricciones de acceso a la red por vpn, proxies, firewall, o restricciones de acceso por usuario y grupos de usuarios a servicios internos y externos de la red, etc.

Estos son temas que normalmente un desarrollador no debería verse obligado a conocer.

Y finalmente se debe realizar la configuración del servicio de CI, de esto depende que el administrador conozca el software, cómo programar crons, eventos del versionador, ruta del proyecto, etc.

Para aquellas empresas que no desean mantener este servidor bajo su responsabilidad existen actualmente soluciones de servidores CI en la nube, hay que diferenciar que están aquellas que son para código abierto y para código privado, entre las más populares están Jenkins, GitLab, Snap CI, Travis, Team City, Team Fundation Server, pero existen más soluciones.

1.6.2. Complejidad de Software.

Este caso depende mucho del software a desarrollar, por instancia; si tarda mucho el proceso de build del proyecto incluyendo los tests, probablemente estamos trabajando con una lógica de negocio que requiere mucho procesamiento, o en su lugar depende de varios servicios online y estos tardan en responder al servidor, de cualquier forma cuando este tipo de procedimientos tardan, habría que considerar el hecho de soportar un servidor con las capacidades necesarias, o alojar el servicio en la nube.

1.7. Requerimientos del Servidor de CI

Los requerimientos de hardware para el servidor de CI son exactamente los mismos que del proyecto a desarrollar, sin embargo hay que tomar en cuenta que el servidor de CI dependiendo de cómo sea programado tendrá crons (para automatizar hacer builds), o también podría tener programado eventos del versionador para ejecutar builds, si el proceso de building es pesado y las integraciones de software son frecuentes, probablemente estemos encolando builds y no tendremos una retroalimentación temprana del servidor.

La frecuencia de estas actividades y el tiempo que tardan en dar respuesta al cliente, son alertas para considerar aumentar las capacidades del servidor, si por ejemplo; el proceso de building y testing pone en cola más builds conforme se incrementa el código, entonces el equipo debe investigar si es algún problema por parte del rendimiento del código, o si verdaderamente es bajo rendimiento del servidor.

1.8. Administración del Servidor de CI

En cuanto a los ambientes de servidor, es preferible que sean manejadas usando cajas de máquinas virtuales automatizadas, ya sea con vagrant o docker (las más populares); cualquiera de estas soluciones permite que tanto el equipo de desarrollo como el administrador de servidores mantengan copias exactas de los ambientes y no se pierda o se alteren las configuraciones de servidor.

Seguido a esto también se suma el uso de herramientas para automatizar la administración de configuraciones del servidor, tecnologías como Ansible, Chef, Puppets (las más populares) gracias a su scripts facilitan procesos de administrador como instalación desatendida de software, mantenimiento de logs, ejecución de scripts, programación de crons, etc. Todo esto con el fin no comprometer al desarrollador con las labores del administrador.

Hay que destacar que lo mencionado en esta sección no es imprescindible para sacar provecho del servidor de CI, pero que son soluciones inteligentes y actuales que acompañan la solución que ofrece el servidor de CI.

1.9. Servidor de CI en la práctica

El lenguaje más popular en programación es Java, y a su vez [7] Jenkins es el software de CI más conocido en el mundo de la programación.

Jenkins surgió como un ‘fork’ del proyecto Hudson de código abierto desarrollado por Oracle, este software está escrito en lenguaje Java. Actualmente Jenkins es soportado por la comunidad Java, quienes han surtido al proyecto con variedad de plugins para poder satisfacer diferentes necesidades como hacer test sobre diferentes lenguajes de programación (ruby, php, python, etc), o automatizar builds usando versionadores (git, svn, etc), incluso dando soporte de herramientas de building para ese ambiente de desarrollo (ant, maven, gradle, composer, etc).

Llevando el uso de Jenkins a la práctica se nos presentan ciertas condiciones, y esto le corresponde al arquitecto de software o al desarrollador senior

encargado de velar por las políticas de calidad y las buenas prácticas empleadas en el desarrollo del software.

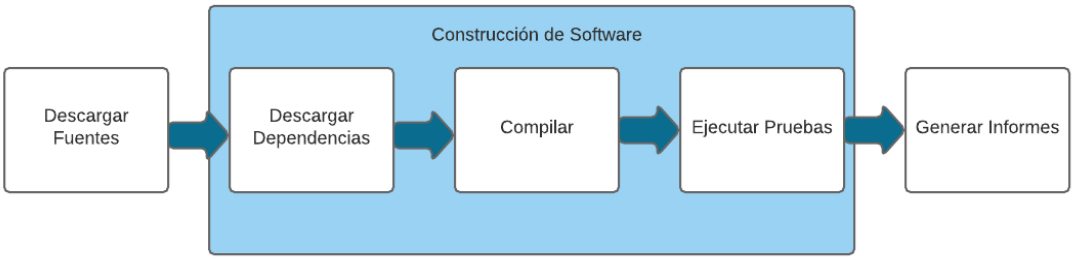

Figura 1.3: Proceso de Integración Continua

1.10. Soporte de Jenkins en el proyecto

Jenkins nos exige 3 cosas principalmente, para que el proyecto pueda ser reconocido.

Suite de tests

Archivo de configuración de construcción del Proyecto

Herramienta de Build.

1.10.1. Suite de tests.

Esto nos obliga a usar un framework de testing que se encargue de entregar estas respuestas al servidor, el framework de testing más conocido en java es Junit. Junit fue desarrollado en sus comienzos por Kent Beck, es de código abierto y actualmente es soportado por la comunidad de java.

El procedimiento más común para integrar un framework de testing en un proyecto es descargarlo y ubicarlo en algún directorio estratégico del proyecto donde se pueda hacer uso del package.

Actualmente este tipo de dependencias o también llamado package, que un proyecto tiene sobre otro, se ha solucionado en gran parte gracias a

la comunidad de cada lenguaje, siguiendo los principios de linux de usar una herramienta de instalación basada en un repositorio con proyectos de diferentes niveles de desarrollo (alpha, beta, etc), el ejemplo más conocido en linux es el comando apt de las distribuciones basadas en Debian, que lee un archivo de configuración que actúa como fuente central de repositorios donde puede buscar y descargar el software deseado. De la misma manera se han implementado soluciones que se encargan de resolver estas dependencias de código que puede tener un proyecto sobre otro.

Su principal propósito es automatizar las actualizaciones de proyectos de terceros sin temer el alterar o cambiar código base. Herramientas como Composer (php), Maven (java), Npm (node.js), hacen que levantar un ambiente de desarrollo para un novato en el equipo sea relativamente fácil, pero esto hay que compensarlo aprendiendo de las herramientas.

1.10.2. Archivo de Construcción del Proyecto

Generalmente para que nuestro proyecto soporte interacción con el servidor de CI, el servidor depende de un archivo de configuración comúnmente alojado en el directorio raíz del proyecto, donde se especifica:

Ubicación de la suite de tests y la ubicación del framework de Testing

Ubicación del archivo de building y la ubicación del comando

El formato del archivo así como su nombre y extensión, están escritos en la documentación del proyecto de CI, en Jenkins por ejemplo, existe una interfaz visual que se encarga de facilitar esta dependencia.

1.11. Herramientas que facilitan el trabajo del servidor de CI

1.11.1. Administrador de paquetes o dependencias

En las comunidades de programación existen muchos proyectos de open source que como es normal, surgen para satisfacer las necesidades de ese momento, pero ya que vivimos en un mundo globalizado, y la mayoría de proyectos quieren entregas rápidas, es complicado pensar en desarrollar algo que ya está hecho y que además es de uso libre, aquí es donde actúan las comunidades de programación que aportan sus proyectos como código libre para uso de los demás.

Así pues, con la innovación de los lenguajes de programación y las necesidades emergentes para automatizar y facilitar el desarrollo, se crearon administradores de paquetes o dependencias, que como su nombre lo indica es un software que se encarga de descargar las dependencias que un proyecto pueda tener, ej: Si se está haciendo un desarrollo web en php, y el proyecto depende de una librería de ORM, un Router, y Collection, aunque muchos quisieran hacer el desarrollo individual de cada dependencia; lo más probable es que alguien más haya desarrollado algo similar o igual hace mucho tiempo, y es mucho más probable que este proyecto esté en mantenimiento por mucha gente que se sirve del mismo, aquí entra el administrador de paquetes, en php el administrador de paquetes seria Composer, y solo con agregar un archivo en un formato específico, Composer puede identificar de qué otros proyectos depende el proyecto principal, y finalmente cuando se ejecute el comando de Composer; éste se encargará de descargar y actualizar todas las dependencias de código que tenga el proyecto conforme se haya dado mantenimiento y actualizaciones en esos proyectos.

1.11.2. Herramientas de Building

En ambientes de programación como Java que generan una respuesta del código en un paquete compilado, o lenguajes que entregan su código fuente al cliente como Javascript, es muy común el uso de este tipo de herramientas.

Tomemos como ejemplo javascript para programación frontend. Normalmente al servir una página web el comportamiento dinámico de la página la controlamos usando javascript, éste generalmente se descarga en la caché del cliente cumpliendo ciertos requisitos: que sea minificado, que sea versionado, y que su definición global no afecte otras librerías/frameworks.

En node.js esta solución se ha presentado con varios proyectos que cumplen funciones similares, es decir, ejecutan tests, minifican, versionan, hacen linting de código, brindan soporte de la versión 6 de javascript para varios navegadores, compilan lenguajes de preprocesadores en js/css/html y más, pero es el resultado de todo esto lo que debe descargarse en el cliente, ejemplos de herramientas: bower, grunt, gulp, browserify, webpack (las más populares).

Pero ese no es el foco principal de proyectos backend, en este ambiente se busca poder automatizar el código para que sea capaz de levantar su propio sistema sin depender de la ayuda de terceros, en la práctica vemos que se programa la creación y modificación de tablas en la base de datos, se manipula la estructura de archivos con sus respectivos permisos de acceso, y cualquier otro tema que sea relacionado para levantar correctamente el sistema.

Si tomamos como ejemplo Java, estos builds podrían estar escritos usando ant, maven, o gradle (las más populares). Aunque los tres

cumplen el mismo propósito, la selección de alguna de estas herramientas depende de la facilidad de lectura y mantenimiento, y si tomamos por ejemplo gradle que usa groovy que a su vez se ejecuta sobre la jvm, su preferencia de uso radica en su sintaxis que es fácil de entender e interpretar (groovy está inspirado en ruby).

Pero no importa que herramienta se haya seleccionado, finalmente lo que se quiere lograr es poder levantar el sistema con un par de comandos y no haciendo operaciones manuales.

CAPITULO 2

2. IMPLEMENTACIÓN

2.1. Entorno de pruebas usando Vagrant

Vagrant es una herramienta que se encarga de automatizar la creación y configuración de máquinas virtuales dependiendo del proveedor del virtualizador (vmware, virtualbox, etc), a través de un script escrito en lenguaje ruby [8].

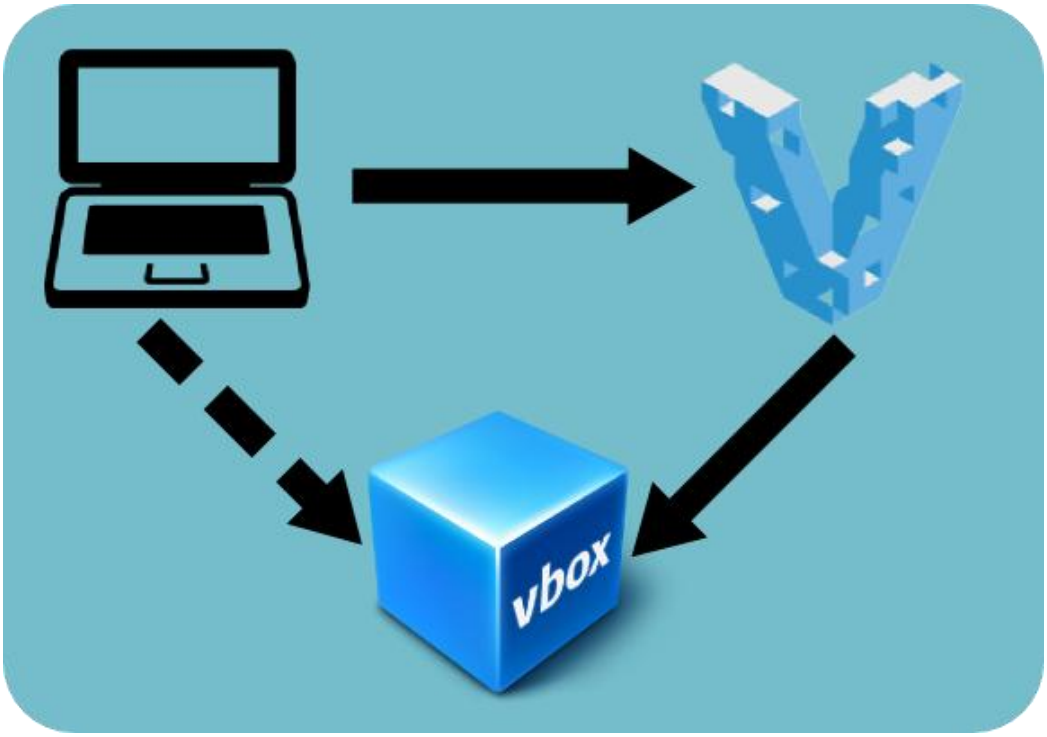

Figura 2.1: Entorno de Pruebas con Vagrant

Para utilizar vagrant correctamente, se debe crear un directorio (que servirá como repositorio), y dentro crear un archivo sin extensión llamado *Vagrantfile*, en el que se indica (usando el api de vagrant) de donde crear o usar la máquina virtual, que configuraciones tendrá nuestra máquina virtual, y si fuera necesario;

ejecutar scripts en bash con privilegios de administrador para la instalación de paquetes en nuestro ambiente virtual.

```
~ jenkins ll
total 8,0K
-rw-r----- 1 well well 3,4K Dec 16 20:55 provision.sh
-rw-r----- 1 well well 1,1K Dec 16 20:55 Vagrantfile
```

Figura 2.2: Archivos para Vagrant

Una vez que se tienen ambos archivos, se puede levantar la máquina virtual usando vagrant, pero antes de levantar la máquina virtual, hay que explicar el proceso de instalación.

Normalmente cuando se inicia un ambiente virtual usando vagrant, lo primero que el sistema hace es leer el archivo de configuración de la máquina virtual *Vagrantfile*, y en el mismo archivo ubica el repositorio de donde va a descargar la 'caja virtual' con el sistema operativo.

```
# Vagrantfile API/syntax version. Don't touch unless you know what you're doing!
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|

  # Every Vagrant virtual environment requires a box to build off of.
  # Named boxes, like this one, don't need a URL, since the are looked up
  # in the "vagrant cloud" (https://vagrantcloud.com)
  config.vm.box = "bento/debian-7.8"

  # Publish guest port 6060 on host port 6060
  config.vm.network "forwarded_port", guest: 6060, host: 6060
  config.vm.network "forwarded_port", guest: 8080, host: 7070

  config.vm.provider "virtualbox" do |vb|
  #   # Don't boot with headless mode. Use for debugging
  #   vb.gui = true

  #   # Use VBoxManage to customize the VM. For example to change memory:
  vb.customize ["modifyvm", :id, "--memory", "1024"]
  end

  # Provision the box using a shell script
  # This script is copied into the box and then run
  config.vm.provision :shell, :privileged => true, :path => "provision.sh"
end
```

Figura 2.3 Vagrantfile

Luego de haber realizado las configuraciones de la máquina virtual, se le indica por medio de un script en bash los comandos que debe ejecutar para instalar los paquetes que necesarios en el sistema operativo.

2.2. Descripción de la instalación en bash script

Jenkins tiene integración nativa para diferentes distribuciones de sistemas operativos linux, en este caso se usará Debian 7.8 (por facilidad de uso en el ambiente virtual) [9].

El archivo *provision.sh* tiene en orden los comandos necesarios para instalar un servidor de Jenkins correctamente, el script se describe a continuación:

- Instalación de Java

```
if [ ! -f /usr/lib/jvm/java-7-oracle/bin/java ];
then
    echo "-------- PROVISIONING JAVA ------------"
    echo "---------------------------------------"

    ## Make java install non-interactive
    ## See http://askubuntu.com/questions/190582/installing-java-automatically-with-silent-option
    echo debconf shared/accepted-oracle-license-v1-1 select true | \
      debconf-set-selections
    echo debconf shared/accepted-oracle-license-v1-1 seen true | \
      debconf-set-selections

    ## Install java 1.7
    ## See http://www.webupd8.org/2012/06/how-to-install-oracle-java-7-in-debian.html
    echo "deb http://ppa.launchpad.net/webupd8team/java/ubuntu precise main" | \
      tee /etc/apt/sources.list.d/webupd8team-java.list
    echo "deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu precise main" | \
      tee -a /etc/apt/sources.list.d/webupd8team-java.list

    apt-key adv --keyserver keyserver.ubuntu.com --recv-keys EEA14886
    apt-get update
    apt-get -y install oracle-java7-installer
else
    echo "CHECK - Java already installed"
fi
```

Figura 2.4 Instalación de Java

- Instalación de Jenkins:

```
if [ ! -f /etc/init.d/jenkins ];
then
    echo "-------- PROVISIONING JENKINS ------------"
    echo "------------------------------------------"


    ## Install Jenkins
    #
    # URL: http://localhost:6060
    # Home: /var/lib/jenkins
    # Start/Stop: /etc/init.d/jenkins
    # Config: /etc/default/jenkins
    # Jenkins log: /var/log/jenkins/jenkins.log
    wget -q -O - http://pkg.jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add -
    sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ > /etc/apt/sources.list.d/jenkins.list'
    apt-get update
    apt-get -y install jenkins

    # Move Jenkins to port 6060
    sed -i 's/8080/6060/g' /etc/default/jenkins
    /etc/init.d/jenkins restart
else
    echo "CHECK - Jenkins already installed"
fi
```

Figura 2.5: Instalación de Jenkins

Desde este punto en adelante no es obligatorio instalar los siguientes paquetes:

- Instalación de Tomcat:

```
### Everything below this point is not stricly needed for Jenkins to work
###

if [ ! -f /etc/init.d/tomcat7 ];
then
    echo "-------- PROVISIONING TOMCAT ------------"
    echo "-----------------------------------------"


    ## Install Tomcat (port 8080)
    # This gives us something to deploy builds into
    # CATALINA_BASE=/var/lib/tomcat7
    # CATALINE_HOME=/usr/share/tomcat7
    export JAVA_HOME="/usr/lib/jvm/java-7-oracle"
    apt-get -y install tomcat7

    # Work around a bug in the default tomcat start script
    sed -i 's/export JAVA_HOME/export JAVA_HOME=\"\/usr\/lib\/jvm\/java-7-oracle\"/g' /etc/init.d/tomcat7
    /etc/init.d/tomcat7 stop
    /etc/init.d/tomcat7 start
else
    echo "CHECK - Tomcat already installed"
fi
```

Figura 2.6 Instalación de Tomcat

- Instalación de Git

```
if [ ! -f /usr/bin/git ];
then
    echo "-------- PROVISIONING GIT ------------"
    echo "--------------------------------------"

    ## Install git
    apt-get update
    apt-get -y install git
else
    echo "CHECK - Git already installed"
fi
```

Figura 2.7: Instalación de Git

Con estos archivos listos, ya podemos iniciar nuestro ambiente virtual con vagrant.

```
 jenkins  vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'bento/debian-7.8' is up to date...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 6060 => 6060 (adapter 1)
    default: 8080 => 7070 (adapter 1)
    default: 22 => 2222 (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default: Warning: Connection timeout. Retrying...
    default: Warning: Remote connection disconnect. Retrying...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the installed version of
    default: VirtualBox! In most cases this is fine, but in rare cases it can
    default: prevent things such as shared folders from working properly. If you see
    default: shared folder errors, please make sure the guest additions within the
    default: virtual machine match the version of VirtualBox you have installed on
    default: your host and reload your VM.
    default:
    default: Guest Additions Version: 5.0.4
    default: VirtualBox Version: 4.3
==> default: Mounting shared folders...
    default: /vagrant => /home/well/Code/jenkins
==> default: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> default: to force provisioning. Provisioners marked to run always will still run.
```

Figura 2.8: Puesta en marcha de máquina virtual

Ahora podemos usar Jenkins

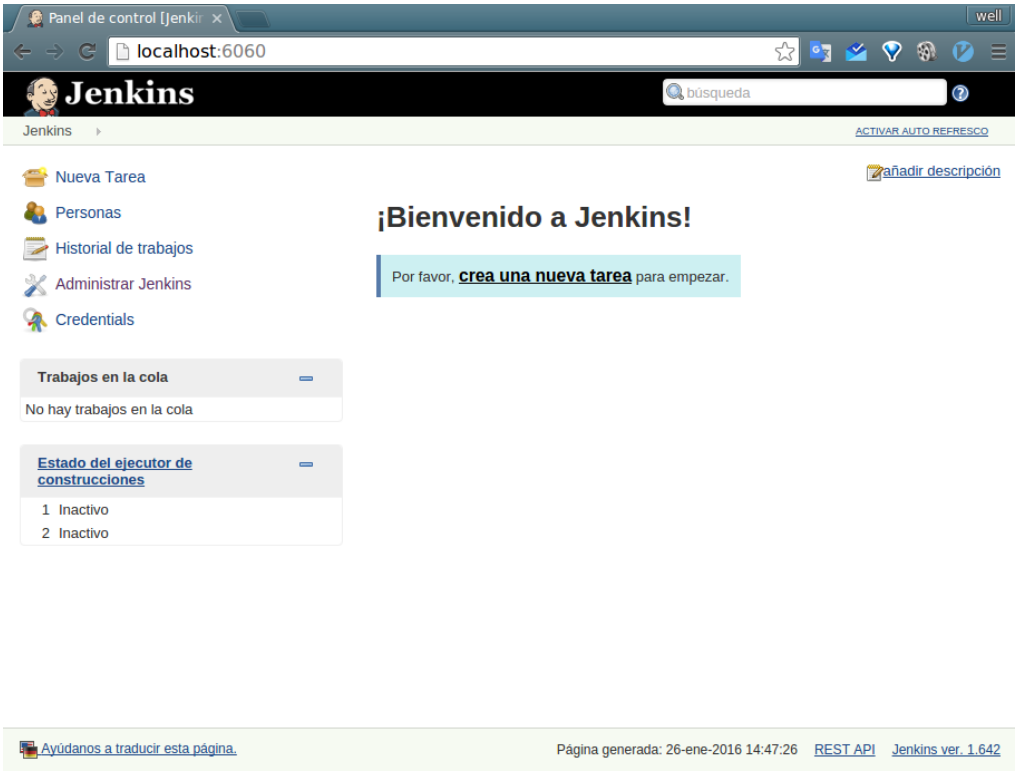

Figura 2.9: Interfaz web de Jenkins

2.3. Proyecto (código) de prueba

Para nuestro ambiente de pruebas contamos con estas condiciones en el código [10]:

- El proyecto está versionado con git y se aloja GitHub
- El código está escrito en Java
- Los pruebas están escritas en Groovy
- Se usa Gradle como herramienta de building

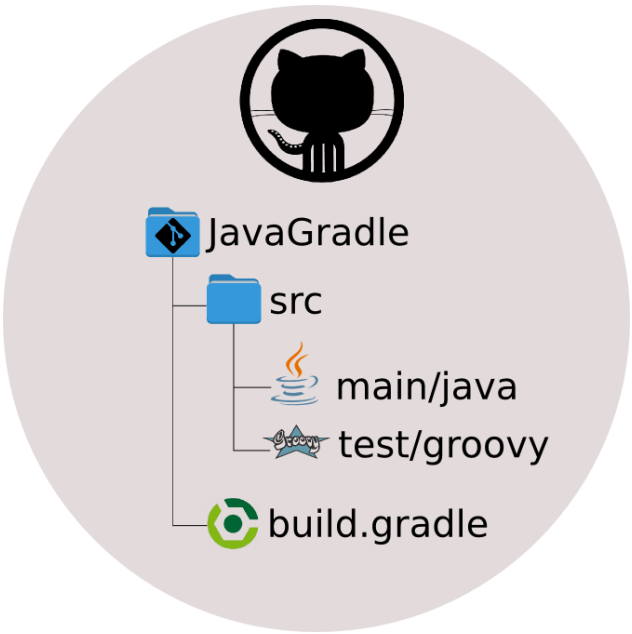

Figura 2.10: Estructura de archivos del Proyecto (código)

Aquí presentamos el código en github.

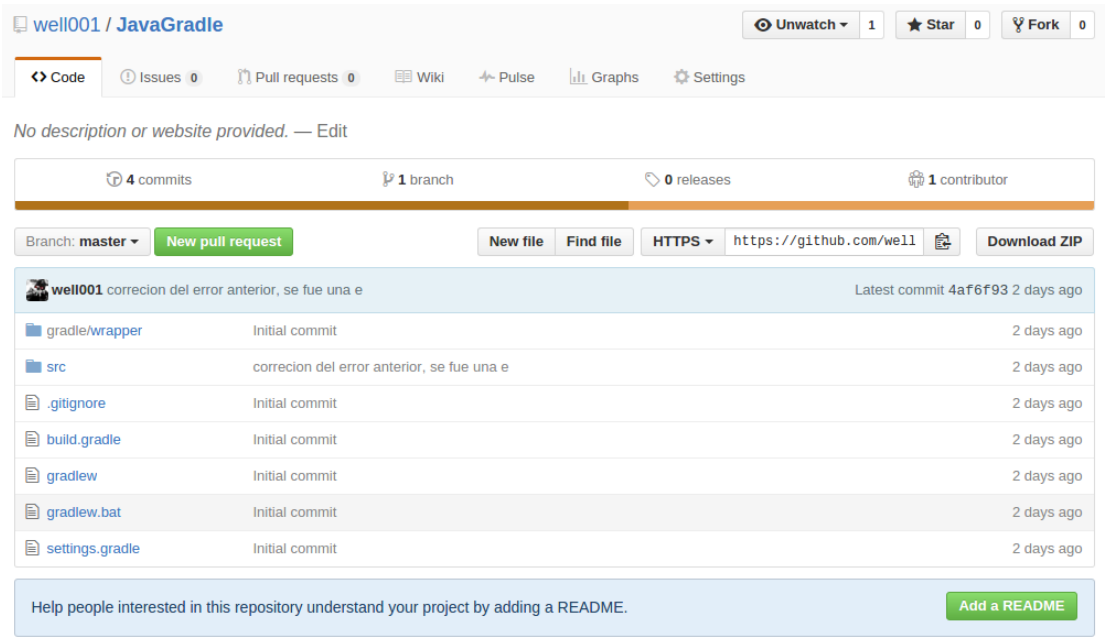

Figura 2.11: Proyecto de Prueba en GitHub

2.4. Configuración de Jenkins para integrar el proyecto

2.4.1. Plugins en Jenkins para compatibilidad

Para suplir las dependencias del código antes de poder crear nuestra primera tarea, debemos instalar los plugins que se encargan de reconocer las herramientas que estamos usando.

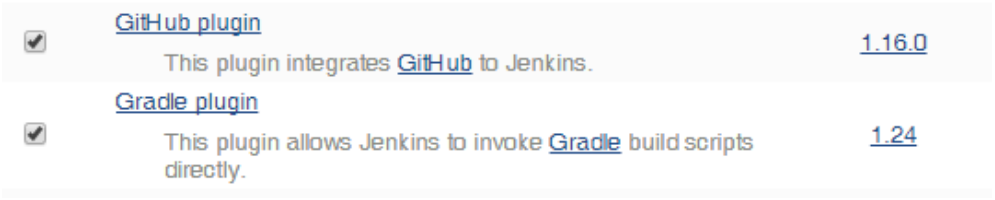

Figura 2.12: Activación de Plugins en Jenkins

Una vez terminada la instalación, se reinicia el servicio si habilitamos esa opción en la instalación de los plugins.

2.4.2. Creación de tarea en Jenkins

El primer paso en la creación del proyecto es identificarlo con un nombre, y en nuestro caso que usamos GitHub, incluir la dirección del proyecto que servirá como referencia.

Figura 2.13: Creación de tarea en Jenkins

El siguiente paso es indicarle la dirección de donde debe descargar el código. En esta prueba estamos usando Github, que es un repositorio público, por tanto no necesitamos credenciales para descargar el código.

Figura 2.14: URL de Repositorio de Proyecto

Finalmente, ya que nuestro proyecto cuenta con una herramienta de building, le indicamos que comandos debe ejecutar para hacer el building del proyecto.

Figura 2.15: Herramienta de Building

2.5. Ejecutar buildings en nuestro proyecto

Para la ejecución de buildings, las metodologías más usadas son:

- Ejecución atendida
- Ejecución programada por horarios (cron)
- Ejecución por eventos del versionador

2.5.1. Ejecución atendida

Básicamente se entra al dashboard del proyecto y se ejecuta manualmente el botón

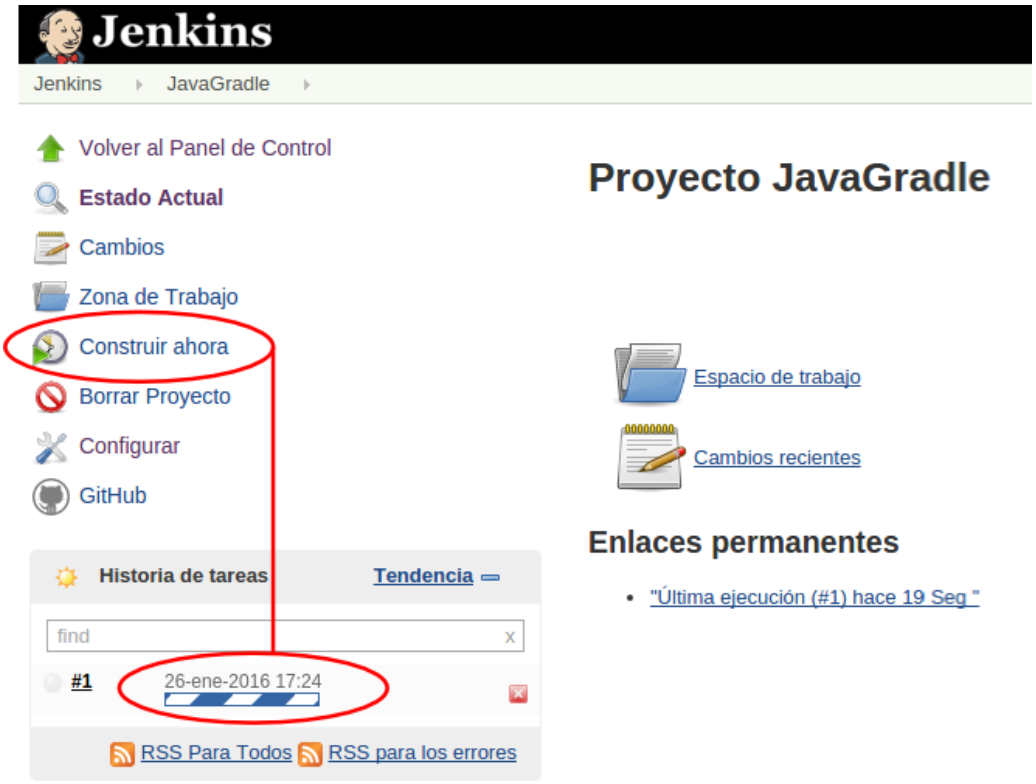

Figura 2.16: Ejecución atendida de test

2.5.2. Ejecución programada por horarios

En esta sección solo es necesario indicarle el horario siguiendo el formato standard de cron de linux. En el ejemplo lo hemos programado para que se ejecute a las 23 horas todos los viernes.

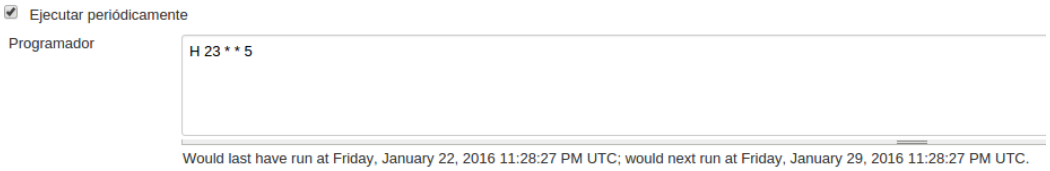

Figura 2.17: Ejecución de test por horario

CAPITULO 3

3. ANÁLISIS Y RESULTADOS

Una vez implementado el servidor, se debe conocer el proceso que ejecutan los sistemas involucrados en la integración continua, en este capítulo se explicará dicho proceso y se hará un análisis a los resultados obtenidos en laboratorio.

3.1. Proceso de Integración Continua

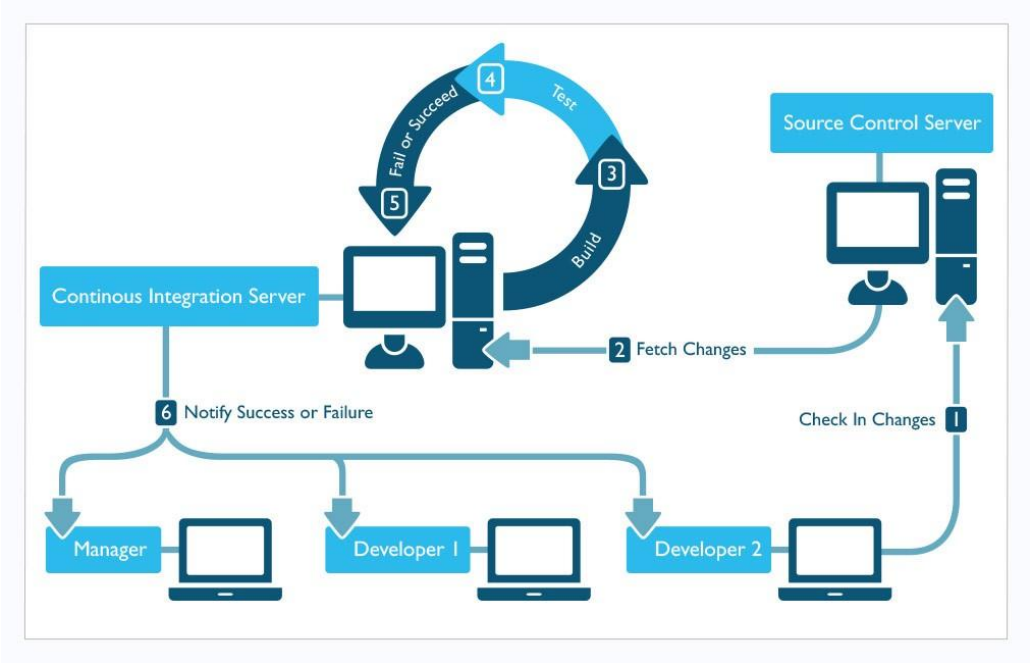

Figura 3.1: Proceso de Integración Continua

FUENTE: HTTPS://INSIGHTS.SEI.CMU.EDU/DEVOPS/2015/01/CONTINUOUS-INTEGRATION-IN-DEVOPS-1.HTML

El flujo de procesos que tendría el servidor de CI en el código, sería el siguiente:

Los desarrolladores actualizan su código subiéndolo a un repositorio central, que guarda los cambios de todo el proyecto. Para este laboratorio usaremos GitHub por ser público y popular en el medio.

El servidor de CI, cuando recibe la orden de construir el proyecto, este descarga las fuentes del proyecto desde el repositorio central.

El proceso de construcción de software requiere comprobar que no existan errores de sintaxis, y ejecutar la herramienta de construcción para automatizar el levantamiento de los requerimientos del software (ejemplo creación de tablas).

Una vez que el sistema se ha construido correctamente, empieza a ser ejecutada la suite de tests.

Cuando ha finalizado todo correctamente, el servidor guarda un historial de todo lo que se ha construido y funcionando en el servidor.

Finalmente el resultado está disponible y visible en una interfaz para uso del equipo.

3.2. Salida de Consola

Cada vez que el servidor está ejecutando un build, lo primero que podemos revisar para comprobar que se están realizando las pruebas y como se están realizando, es la consola de logs.

Generalmente aquí se ven los resultados que saldrían si construyéramos el proyecto manualmente usando el IDE o la consola, en la siguiente imagen vemos como se aprecia el resultado de un build exitoso.

Figura 3.2: Build Exitoso

Si luego algún programador decide subir cambios al repositorio, y entre estos cambios hay errores, este es el resultado esperado de parte del servidor.

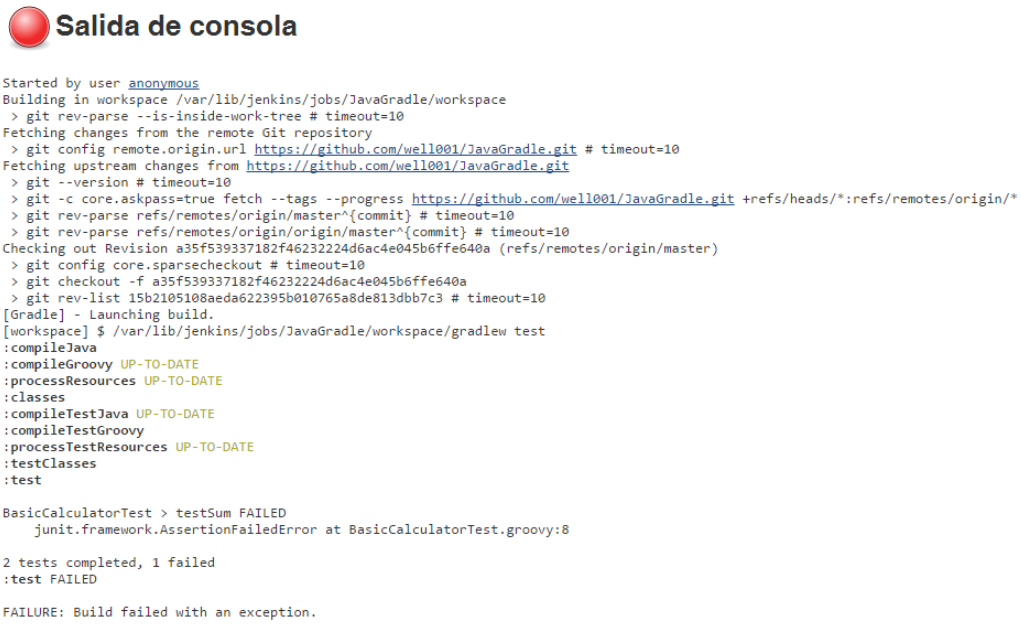

Figura 3.3: Build con errores

3.3. Build

Si quisiéramos un resumen del resultado del build, existe una interfaz que nos da aquello. En la siguiente imagen vemos que se hizo un build y este tuvo errores en sus tests.

Figura 3.4: Resumen de Build con errores

Luego podemos ver que en la siguiente versión de código se ha corregido el error al parecer y este nos da como resultado un build exitoso.

Figura 3.5: Resumen de Build exitoso

3.4. Cambios entre versiones

Si quisiéramos ver el resultado de cambios hechos en esa diferencia de historias de versionamiento, en el caso de que tengamos errores de builds, tendremos una interfaz similar a la siguiente imagen:

Figura 3.6: Resumen de cambios de versiones con errores

Si por el contrario nuestro resultado fue exitoso, la interfaz sería algo similar a la siguiente imagen.

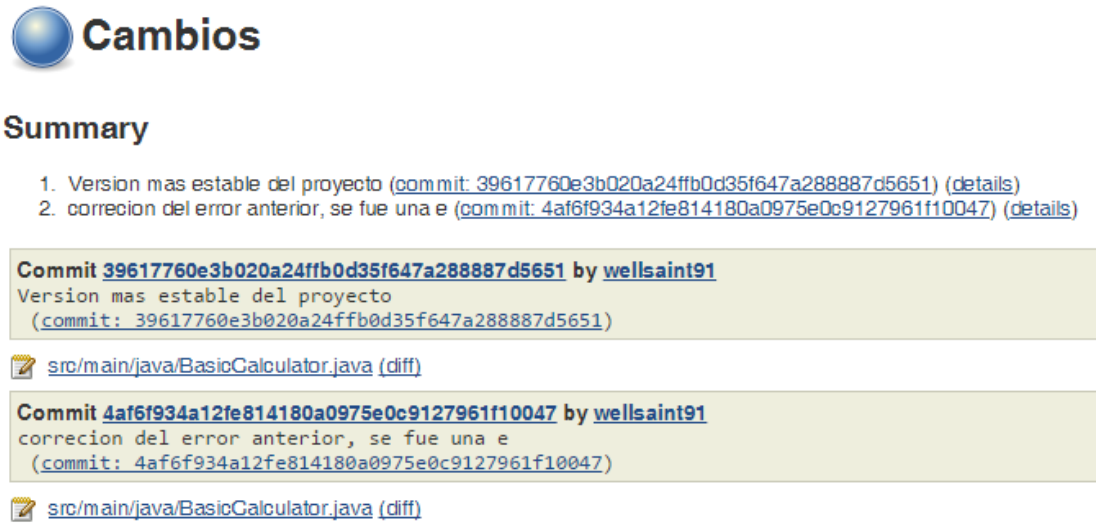

Figura 3.7: Resumen de cambio de versiones exitoso

3.5. Histórico de Builds

Si entramos al visor del proyecto, del lado izquierdo tendremos un panel con un historial de builds.

Figura 3.8: Histórico de Builds

Y en el dashboard principal tendremos una interfaz como la siguiente imagen:

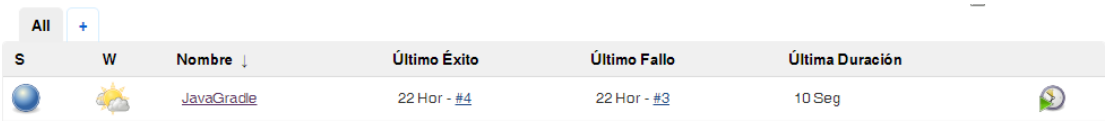

Figura 3.9: Dashboard Principal

3.1. Tabla de resultados

N° Prueba	Efectividad	Tiempo	Prueba realizada
1	OK	16 min	Se descargan todas las dependencias, al ser la primera vez que se ejecuta la tarea. No existe error de código.
2	OK	5.9 sec	No se realizan cambios, por lo cual la tarea se ejecuta más rápido.
3	Error	9.7 sec	Se modifica el código, haciendo un error voluntario en la lógica del mismo.
4	OK	10 sec	Se corrige el error de la construcción anterior

Tabla 1 - Tabla de Resultados

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1. La Integración continua permite a los desarrolladores entregar código probado al usuario final.
2. Las empresas de desarrollo que implementan integración continua, mejoran la entrega de código en tiempo y eficiencia.
3. La Integración Continua ayuda a que se desarrolle software colaborativo con metodologías Agiles de forma que aumente la eficiencia.
4. El ambiente de integración continua de Jenkins permite observar una a una las pruebas que se definen para verificar efectividad de la lógica del código.
5. Se pueden observar en Jenkins los errores que se puedan producir por problemas de código. Indicando al desarrollador que test no fue satisfactorio.
6. Jenkins tiene una interfaz web, la cual nos permite revisar cada uno de los procesos que este realiza, desde la construcción de código, los test, y finalmente los errores que se puedan producir.

RECOMENDACIONES

1. Se recomienda que un proyecto con Integración Continua use un repositorio central de versiones.
2. Se recomienda que un proyecto con Servidor de Integración Continua sea acompañada de toda una Metodología en la cual se involucre desarrolladores, operativos, y se establezcan políticas de entrega de código de acuerdo a alguna Metodología Agile.

BIBLIOGRAFIA

[1] C. ARON COIS, "Continuous Integration in DevOps", SEI Insights, 2016. [Online]. Available: https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html. [Accessed: 27- Jan- 2016].

[2] J. Langr, T. Ottinger and S. Pfalzer, Agile in a flash. [Raleigh]: Pragmatic Bookshelf, 2011.

[3] Thoughtworks.com, "Continuous Integration | ThoughtWorks", 2016. [Online]. Available: https://www.thoughtworks.com/es/continuous-integration. [Accessed: 27- Jan- 2016].

[4] Wikipedia.com, "Continuous Integration - Wikipedia", 2016. [Online]. Available: https://en.wikipedia.org/wiki/Continuous_integration. [Accessed: 27- Jan- 2016].

[5] martinfowler.com, "Continuous Integration", 2016. [Online]. Available: http://www.martinfowler.com/articles/continuousIntegration.html. [Accessed: 27- Jan- 2016].

[6] Wiki.jenkins-ci.org, "Home - Jenkins - Jenkins Wiki", 2008. [Online]. Available: https://wiki.jenkins-ci.org/display/JENKINS/Home. [Accessed: 27- Jan- 2016].

[7] Vagrantup.com, "Documentation - Vagrant by HashiCorp", 2016. [Online]. Available: https://www.vagrantup.com/docs/. [Accessed: 27- Jan- 2016].

[8] J. Smart, BDD in action. Shelter Island, N.Y.: Manning, 2015.

[9] A. Weijnitz, "Jenkins build server with a Vagrant Box", Automation.better-than.tv, 2016. [Online]. Available: http://automation.better-than.tv/. [Accessed: 27- Jan- 2016].

[10] V. Subramaniam and A. Hunt, Practices of an agile developer. Raleigh, N.C.: Pragmatic Bookshelf, 2006.

GLOSARIO

Términos generales de programación

Linting: Surgió originalmente en linux como una utilidad para hacer revisión del código, lo que hace es leer el contenido del archivo como texto, y revisar que siga un estilo de programación definido, además dependiendo de la herramienta podría identificar malas prácticas de programación, y posibles errores futuros.

https://en.wikipedia.org/wiki/Lint_(software)

Backend: Programación servidor, se refiere a la programación de interfaces publicas para interacción con un sistema que generalmente tiene conexión a un recurso de base de datos.

https://en.wikipedia.org/wiki/Front_and_back_ends

Frontend: Programación cliente, este término se usa mayormente en ambientes de desarrollo web, y se entiende que es programación que interactúa directamente con el usuario, generalmente Interfaz Gráfica de Usuario (GUI).

https://en.wikipedia.org/wiki/Front_and_back_ends

Build: Se refiere a la construcción de software, es decir; poder levantar un sistema con pasos simples que no dependan de un desarrollador.

https://en.wikipedia.org/wiki/Software_build

Fork: Es la copia independiente de un proyecto.

https://en.wikipedia.org/wiki/Fork_(software_development)

Términos usados en Administración de configuraciones

Ansible: Herramienta para automatizar configuraciones de servidor.

https://en.wikipedia.org/wiki/Ansible_(software)

Chef: Herramienta para automatizar configuraciones de servidor.

https://en.wikipedia.org/wiki/Chef_(software)

Puppets: Herramienta para automatizar configuraciones de servidor.

https://en.wikipedia.org/wiki/Puppet_(software)

Términos usados en Java

Jvm: Java virtual machine, plataforma que transcribe el resultado de un lenguaje de programación de alto nivel a un bajo nivel para la máquina.

https://en.wikipedia.org/wiki/Java_virtual_machine

Java: Lenguaje de programación mayormente conocido por ser de sintaxis estricto, además de su paradigma de programación imperativa, y su influencia sobre programación orientada a objetos.

https://en.wikipedia.org/wiki/Java_(software_platform)

Groovy: Lenguaje de programación, conocido por ser su facilidad de sintaxis parecida a ruby, y su multiparadigma que incluye programación declarativa y funcional.

https://en.wikipedia.org/wiki/Groovy_(programming_language)

Maven: Herramienta de build y manejador de dependencias basado en java, bastante popular en proyectos modernos de java.

https://en.wikipedia.org/wiki/Apache_Maven

Gradle: Herramienta de build y manejador de dependencias basado en groovy, es el builder de preferencia para proyectos basados en android.

https://en.wikipedia.org/wiki/Gradle

Ant: Herramienta de build para levantar proyectos, ant está escrito en java, surgió como un reemplazo al comando make de linux que causaba problemas en ciertos proyectos de java.

https://en.wikipedia.org/wiki/Apache_Ant

Términos usados en Php

Composer: Herramienta de build y manejador de dependencias basado en php, popular en la comunidad php, y programación moderna usando php.

https://en.wikipedia.org/wiki/Composer_(software)

Términos usados en Javascript

EcmaScript: Nombre clave para referirse a javascript (como es conocido popularmente).

https://en.wikipedia.org/wiki/ECMAScript

Node.js: Ambiente de desarrollo servidor basado en javascript.

https://en.wikipedia.org/wiki/Node.js

Minificado: Se usa comúnmente en programación javascript, significa que va a leer el contenido de un archivo, y quitar espacios blancos ej: espacios, tabs, etc, con la intensión de reducir el peso del archivo.

https://en.wikipedia.org/wiki/Minification_(programming)

Grunt: Herramienta de build y ejecutor de tareas de minificado y ofuscacion en javascript para frontend, conocido principalmente por ser una de las primeras soluciones en aparecer para javascript para realizar este tipo de tareas.

http://gruntjs.com/

Bower: Herramienta de build y manejador de dependencias basado en node.js para descargar dependencias en frontend, conocida por su creador Twitter.

https://en.wikipedia.org/wiki/Bower_(software)

Gulp: Herramienta de build y manejador de dependencias basado en javascript para frontend, considerada competencia de bower.

https://github.com/vigetlabs/gulp-starter/wiki/What-is-Gulp%3F

Browserify: Herramienta de build y manejador de dependencias basado en javascript para frontend y backend.

https://en.wikipedia.org/wiki/Browserify

Webpack: Herramienta de build y manejador de dependencias basado en javascript para frontend y backend.

https://en.wikipedia.org/wiki/Webpack