Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Entorno con integración continua para aplicaciones web desarrolladas con AngularJS

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Francisco Fuster Just

Tutor: Germán Moltó Martinez

2020/2021

Resumen

En este trabajo de fin de grado se presenta y analiza un entorno de trabajo con integración continua para el desarrollo de aplicaciones web creadas con AngularJS. En primer lugar he decidido utilizar sistema de control de versiones distribuido GitLab. Dentro de GitLab tendremos ramas correspondientes a versiones diferentes del proyecto; por ejemplo, versión cliente y versión desarrollo. A su vez, cada rama aloja los ficheros correspondientes a la versión de nuestra aplicación web que elaboramos con AngularJS y los ficheros de test correspondientes para los tests unitarios de Karma, pruebas end-to-end de Protractor y las pruebas de rendimiento de JMeter. En segundo lugar utilizaremos contenedores Docker y dentro de la imagen Docker alojaremos el servidor Jenkins, que ayudará a la construcción, implantación y automatización del proyecto. Jenkins se encargará de hacer la descarga del repositorio Git y de ejecutar los tests automáticos. Estas pruebas se ejecutarán para evaluar cada parte de la aplicación y verificar que el comportamiento es el esperado. Jenkins también se encargará de enviar los reportes en caso de problemas y de hacer el despliegue de la web una vez esté todo correcto y funcionando. Finalmente, después de presentar todo lo que vamos a utilizar, se pretende que este entorno de desarrollo ayude a mejorar la calidad y los tiempos de producción de software de la empresa en la que se ha desarrollado este trabajo.

Palabras clave: Integración continua, sistema de control de versiones, AngularJS, GitLab, Docker, Jenkins, Karma, Protractor, JMeter.

Abstract

In this final degree project, a work environment with continuous integration for the development of web applications created with AngularJS is presented and analyzed. First of all, I have decided to use the distributed version control system GitLab projects. Within GitLab we will have branches corresponding to different versions of the project; for example, client version and development version. In turn, each branch houses the files corresponding to the version of our web application that we elaborate with AngularJS and the corresponding test files for the Karma unit tests, Protractor end-to-end tests and the JMeter performance tests. Secondly, we will use Docker containers and within the Docker image we will host the Jenkins server, which will help with the construction, implementation and automation of the project. Jenkins will take care of downloading the Git repository and running the automatic tests. These tests will be run to evaluate each part of the application and verify that the behavior is as expected. Jenkins will also be in charge of sending the reports in case of problems and of making the web deployment once everything is correct and working. Finally, after presenting everything that we are going to use, this development environment is intended to help improve the quality and production times of the software of the company in which this work has been developed.

Keywords : Continuous integration, version control system, AngularJS, GitLab, Docker, Jenkins, Karma, Protractor, JMeter.

Tabla de contenidos

1. Introducción

Este trabajo describe el estudio y mejora del entorno de desarrollo utilizado en la empresa ODEC Centro de Cálculo y Aplicaciones Informáticas, S.A.[1]. Fue la empresa donde realicé las prácticas universitarias, además de ser el lugar donde trabajé durante dos años en el departamento de informática como Web tester y en el equipo de Front-End

Durante el periodo que estuve trabajando en la empresa, se me propuso el estudio de nuevas metodologías de desarrollo, entre ellas, metodologías ágiles que incorporen la integración continua. De esta forma, la empresa quería mejorar los problemas que con los que se encontraba a la hora de mantener un correcto desarrollo y que esta metodología llegara a implantarse en la empresa como el modelo de desarrollo a seguir en los futuros proyectos.

1.1 ODEC

ODEC es una empresa con más de 50 años de experiencia que fue pionera en la gestión integral de datos.

En la actualidad ODEC ofrece servicios informáticos especializados para investigación de mercados, estadística, medios y marketing, ofreciendo como principal valor diferencial la integración de cinco áreas fundamentales [1]:

- Desarrollo de software.
- Presentación de resultados.
- Captura de datos.
- Tratamiento de información.
- Subcontratación de servicios externos.

En concreto, el departamento de informática donde estuve trabajando se dedica a desarrollar plataformas web para el sector de la automoción donde se publican indicadores sobre estudios de calidad de servicio, estudios de satisfacción del cliente con alertas del grado de respuestas y estudios de satisfacción sobre la web de la marca, entrega vehículo y posventa.

[1] https://www.odec.es/es/

1.2 Objetivo

En este Trabajo Final de Grado se realiza un análisis de las estrategias, herramientas y buenas prácticas para desarrollo de código con el objetivo de proponer una metodología ágil con arquitectura de componentes que permita la integración continua para facilitar el trabajo de las personas que están trabajando en los diferentes desarrollos.

Con la integración continua se mejorará de forma notable la creación de código por parte de diferentes desarrolladores ya que no es necesario que el código esté finalizado como se hacía con las metodologías tradicionales. Por el contrario, la creación de código se divide en tareas más pequeñas donde el tiempo a emplear es menor y es más fácil comprobar frecuentemente estas tareas con la ayuda de tests automáticos.

Una metodología ágil implica trabajar de forma más rápida donde se quiere reducir el riesgo de cometer errores. Para ello es necesario el trabajo en equipo, de forma que se ayuden unos a otros y lograr superar los problemas de desarrollo. Además de separar los grandes proyectos en tareas más pequeñas para que el trabajo sea mucho más llevadero, la presión es menor y los tiempos de entrega más cortos.

Estas mejoras implican mantener un repositorio de código fuente único que ayude a la gestión del código, sobre todo cuando hay diferentes equipos y personas involucradas en el mismo proyecto. Este repositorio nos permitirá tener disponibilidad constante de las diferentes versiones del proyecto y además nos permitirá crear diferentes ramas para tener diferentes corrientes de desarrollo.

En segundo lugar, reducir el riesgo porque a partir de ahora se podrán detectar los fallos de forma temprana y así poder ahorrar mucho más tiempo que si se hubiese detectado más tarde. Además vamos a conocer en todo momento cual es el estado de nuestro código, por lo que a diario se podrá saber la salud de la aplicación.

Seguidamente, automatizar la compilación de las aplicaciones web ya que una compilación frecuente permite definir nuevas características más rápidamente, dar retroalimentación sobre estas características y en general tener un equipo de trabajo más colaborativo en el ciclo de desarrollo.

Para encontrar los errores, se diseñarán tests automáticos que se ejecutarán en la fase de construcción. Gracias a ellos se podrá cancelar el despliegue y gracias a sus informes se podrán detectar los errores mucho más rápido.

Con estas mejoras se pretende que se rompan las barreras entre clientes y desarrolladores, ya que el cliente puede ver con más frecuencia el estado del desarrollo y de esta forma ponerse en contacto más rápidamente con los desarrolladores ya que ahora las instalaciones serán más frecuentes y el estado del proyecto será más fácil de ver.

1.3 Estructura del documento

La estructura de este Trabajo de Fin de Grado está dividida en cuatro secciones.

En primer lugar, se describen las metodologías que existen actualmente en el proceso de desarrollo de software. También se mencionan diferentes herramientas que se pueden utilizar para Continuous Integration and Continuous Delivery (CI/CD), como los sistemas de control de versiones y la virtualización en contenedores. Además, describen la funcionalidad de los ensayos automatizados en el proceso de CI/CD.

En el segundo capítulo se define en detalle la plataforma diseñada para permitir la integración continua (CI), explicando en qué consiste el desarrollo web, la metodología que se ha adoptado y el lenguaje de programación utilizado, en nuestro caso AngularJS. Mediante el diagrama de la solución en la empresa, se describen las herramientas principales utilizadas para este fin como Jenkins que se utiliza para la automatización de procesos, Docker para la virtualización de contenedores y GitLab, como herramienta para compartir código y lanzar las ejecuciones. Finalmente este capítulo termina con la descripción de las pruebas automáticas utilizadas.

El tercer capítulo consiste en las contribuciones personales que he realizado en la empresa. De esta forma se explican los tests desarrollados de Protractor y JMeter además de explicar su configuración y otros usos para los que se han llevado estos tests.

Por último, el cuarto capítulo corresponde a la conclusión obtenida del trabajo, a las limitaciones que nos hemos encontrado y a los trabajos futuros a realizar.

2. Estudio del Arte

En este capítulo se describen y analizan las buenas prácticas para el desarrollo del software y con ello las diferentes metodologías que existen para este fin.

Como se quiere adoptar una metodología ágil, se nombran las principales herramientas y competencias que existen para su uso, como son los sistemas de control de versiones. Seguidamente se explica el concepto de Integración continua,las herramientas y prácticas utilizadas para este fin.

2.1 Buenas prácticas para el desarrollo de software

Debido a que hoy en día no existe un consenso generalizado sobre las mejores herramientas o métodos que describan los requisitos y patrones a utilizar en el desarrollo del software, la gran parte de los proyectos de desarrollo de software terminan sin completarse de forma correcta, o sin éxito[2]. Es por este motivo que a continuación se detalla una lista con las buenas prácticas para el proceso de desarrollo de software.

1. **Modelos de desarrollo:** es importante elegir el modelo de desarrollo apropiado para el proyecto en cuestión porque todas las demás actividades se derivan del modelo. En la mayoría de los proyectos de desarrollo de software modernos, se utiliza algún tipo de metodología basada en espiral o en cascada [3]. Las metodologías utilizadas comúnmente que se describirán posteriormente contienen orientación sobre cómo ejecutar el proceso.

2. **Requisitos:** reunir y acordar requisitos es fundamental para un proyecto exitoso. Esto no implica necesariamente que todos los requisitos deban solucionarse antes de que se realice la arquitectura, el diseño y la implementación, pero es importante que el equipo de desarrollo entienda lo que se debe construir.

   Los requisitos de calidad se dividen en dos tipos: funcional y no funcional.

   - Los requisitos funcionales son declaraciones sobre los servicios que proporcionará el sistema y cómo reacciona a determinadas entradas.

   - Los requisitos no funcionales describen el rendimiento y las características del sistema de la aplicación. Es importante reunirlos porque tienen un gran impacto en la arquitectura, el diseño y el rendimiento de la aplicación.

3. **Arquitectura:** la clave es elegir la arquitectura adecuada para la aplicación. Se selecciona y diseña una buena arquitectura de software de acuerdo con los objetivos (requisitos) y las limitaciones.

- Un objetivo es una meta predeterminada para un sistema de información, pero no solo es una meta funcional, sino también otras metas, como la capacidad de mantenimiento, la auditabilidad, la flexibilidad y la interacción con otros sistemas de información.

- Las limitaciones son aquellas que se derivan de la tecnología que se puede utilizar para implementar el sistema de información.

Algunas arquitecturas son más recomendables para usar ciertas tecnologías, mientras que otras tecnologías no son adecuadas para ciertas arquitecturas. Por ejemplo, no es factible utilizar una arquitectura de software de tres niveles para implementar un sistema en tiempo real.

4. **Diseño:** Incluso con una buena arquitectura puede haber un mal diseño. Muchas aplicaciones tienen un diseño excesivo o insuficiente. El diseño en el desarrollo de software incluye una descripción escrita del producto de software. El diseñador escribe esta descripción para proporcionar al equipo de desarrollo de software el posicionamiento general de la arquitectura del proyecto de software. Por lo general, va acompañado de un diagrama de arquitectura, que contiene indicadores que detallan las especificaciones de los componentes de diseño.

5. **Construcción del código:** la construcción del código es una parte del esfuerzo total del proyecto, que a menudo es lo más visible. Otros trabajos igualmente importantes incluyen requisitos, arquitectura, análisis, diseño y prueba. En proyectos sin proceso de desarrollo (el llamado "código y corrección"), estas tareas también están ocurriendo, pero bajo el pretexto de la programación. Una buena práctica para construir el código incluye la compilación diaria y un testeo. Esta práctica es conocida como la integración continua que también integra el concepto de pruebas unitarias y código de autoprueba. Aunque la integración continua y las pruebas unitarias han ganado popularidad, se pueden usar estas prácticas en todo tipo de proyectos.

6. **Pruebas:** es una parte primordial en el desarrollo de software que debe planificarse. Las pruebas proactivas también son importantes; esto significa planificar casos de prueba antes de comenzar a codificar y desarrollar casos de prueba mientras se diseña y codifica la aplicación. Estos casos de prueba, se pueden programar mediante **tests de End to End** (E2E) y con **tests unitarios**.

7. **Pruebas de rendimiento:** las pruebas de rendimiento suelen ser el último recurso para detectar defectos de la aplicación. Es laborioso y, por lo general, sólo se detectan los problemas de codificación. Los problemas de arquitectura y diseño pueden pasarse por alto. Un método para detectar algunos defectos arquitectónicos es simular las pruebas de carga en la aplicación antes de implementarla y tratar de encontrar los problemas de rendimiento antes de que se conviertan en problemas más graves. Estas pruebas de carga pueden ser implementadas mediante **JMeter.**

8. **Administración del sistema:** la administración implica conocer el estado de todos las máquinas que conforman su sistema o proyecto, administrar el estado de estas máquinas y lanzar versiones distintas de un sistema.

9. **Implantación:** la implantación es la etapa final de lanzamiento y puesta en marcha de una aplicación. Sin embargo, todavía hay cosas que pueden salir mal. Se debe planificar la implantación.

10. **Operaciones y soporte del sistema:** El área de soporte es un factor vital para responder y resolver los problemas de los usuarios. Para facilitar el flujo de problemas, se puede usar un sistema de seguimiento e incidencias de defectos para la aplicación.

11. **Gestión de proyectos:** la gestión de proyectos es clave para un proyecto exitoso. Muchas de las otras áreas de mejores prácticas descritas están relacionadas con la gestión de proyectos y un buen gerente de proyectos ya está al tanto de la existencia de estas mejores prácticas. Es sorprendente la cantidad de gerentes de proyectos que no los conocen y no aplican las lecciones aprendidas de proyectos anteriores.

2.2 Modelos de desarrollo de software

En el proceso de desarrollo de software se pueden utilizar diferentes metodologías que sirven para estructurar, planificar y controlar el proceso de desarrollo de aplicaciones software. Estas metodologías, pueden ser adaptadas y modificadas según las necesidades del software durante el proceso de desarrollo [4].

Los modelos que se describen a continuación comparten alguna de las siguientes áreas: Análisis del problema, investigación de mercado, recopilación de requisitos, diseño para la solución basada en software, implementación del software, testeo del software, despliegue, mantenimiento y corrección de errores.

2.2.1 Modelo en cascada

Conocido como el modelo tradicional, su nombre viene definido por la posición que adoptan las fases de desarrollo. Estas fases de desarrollo tienen una dependencia lineal y secuencial donde la siguiente fase siempre depende de la anterior.

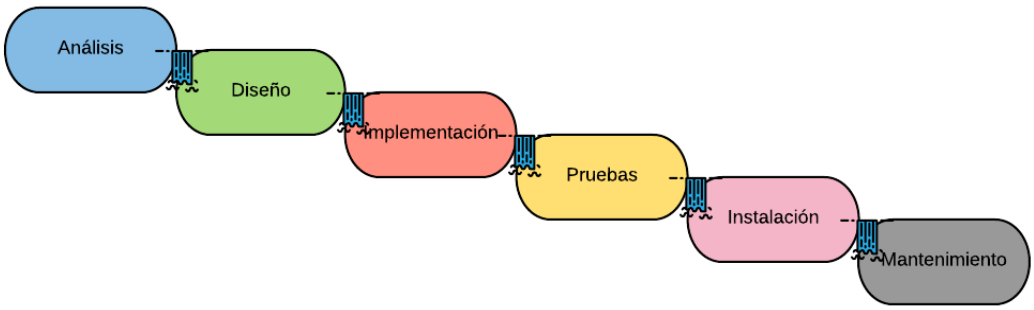


Figura 2.1 Modelo en cascada

Ventajas:

- Control. Después de cada fase, se puede realizar una revisión antes de empezar la siguiente.
- Bajo coste en la planificación del proyecto.

Desventajas:

- Participación limitada de los usuarios en el desarrollo del proyecto y se requiere mucho tiempo para su desarrollo.
- El coste puede ser muy elevado en caso de detectar un problema después de la instalación.

2.2.2 Modelo de prototipo

El modelo de prototipo sirve para mostrar al cliente una vista o prototipo funcional de parte del software mientras el producto se encuentra en fase de desarrollo.
Este modelo es básicamente una muestra que puede ser rechazada ya que si al cliente no le gusta una parte del prototipo, significa que la prueba ha fallado, por lo que se debe corregir y hacer un nuevo prototipo hasta que el cliente quede satisfecho.

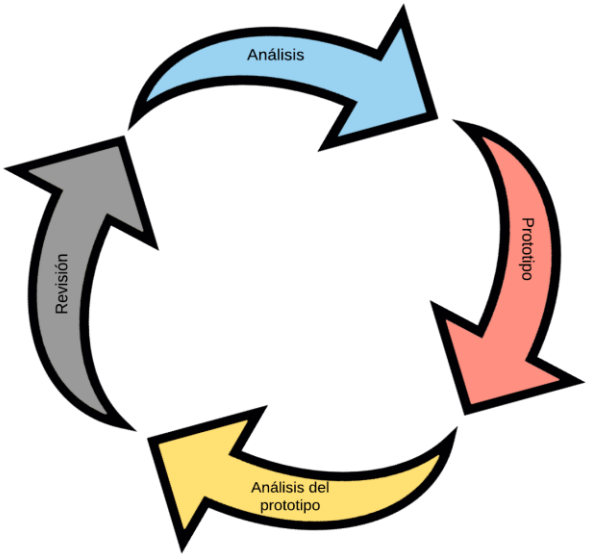


Figura 2.2 Modelo de prototipo

Ventajas:

- Desde un principio se dan a conocer los requerimientos que necesita el desarrollo.
- El desarrollador es consciente de lo que requiere el cliente.
- Facilita que los desarrolladores se percaten de cómo está avanzando el proyecto.

Desventajas:

- Dificultad en la administración del desarrollo. Es muy difícil mantener el propósito inicial.
- Se suele considerar al prototipo como desarrollo final cuando aún está en fase de desarrollo e incompleto.
- Aparecen imprevistos que retrasan el desarrollo del software.

2.2.3 Modelo de desarrollo incremental

El modelo de desarrollo incremental consiste en una combinación de elementos del modelo en cascada con el modelo de prototipos. Se basa en ir incrementando el contenido del modelo fase por fase. Este modelo aplica secuencias lineales de forma escalonada mientras progresa en el tiempo. Cada secuencia lineal produce un incremento del software (diseño, implementación y fase de pruebas).

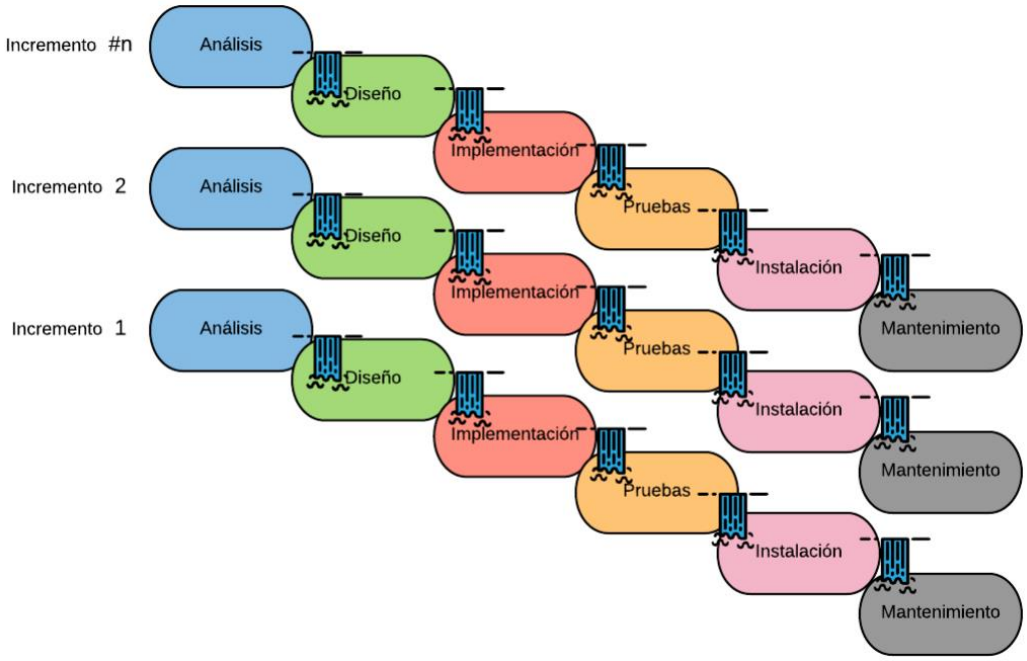


Figura 2.3 Modelo incremental

Ventajas:

- Al igual que en el modelo en cascada, después de cada iteración se puede hacer una revisión para comprobar que todo ha ido correctamente, y se puede arreglar los problemas que hayan aparecido.
- El cliente puede responder a los cambios y analizar constantemente el producto para posibles cambios.

Desventajas:

- El producto resultante puede ser mucho más caro de lo previsto.
- Mientras se va progresando, pueden aparecer problemas de estructura no presentes en prototipos de fases anteriores.
- En cada iteración hay una fase que es inflexible y no se solapa con otras.

2.2.4 Modelo de desarrollo iterativo

El modelo de desarrollo iterativo es un modelo derivado del modelo de desarrollo en cascada, tiene como objetivo disminuir la forma de malinterpretar los riesgos entre el usuario y el producto final durante el proceso de recopilación de condiciones.

Se basa en realizar varias iteraciones del ciclo de vida en cascada, donde al final de cada iteración, se le concede al cliente una versión mejorada con las funciones deseadas. El cliente es la persona que evalúa el producto y lo corrige o proporciona sugerencias de mejora después de cada iteración. Estas iteraciones se repetirán hasta que se obtenga un producto que cumpla con los objetivos del cliente.

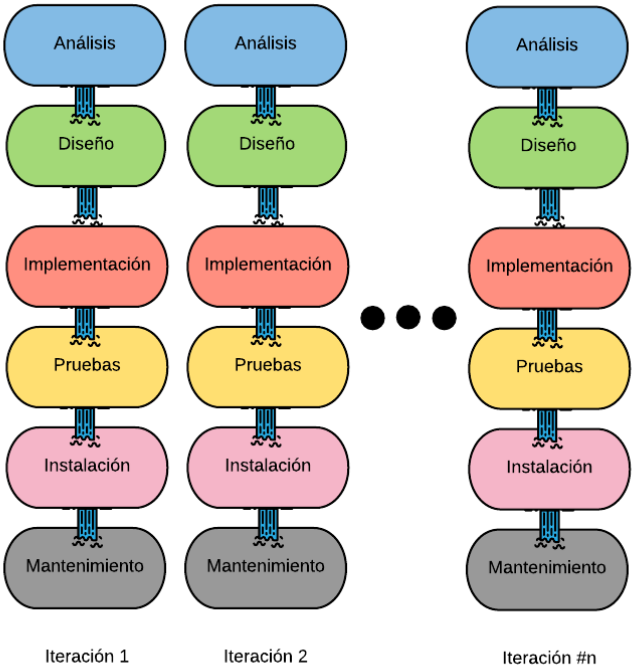


Figura 2.4 Modelo iterativo

Ventajas:

- No es necesario que los requisitos estén definidos completamente al comienzo del desarrollo. Se pueden ir mejorando en cada una de las iteraciones.
- Ventajas propias de contribuir a un desarrollo de ciclos pequeños, lo que facilita el control de los riesgos y las entregas.

Desventajas:

- Como no son necesarios tener definidos los requisitos, pueden surgir problemas relacionados con la arquitectura.

2.2.5 Modelo de desarrollo en espiral

La fase de desarrollo de este modelo se basa en una espiral y cada ciclo o iteración representa un conjunto de actividades. Estas actividades se seleccionan en función de los factores de riesgo que representan, después del ciclo anterior.

Se combinan aspectos claves del modelo de cascada y el modelo rápido de aplicaciones para intentar aprovechar las ventajas que aportan los dos modelos.

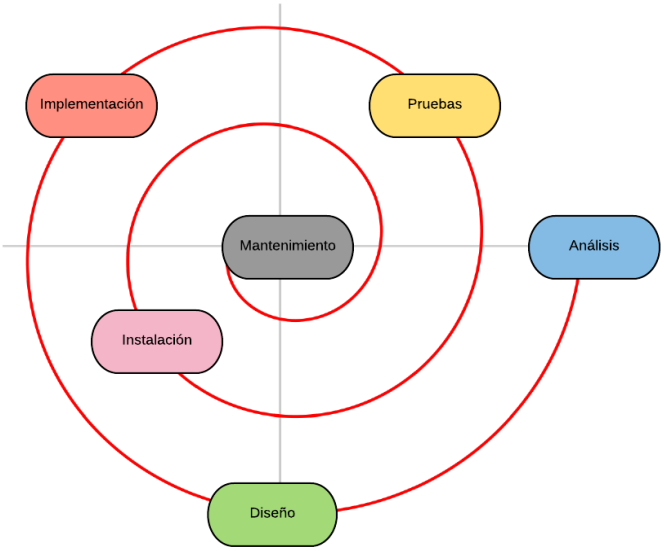


Figura 2.5 Modelo en espiral

Ventajas:

El modelo en espiral se puede ajustar y aplicar durante todo el ciclo de vida del software.

- A medida que el software evoluciona con el progreso del proceso, los desarrolladores y los clientes pueden comprender y reaccionar mejor a los riesgos de cada nivel de evolución.El modelo en espiral permite a los desarrolladores aplicar métodos de prototipos en cualquier etapa de la evolución del producto.

Desventajas:

- Es difícil convencer a los clientes que han encargado el desarrollo de que el desarrollo en espiral es controlable.
- Debido a su alta complejidad, no se recomienda su uso en sistemas pequeños.
- La mayor parte del tiempo se consume en la implementación.

2.2.6 Modelo de desarrollo rápido de aplicaciones

El modelo de desarrollo rápido de aplicaciones (Rapid Application Development - RAD) combina el desarrollo iterativo con la rápida construcción de prototipos en vez de planificaciones a largo plazo. La falta de esta planificación normalmente permite que se escriba mucho más rápidamente el software, y hace más fácil el cambio de requerimientos.

El proceso de desarrollo rápido comienza con el desarrollo de modelos de datos preliminares y la utilización de técnicas estructuradas para los modelos de

procesamiento. En la siguiente fase, los requerimientos son verificados mediante prototipos. Estas fases se repiten de manera iterativa.

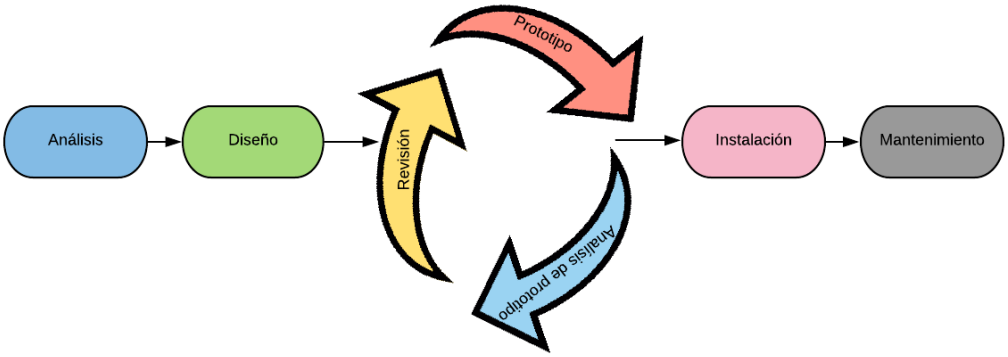


Figura 2.6 Modelo de desarrollo rápido

Ventajas:

- Control de riesgos.
- Mejor calidad y la relación entrega-coste-calidad (entregar el proyecto dentro de los plazos al menor coste posible manteniendo la calidad).

Desventajas:

- Si los clientes y desarrolladores no se comunican y entienden bien las actividades para la construcción del sistema, los proyectos no tendrán éxito.
- Tiene un alto coste de equipo necesario y herramientas integradas.
- Proceso más difícil de medir.

2.2.7 Modelo de desarrollo ágil

En este modelo, nos referimos a un conjunto de métodos de desarrollo de software basados en la iteración, en el que los requisitos y las soluciones se resuelven a través de la colaboración entre equipos organizados.

El desarrollo iterativo se utiliza como base para defender una visión más relajada y centrada en el ser humano que las soluciones tradicionales.

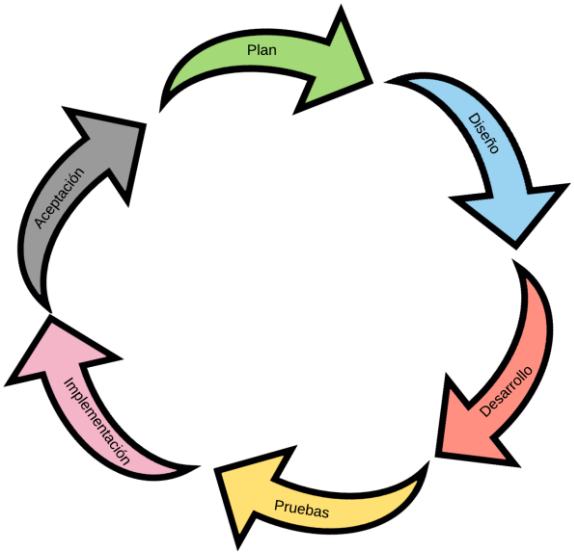


Figura 2.7 Modelo de desarrollo ágil

Los procesos ágiles utilizan la retroalimentación en lugar de la planificación como principal mecanismo de control. Esta retroalimentación se proporciona a través de pruebas periódicas y versiones de software frecuentes.

Ventajas:

- El cliente, al poder observar cómo se va construyendo el proyecto, puede opinar sobre su evolución gracias a las reuniones que se realizarán con el equipo de desarrollo. Esto le facilita al cliente cierta tranquilidad.
- Evita la mala comprensión de requisitos entre desarrolladores y cliente.
- Continua mejora de los procesos y el equipo de desarrollo.
- Cada componente del producto final se prueba para ver si cumple con los requisitos.

Desventajas:

- El problema surge del fracaso de los proyectos ágiles. Si un proyecto ágil falla, hay poca o ninguna documentación, al igual que el diseño.
- Dado que el proyecto se encuentra en una etapa de desarrollo continuo, depende en gran medida de los clientes y desarrolladores.
- Falta de reutilización por falta de documentación. Restricciones a la escala del proyecto.
- La comprensión del sistema permanece en la mente del desarrollador.
- Encontrar errores tardíos entre requisitos y soluciones.

2.3 Sistemas de control de versiones

Actualmente los sistemas de control de versiones (VCS) se utilizan para registrar los cambios llevados a cabo en archivos o colecciones de archivos en el paso del tiempo para poder restablecer versiones específicas más adelante. Los sistemas de control de versiones han evolucionado con el paso del tiempo y se pueden dividir en tres tipos: Sistemas de Control de Versiones Locales, Sistemas de Control de Versiones Centralizados y Sistemas de Control de Versiones Distribuidos [5].

2.3.1 Sistemas de control de versiones locales

Una manera simple de usar un sistema de control de versiones es copiar los archivos a otro directorio (preferiblemente con la fecha y la hora en que fueron creados).

Esta solución es usada numerosas veces porque es muy fácil de usar, pero también extremadamente propensa a errores. Es fácil olvidar en qué directorio se encuentran guardados los archivos y además se puede guardar por error en el archivo o carpeta que no corresponde.

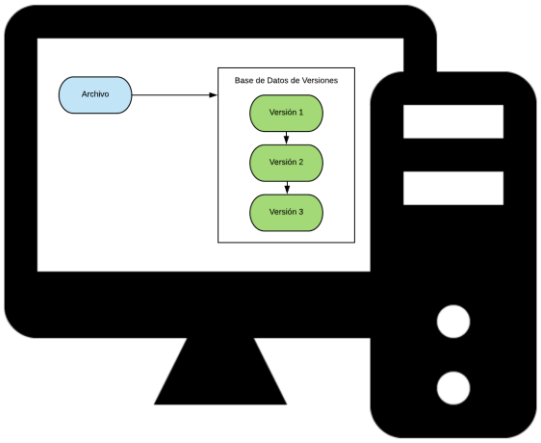


Figura 2.8 Sistema de control de versiones local

Una manera de resolver este problema, es usar una herramienta que los programadores desarrollaron hace mucho tiempo, como es el sistemas de control de versiones local que incluye una base de datos simple que realiza un seguimiento de todos los cambios en los archivos.

Una de las herramientas de control de versiones más populares es un sistema llamado Revision Control System (RCS)[2] que a día de hoy encontramos en muchos ordenadores [6].

Incluso el sistema operativo macOS incluye comandos RCS cuando se instalan las herramientas de desarrollo

Revision Control System

El Revision Control System (RCS) es un conjunto de comandos que ayudan a mantener los sistemas software que consisten en muchas versiones y configuraciones bien organizados. La función principal de RCS es gestionar grupos de revisión. Un grupo de revisión es un conjunto de documentos de texto, llamados revisiones, que evolucionaron entre sí. Se crea una nueva revisión editando manualmente una existente. RCS organiza las revisiones en un árbol ancestral. La revisión inicial es la raíz del árbol, y los bordes del árbol indican desde qué revisión evolucionó. Además de administrar grupos de revisión individuales, RCS proporciona funciones de selección flexibles para componer configuraciones.

RCS también ofrece facilidades para combinar actualizaciones con modificaciones del cliente, para el desarrollo de software distribuido y para la identificación automática. La identificación es el "estampado" de revisiones y configuraciones con marcadores únicos. Estos marcadores son similares a los números de serie, que indican a los desarrolladores de software la versión en que se encuentran.

RCS está diseñado para entornos tanto de producción como experimentales. En los entornos de producción, los controles de acceso detectan conflictos de actualizaciones y evitan cambios superpuestos. En ambientes experimentales, donde los controles fuertes son contraproducentes, es posible aflojarlos.

Aunque RCS fue originalmente diseñado para programas, es útil para cualquier texto que se revise con frecuencia y cuyas revisiones anteriores se deban conservar. RCS se ha aplicado con éxito para almacenar texto fuente para dibujos, diseños VLSI[3], documentación, especificaciones, datos de prueba, formularios y artículos.

2.3.2 Sistemas de control de versiones centralizados

[2] RCS es un software de control de versiones que automatiza las tareas de guardar, recuperar, registrar, identificar y mezclar versiones de archivos.
[3] Integración a gran escala

En los proyectos de gran tamaño la gente necesita colaborar con los desarrolladores de contenido software en otros sistemas. Para resolver este problema, se desarrolló un sistema de control de versiones centralizado. Estos sistemas, como Concurrent Versions System (CVS), Subversion y Perforce, tienen un servidor que contiene todos los archivos de versión y varios clientes que descargan archivos desde esta ubicación central. Este ha sido el estándar para el control de versiones durante muchos años.

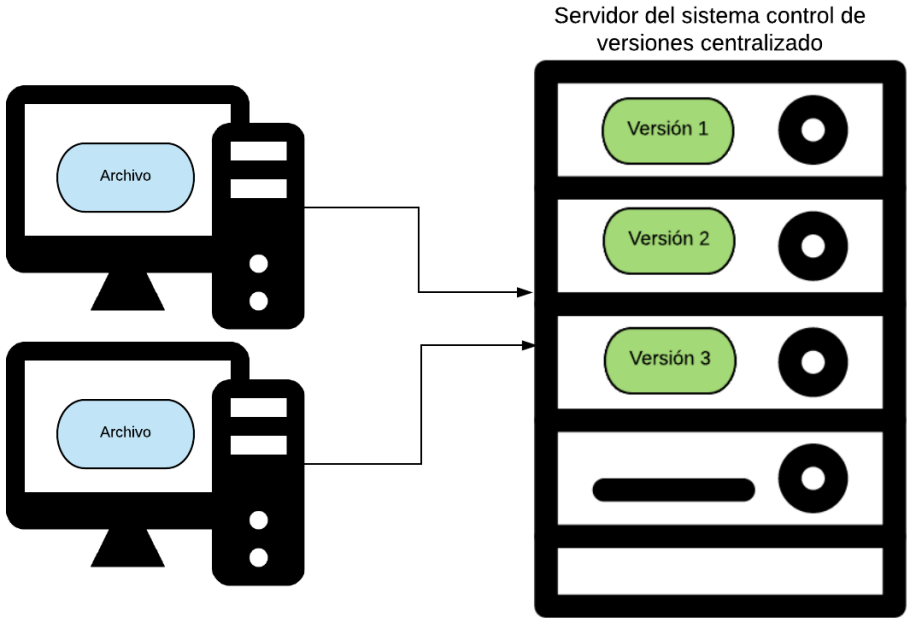


Figura 2.9 Sistema de control de versiones centralizado

Esta configuración ofrece muchas ventajas, especialmente en comparación con los sistemas de control de versiones locales. Por ejemplo, todo el mundo puede (hasta cierto punto) saber qué están haciendo los demás colaboradores del proyecto. Los administradores pueden controlar en detalle lo que todos pueden hacer; administrar un sistema de control de versiones centralizado es mucho más fácil que tratar con bases de datos locales en cada cliente.

Sin embargo, esta configuración también tiene graves inconvenientes. El más obvio es el único punto de fallo que representan los servidores centralizados. Si el servidor está inactivo durante una hora, nadie puede colaborar ni guardar cambios de versión del contenido en el que están trabajando durante esa hora. Si el disco duro en el que se encuentra la base de datos central está dañado y no se realiza una copia de seguridad adecuada, se perderá todo. El sistema de control de versiones local también se enfrenta al mismo problema.

CVS

El sistema de versión concurrente (CVS), es una aplicación informática que implementa un sistema de control de versiones, registra todo los trabajos y cambios realizados en los archivos que forman un proyecto, y permite que diferentes desarrolladores (posiblemente muy separados) contribuyan en él. CVS se ha vuelto

habitual en el campo del software libre. Esta herramienta está creada bajo la licencia de código abierto GPL[4].

Una arquitectura cliente-servidor, es la que utiliza el CVS. El servidor custodia la versión vigente del proyecto y el historial. El cliente se conecta a él para obtener una copia íntegra del proyecto. Esto se elabora con el fin de que provisionalmente puedan utilizar la copia y luego usar los comandos GNU[5] para ingresar sus cambios.

Normalmente, el cliente y el servidor utilizan la conexión a Internet, sin embargo con CVS, el servidor y el cliente pueden encontrarse en el mismo equipo. La tarea del sistema CVS es rastrear el historial de versiones del programa de un proyecto que donde se trabaja de manera local. Inicialmente, el servidor usaba un S.O. idéntico a Unix; no obstante, la versión CVS ahora existe en otros sistemas operativos, incluido Windows. El cliente que utiliza CVS lo puede ejecutar en cualquiera de los sistemas operativos más populares.

Diversos clientes logran hacer copias del proyecto al mismo tiempo. Más tarde, en el momento que se actualicen y modifiquen, el servidor intentará vincular los distintos modelos. Si surge algún defecto, por ejemplo, cuando dos clientes intentan modificar el mismo trozo de código de un archivo en concreto, el servidor rechazará la última adaptación y notificará al cliente la disputa que debe ser resuelta de forma manual por el usuario. Si la operación de entrada es exitosa, los números de versión de todos los archivos involucrados se intensificarán instantáneamente y el servidor CVS almacenará información sobre dicha actualización, incluida la explicación facilitada por el usuario, el nombre del autor, la fecha y su archivo de registro.

Los clientes también pueden cotejar las distintas versiones de archivos; solicitar un historial completo de cambios y obtener una "instancia" histórica del proyecto en una fecha específica o número de revisión.

Los clientes también pueden usar la secuencia de actualización para mantener su copia sincronizada con la última versión utilizada en el servidor, eliminando la obligación de descargar de forma reiterada todo el proyecto, solo lo actualizado.

CVS puede sostener diferentes "ramas" de un proyecto. Por ejemplo, en una distribución del proyecto software, puede crear una rama y usarse para solucionar errores. Esto se puede hacer con la nueva rama que se encuentra en desarrollo en la actualidad y contiene las modificaciones nuevas, formando así otra rama independiente.

[4] La Licencia Pública General, en inglés GPL General Public License.
[5] Comandos utilizados para el S.O Unix/Linux.

Apache Subversion

Apache Subversion (SVN) es un sistema de control de versiones de software distribuido con código abierto bajo la Licencia Apache [7].

Usa el criterio de revisión para almacenar los cambios completados en un repositorio. Entre dos revisiones, solo almacena el conjunto de modificaciones, mejorando de esta forma la utilización del espacio en disco. SVN facilita a los individuos hacer, copiar y eliminar carpetas con la misma facilidad que en un disco duro local. Es requisito utilizar buenas prácticas para administrar correctamente la edición del software generado.

Con Subversion se puede entrar al repositorio por medio de la red, lo que facilita que la gente lo use desde diferentes PCs. Hasta cierto punto, la aptitud de numerosas personas para cambiar y gestionar el mismo grupo de datos desde áreas separadas incentiva la colaboración. No es necesario que todo el desarrollo pase únicamente por el canal por el que deben atravesar todas las ediciones de software, por este motivo el desarrollo puede surgir más rápidamente. Gracias a que este trabajo se encuentra bajo el control de versiones, no debemos preocuparnos si se ve afectada la calidad del trabajo; si se ejecuta un cambio incorrecto en los datos, sólo requiere que se deshaga el cambio realizado.

Perforce

Perforce es un sistema de control de versiones que se ejecuta en modo cliente-servidor, es multiplataforma y tiene una interfaz gráfica avanzada, rápida e intuitiva. La posibilidad de hacer operaciones distribuidas asegura una gran escalabilidad y se adapta al desarrollo de las empresas internacionales [8].

Perforce es intuitivo y fácil de aprender. Ya sea usando: un intérprete de comandos, un cliente visual en Windows, macOS, Linux, los principales IDEs líderes o herramientas de desarrollo. Los desarrolladores pueden usar Perforce cómodamente con estas plataformas.

La salud y la evolución del código siempre es visible. Los desarrolladores pueden ver rápidamente el historial de cambios a nivel de carpeta y archivo. Además pueden obtener una vista previa de las imágenes, audios, videos y contenido web. Con esto los administradores de calidad y versiones pueden controlar el desarrollo y analizar los impactos y desviaciones.

2.3.3 Sistemas de control de versiones distribuidos

En un sistema de control de versiones distribuido (como Git, Mercurial o Bazaar) [9], los clientes no solo se descargan la última instancia del archivo, sino que copian el repositorio por completo. Por lo tanto, si el servidor falla y estos sistemas colaboran a través de él, cualquier repositorio de un cliente puede copiarse al servidor para restaurarlo. Se realiza una copia de seguridad completa con todos los datos cada vez que se baja una instancia.

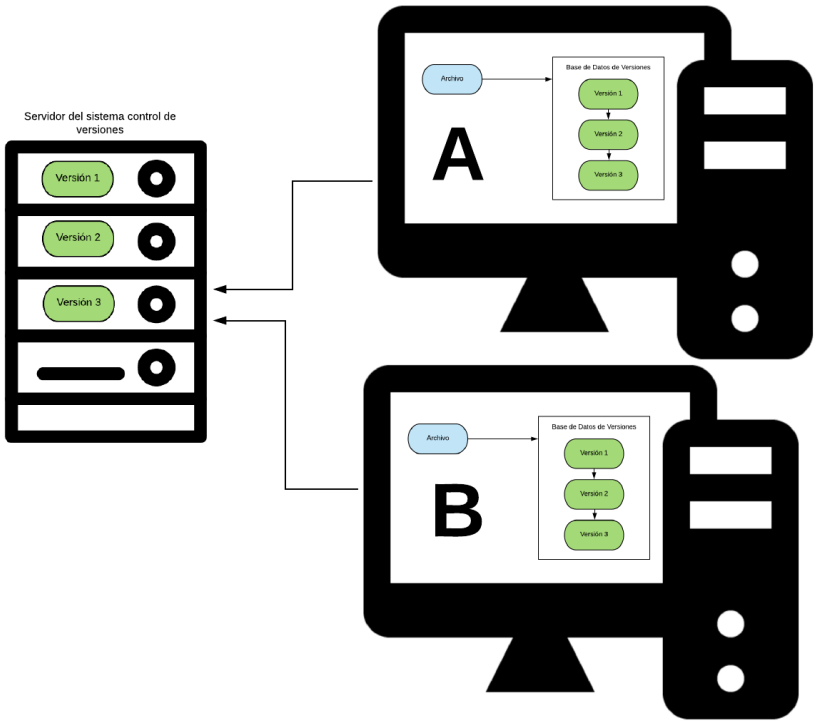


Figura 2.10 Sistema de control de versiones distribuido

Básicamente, muchos de estos sistemas funcionan de forma correcta con varios repositorios, por lo que puede colaborar instantáneamente con diferentes personas en el mismo proyecto, permitiendo así fijar diferentes flujos de trabajo que no son factibles en un sistema centralizado, como en un modelo jerárquico.

Git

Figura 2.11 Logo de Git

Desarrollado bajo la dirección de Linus Torvalds[6], el sistema de control de versiones de Git ciertamente tuvo algo de celebridad desde el principio[10].

El desarrollo de Git fue impulsado por la decisión de BitKeeper[7] de dejar de proporcionar licencias gratuitas a la sociedad de desarrollo de Linux en 2005 (Bitkeeper había sido el VCS elegido para el desarrollo del kernel de Linux). Después de la ruptura con BitKeeper, el proyecto Git se lanzó el 7 de abril de 2005, sólo días antes del lanzamiento de Mercurial, que se inició con el mismo propósito, es decir, para ser un reemplazo de código abierto de Bitkeeper para ser utilizado para el desarrollo de Linux.

Git ha ganado mucha tracción a lo largo de los años y se beneficia de interfaces en línea como GitHub o GitLab, bien diseñada que permite colaboración, revisión de código y administración de código. Poderoso para proyectos privados y de código

[6] Linus Benedict Torvalds es un ingeniero de software finlandés-estadounidense, conocido por iniciar y mantener el desarrollo del kernel Linux.
[7] BitKeeper es un sistema de control de versiones distribuido para el código fuente de los programas producidos a partir de BitMover Inc. y se distribuye bajo la licencia Apache.

abierto". Con GitHub, "los proyectos públicos son siempre gratuitos", y compartir repositorios de Git se ha convertido en el método al que recurren muchos desarrolladores de software de código abierto para compartir y colaborar en proyectos.

Escrito en C / C ++, Git es rápido y estable. Al igual que el shell de Unix, Git es una combinación de muchos componentes que juntos proporcionan la funcionalidad. Por el lado positivo, Git puede hacer casi todo lo que quieras. Pero en el lado negativo, Git puede ser complicado de aprender, por lo que mantener el ritmo de todas sus funciones puede ser un desafío.

Git logra ser el mejor sistema de control de versiones para:

- Los desarrolladores que disfrutan de las secuencias de comandos de shell de Unix.
- Los equipos a los que no les importa una curva de aprendizaje para lograr una funcionalidad asombrosa.

Mercurial

Figura 2.12 Logo de Mercurial

Un competidor directo de Git, y lanzado apenas 12 días después del primero. El sistema de control de versiones de Mercurial es algo más simplificado y bien documentado y está escrito en Python en lugar de C / C ++ (teóricamente lo hace un poco más lento, pero probablemente no se note). Aunque en última instancia se eligió a Git sobre Mercurial para el desarrollo continuo del kernel de Linux, Mercurial ha avanzado y se ha consolidado durante la última década [11].

Mercurial ofrece las siguientes características distintivas:

- Seguimiento del historial familiar de archivos y controles estrictos para preservar la integridad del historial del archivo.
- Soporte de interfaces graficas de usuario (GUI) con herramientas como TortoiseHg y SourceTree.
- Una interfaz de línea de comandos más simple que Git.
- Gran extensibilidad a través de scripts de Shell o usando la API de Python.

Bazaar

Figura 2.13 Logo de Bazaar

Canonical, la compañía con sede en el Reino Unido detrás de Ubuntu (probablemente la distribución de Linux más popular en la actualidad), inició el desarrollo del sistema de control de versiones Bazaar (inicialmente "Baz") en marzo de 2005 (poco antes de los lanzamientos de Git y Mercurial).

Bazaar se ofrece como software gratuito como parte del proyecto GNU, aunque su desarrollo está en curso y todavía está patrocinado por Canonical[8] [12].

Mezclando la ideología de segunda y de tercera generación de sistemas de control de versiones, Bazaar permite configurar ya sea un repositorio centralizado o un repositorio distribuido. Esto puede ser especialmente útil si está migrando desde un repositorio centralizado preexistente con SVN o alguna otra herramienta similar.

Bazaar intenta distinguirse como el "Control de versiones para seres humanos", lo que significa que valoran la simplicidad, los archivos de ayuda limpios y la disponibilidad de GUI para distribuciones de Windows, macOS y Linux. Una característica especial de Bazaar es su soporte para "ramas externas transparentes", que en la práctica significa la capacidad de acceder a repositorios que no pertenecen a Bazaar desde SVN, Git y Mercurial, trabajando en un "entorno de sistemas de control de versiones (VCS) mixto" mientras se mantienen los comandos de Bazaar e interfaz.

Bazaar es una atractiva opción de sistema de control de versiones que ofrece:

- Interfaces GUI simples para Windows, macOS y Linux.
- Archivos de ayuda descriptivos y fáciles de leer para todos los comandos.
- El apoyo de Canonical, un socio de código abierto probado durante más de una década.
- Complementos fáciles de instalar, que recuerdan a la función de complementos en Firefox.

2.4 Integración Continua / Despliegue Continuo (CI/CD)

2.4.1 Integración Continua

En el desarrollo de aplicaciones modernas, el objetivo es permitir que varios programadores manejen diferentes tareas en la misma aplicación a la vez. No obstante, si una empresa reúne todo el código fuente en un día, el trabajo causado puede ser manual y laborioso, además de llevar un tiempo. Todo esto tiene lugar cuando un programador que trabaja de forma independiente pone en funcionamiento una variación en la aplicación. El cambio realizado puede entrar en conflicto con las modificaciones que otros desarrolladores están implementando simultáneamente.

La integración continua ayuda a los programadores a unir las modificaciones realizadas en el código a un repositorio compartido (o "rama") más a menudo, incluso a diario. Una vez que las modificaciones implementadas por los programadores se incorporan a la aplicación, se verificarán mediante la ejecución de pruebas automáticas (generalmente pruebas unitarias y de integración) para verificar que las

[8] Canonical Ltd. es una empresa de programación de ordenadores con base en Reino Unido fundada por el empresario sudafricano Mark Shuttleworth para dedicarse a la promoción y a la venta de soporte comercial y servicios relacionados con Ubuntu y otros proyectos afines.

modificaciones no han dañado la aplicación. Es decir, que se pruebe todo, desde las clases, hasta la funcionalidad de los distintos módulos que componen la aplicación completa. Si la automatización detecta un conflicto entre el código viejo y el código actual, la CI nos ayuda a resolver estos errores de manera fácil y rápida [13].

2.4.2 Despliegue continuo

Después de la automatización del diseño y de las pruebas automáticas de CI, la distribución continua publicará automáticamente el código verificado en el repositorio. Es importante que la CI se haya incorporado a la línea de desarrollo para que el proceso de despliegue continuo resulte efectivo. En un despliegue continuo el objetivo principal es tener un código fuente que se pueda implementar en un entorno de producción cuando se requiera.

Con el despliegue continuo, en cada proceso, la unión de cambios de código hasta la distribución del proyecto preparado para estar en producción, implica pruebas automáticas y que el código quede subido a un repositorio. Finalmente en este proceso, el equipo de desarrollo puede poner en marcha la construcción del proyecto para entrar en producción de forma rápida y sencilla.

2.4.3 Herramientas de integración Continua

Antes de profundizar en las herramientas de automatización de CI/CD, primero debemos comprender el concepto. Como mencionamos anteriormente, la Integración Continua y Despliegue Continuo generalmente van de la mano en un entorno de desarrollo ágil, donde los equipos quieren implementar diferentes piezas de código y que aparezcan a producción tan pronto como sea posible.

El uso de herramientas de CI/CD automatiza el proceso de construcción, prueba e implementación de código. Cada miembro del equipo puede obtener resultados inmediatos sobre la subida a producción de su código, incluso cuando solo cambian una sola línea o carácter. De esa manera, cada miembro del equipo puede llevar su código a producción, mientras el proceso de construcción, prueba e implementación se realiza automáticamente para que puedan pasar a trabajar en una siguiente tarea de la aplicación.

Jenkins

Figura 2.14 Logo de Jenkins

Jenkins es uno de los proyectos más conocidos y comunes en el mercado de CI. Comenzó como un proyecto paralelo de uno de los ingenieros de Sun[9] y se expandió hasta convertirse en una de las mayores herramientas de CI de código abierto que ayuda a los equipos de ingeniería a automatizar sus implementaciones[14].

Tal como promete una herramienta de CI, con Jenkins puede automatizar sus tareas de compilación, prueba e implementación. La herramienta es compatible con Windows, macOS y varios sistemas Unix, y se puede instalar utilizando paquetes de

[9] Sun Microsystems, Inc.

sistema nativos, así como con Docker, o instalarse de forma independiente en cualquier máquina que tenga instalado Java Runtime Environment (JRE)[10].

En el aspecto práctico, Jenkins le da a cualquier miembro del equipo la capacidad de enviar su código a la compilación y obtener reportes inmediatos sobre si está listo para producción o no. En la mayoría de los casos, esto requerirá algunos retoques y ajustes de Jenkins de acuerdo con los requisitos personalizados de su equipo.

Jenkins destaca por su rico ecosistema de complementos. Ofrece una versión extendida con más de 1,000 complementos, lo que permite la integración con casi todas las herramientas y servicios disponibles en el mercado. Al ser una herramienta de código abierto, también le brinda la opción de adaptarla a la medida para una solución local.

El tener código abierto con plugins que mejoran su funcionamiento significa que tendrà una mayor comunidad de desarrolladores. Cualquier configuración, flujo de trabajo, necesidad o mejora, se podrá crear con la ayuda de Jenkins y sus plugins.

Travis

Figura 2.15 Logo Travis CI

Travis CI es uno de los nombres más comunes en el ecosistema CI/CD, creado para proyectos de código abierto y luego expandido a proyectos de código cerrado a lo largo de los años. Se centra en el nivel de CI, mejorando el rendimiento del proceso de construcción con pruebas automáticas y un sistema de alerta.

Travis CI se centra en permitir que los usuarios prueben rápidamente su código a medida que se implementa. Admite cambios de código grandes y pequeños, y está diseñado para identificar cambios en la construcción y las pruebas. Cuando se detecta un cambio, Travis CI puede proporcionar información sobre si el cambio se ha realizado correctamente o no.

Los desarrolladores pueden usar Travis CI para ver las pruebas mientras se ejecutan, ejecutar una serie de pruebas en paralelo e integrar la herramienta con Slack, HipChat, Email, etc[11]. para recibir notificaciones de problemas o compilaciones fallidas.

Travis CI admite compilaciones de contenedores y es compatible con Linux Ubuntu y macOS. Puede usarse en diferentes lenguajes de programación, como Java, C #, Clojure, GO, Haskell, Swift, Perl y mucho más. Tiene una lista limitada de integraciones de terceros y su enfoque está en CI en lugar de CD [15].

Para asegurarse de tener siempre una copia de seguridad de su compilación reciente, Travis CI clona el repositorio remoto en un nuevo entorno virtual cada vez que ejecuta una nueva compilación.

[10] Java Runtime Environment o JRE se usa para la ejecución de aplicaciones Java.
[11] Herramientas de mensajería desde donde se pueden recibir las actualizaciones de Travis CI

Circle CI

Figura 2.16 Logo Circle CI

Circle CI es una herramienta basada en la nube que automatiza el proceso de integración e implementación. También se enfoca en probar cada cambio en el código antes de su implementación, utilizando una serie de métodos como pruebas unitarias, pruebas de integración y pruebas funcionales. La herramienta es compatible con contenedores, macOS, Linux y puede ejecutarse dentro de una nube privada o en su propio centro de datos.

Circle CI se integra con su sistema de control de versiones actual, como GitHub, Bitbucket [12]y otros, y ejecuta una serie de pasos cada vez que se detecta un cambio. Estos cambios pueden ser confirmaciones, relaciones públicas de apertura o cualquier otro cambio en el código.

Cada cambio de código crea una compilación y ejecuta las pruebas en un contenedor limpio o en una máquina virtual, de acuerdo con sus configuraciones y preferencias iniciales. Cada compilación consta de varios pasos, incluidas las dependencias, las pruebas y la implementación. Si la compilación pasa las pruebas, se puede desplegar a través de AWS CodeDeploy, Google Container Engine, Heroku[13], SSH o cualquier otro método de su elección.

El estado de éxito o fracaso de las compilaciones y pruebas en cuestión se envía a través de Slack, HipChat o una serie de otras integraciones, para que el equipo pueda mantenerse actualizado. Es importante tener en cuenta que Circle CI requiere algunos ajustes y cambios para varios idiomas, por lo que es mejor revisar la documentación del idioma que elija.

Circle CI puede cancelar automáticamente las compilaciones redundantes. Si se activa una compilación más nueva en la misma rama, la herramienta la identifica y cancela las compilaciones más antiguas que están en ejecución o en cola, incluso si la compilación no ha finalizado.

Bamboo

Figura 2.17 Logo Bamboo

Bamboo es parte del paquete de productos de Atlassian y, al igual que otras herramientas, ofrece compilación, prueba e implementación de código y es compatible con numerosos idiomas.

Tiene sólidas integraciones con otros productos de Atlassian que son relevantes para el ciclo de CI, como JIRA y Bitbucket[14].

[12] Repositorios web para proyectos.
[13] Plataformas o aplicaciones con servicios en la nube desde donde se pueden desplegar aplicaciones. AWS Codedeploy es propiedad de Amazon, Google Container Engine propiedad de Google y Heroku propiedad de Salesforce.com.
[14] Jira y Bitbucket son herramientas en línea propiedad de Atlassian. Para la administración de tareas y para el almacenamiento en línea de proyectos respectivamente.

La construcción, prueba e implementación son parte del paquete de Bamboo, y la parte de prueba se realiza con la ayuda de Bamboo Agents. Similar a los agentes en el monitoreo de Java, Bamboo también ofrece dos tipos; los agentes locales que se ejecutan en servidor Bamboo como parte de su proceso, mientras que los agentes remotos se ejecutan en otros servidores y computadoras. Cada agente se asigna a las compilaciones que coinciden con sus capacidades, lo que permite asignar diferentes agentes a diferentes compilaciones.

La principal ventaja que ofrece Bamboo son los fuertes vínculos con el resto de productos de Atlassian, como JIRA y Bitbucket. Con Bamboo puede ver los cambios de código y los problemas de JIRA que se han introducido en el código desde la última implementación. De esa manera, los desarrolladores pueden sincronizar su flujo de trabajo y mantenerse siempre encaminados y saber qué versión es la siguiente y qué debería haberse corregido.

Bamboo viene con el respaldo fuerte de Atlassian, junto con mejores flujos de trabajo para los productos existentes de la compañía.

	Jenkins	**Travis CI**	**Circle CI**	**Bamboo**
SO compatibles	Windows, Linux, macOS, todos los SO Unix	Linux, macOS	Linux, iOS, Android	Windows, Linux, macOS, Solaris,
Hosting	Local / Nube	Local / Nube	Cloud	Local / Bitbucket
Soporte de Contenedores	✓	✓	✓	✓
Plugins	★★★★★	★★★★	★★★	★★
Documentación	Adecuada	Pobre	Buena	Buena
Facilidad de uso	Fácil	Fácil	Fácil	Media
Uso	Para grandes Proyectos	Para proyectos pequeños	Para un desarrollo rápido y un presupuesto elevado	Para integración con Atlassian
Precio	Gratuito	$69-$489	$50-$3150	$10-$800

Figura 2.18 Tabla comparativa Jenkins, Travis CI, Circle CI, Bamboo [16]

2.5 Virtualización en contenedores

2.5.1 Virtualización

La virtualización está revolucionando la informática. Consiste en un mecanismo que permite compartir una máquina física para ejecutar varias máquinas virtuales que consisten en sistemas operativos. Los recursos de un PC como es la memoria, CPU, disco y conexión a internet, son utilizados por las máquinas virtuales que de otra manera sin ser usadas esperan sólo picos de trabajo.

Este sistema nos permite ejecutar máquinas virtualizadas con independencia del hardware que tengan por debajo. No es necesario reinstalar ni migrar un sistema para mover una máquina virtual.

La virtualización crea una interfaz externa que oculta la implementación subyacente combinando recursos en diferentes ubicaciones físicas o simplificando el sistema de control.

Los contenedores utilizados en los servidores no son de reciente invención, no obstante por medio de proyectos de código libre como Docker y LXC esta tecnología está a día de hoy muy difundida [17].

2.5.2 Contenedores

El concepto básico, un contenedor de software, se considera como una aplicación creada para ser ejecutada en un servidor. Un contenedor se puede instalar en un ordenador mediante una imagen portable que contiene todos los recursos necesarios para poder poner en marcha un entorno virtual. La mayoría de S.O soportan el uso de contenedores, ya sea Linux, macOS o Windows. Sin embargo, la virtualización de sistemas operativos, para la gran mayoría de usuarios, solo comienza a ser factible a través de plataformas como Docker ya que añaden algunas funcionalidades que ayudan a utilizar estos contenedores.

Un contenedor se encargará de proveer un ambiente de ejecución ligero que contiene los archivos, variables y librerías necesarias para funcionar correctamente. Son utilizados para garantizar que las aplicaciones funcionen correctamente cuando se modifique su entorno, reduciendo los posibles fallos y aumentando su portabilidad.

Docker

Figura 2.19 Logo de Docker

Docker se ha posicionado en muy poco tiempo en un lugar significante entre los grupos de DevOps y CD, es una plataforma libre para desarrollar, enviar y ejecutar aplicaciones que permite separar las aplicaciones de su infraestructura para que se pueda entregar software rápidamente. Con Docker, se puede administrar su infraestructura de la misma manera que administra sus aplicaciones.

Se usan características de aislamiento de recursos del núcleo de Linux , como grupos y espacios de nombres para permitir que los contenedores independientemente se ejecuten dentro de una sola instancia de Linux, evitando de este modo la sobrecarga que implica iniciar y mantener máquinas virtuales. Docker favorece el funcionamiento en paralelo de contenedores ya que están aislados sin usar elementos de la máquina donde se ejecutan.

Docker se ha distribuido como edición libre Community Edition(CE) que es gratuita y compatible con la comunidad colaborativa de Docker. Y también con la edición de pago Enterprise Edition (EE) tiene soporte de Docker, certificación, administración de contenedores (Docker Datacenter) y escaneo de seguridad [18].

Ventajas	Inconvenientes
Docker es compatible con diferentes S.O. y plataformas en la nube.	Docker solo es compatible con su motor.
Docker ofrece herramientas de gestión de clusters como son Swarm y Compose.	El software se proporciona en forma de un archivo general que contiene todas sus funciones.
Gran variedad de imágenes disponibles para su descarga desde Docker Hub.	Los contenedores Docker no virtualiza S.O. solo aíslan los procesos entre sí.
Docker se encuentra en constante crecimiento, con plugins y componentes.	

Figura 2.20 Tabla de ventajas e inconvenientes de Docker

LXC

Figura 2.22 Logo de LXC

Lxc (Linux Containers) es una tecnología de virtualización en el nivel de sistema operativo (SO) para Linux . OpenVZ[15] Permite que un servidor físico ejecute múltiples instancias de sistemas operativos aislados, conocidos como Servidores Privados Virtuales (SPV o VPS en inglés) o Entornos Virtuales (EV). Lxc no provee de una máquina virtual, más bien provee de un entorno virtual que tiene su propio espacio de procesos y redes.

Es similar a otras tecnologías de virtualización al nivel de SO como OpenVZ y Linux-VServer, asimismo se parece a otras herramientas de sistemas operativos como FreeBSD jail y Solaris Containers.

Su objetivo es crear un entorno para contenedores de software que se diferencie lo mínimo posible de la instalación Linux estándar. LXC progresa conjuntamente con otros proyectos de código libre como LXD, LXCFS y CGManager[16].

En su momento, LXC se creó para efectuar diferentes contenedores de sistema (full system containers) en un plan anfitrión. Un contenedor Linux suele empezar una distribución íntegra en un entorno virtual en función de la imagen del sistema operativo

[15] Es una tecnología de virtualización a nivel de S.O para Linux. OpenVZ permite instancias aisladas de S.O.
[16] Infrastructuras desarrolladas por linuxcontainers.org para proyectos de contenedores

y los usuarios interactúan con ella de forma parecida a como harían con una máquina virtual. Los contenedores Linux rara vez inician aplicaciones, con lo que se diferencian claramente de Docker. Mientras que LXC se centra sobre todo en la virtualización de sistemas, Docker se concentra en la virtualización y el despliegue de aplicaciones. Al principio también se utilizaban contenedores Linux con este objetivo. Hoy Docker apuesta por un formato de contenedor propio.

Diferencias entre Docker y LXC

Una diferencia fundamental entre ambas tecnologías de virtualización se encuentra en los procesos que una y otra son capaces de desarrollar, en el caso de los contenedores Linux muchos, pero solo uno en el caso de Docker. Generalmente, las aplicaciones de Docker más complejas están compuestas por varios contenedores. Un despliegue efectivo de estas aplicaciones multicontenedor requiere el uso de herramientas adicionales.

Otra gran diferencia entre los contenedores Docker y LXC es la portabilidad. Desarrollar un software basado en LXC en un sistema de prueba local no garantiza el funcionamiento sin errores del contenedor en otros sistemas (un sistema productivo). La plataforma Docker, sin embargo, abstrae las aplicaciones de un modo más efectivo del sistema subyacente, de forma que un contenedor Docker puede funcionar en cualquier plataforma que tenga Docker instalado sin depender del sistema operativo ni de la configuración de hardware del equipo.

LXC también funciona sin demonio central y, en su lugar, el software se integra en sistemas init como systemd y upstart, a semejanza de rkt, para iniciar y administrar contenedores.

	Docker	LXC
Tecnología de virtualización	En el nivel del sistema operativo	En el nivel del sistema operativo
Full system container	No	Sí
Contenedor de aplicaciones	Sí	No
Licencia	Apache 2.0	GNU LGPLv2.1+
Formato del contenedor	Docker Container	Linux Container (LXC)
Plataformas compatibles	Linux, Windows, macOS, Microsoft Azure, Amazon Web Services (AWS)	Linux
El núcleo de Linux requiere un parche	No	No
Lenguaje de programación	Go	C, Python 3, Shell, Lua

Figura 2.23 Tabla comparativa de Docker RKT y LXC

2.5.3 Ventajas de la tecnología de contenedores

Como hemos dicho anteriormente los contenedores contienen todos los recursos que utilizan las aplicaciones para su funcionamiento, de esta manera se facilita el uso a las personas encargadas de administrar la aplicación, además, también facilita la instalación y la ejecución de los complejos programas que la componen. En cambio, las ventajas más notables de los contenedores, se encuentran, principalmente en la automatización y gestión de software basado en contenedores.

- **Instalación**: es más sencilla, se realiza a través de imágenes o representaciones portables de un contenedor, incorporando todos los componentes que requiere el programa como son, las librerías y archivos de configuración. De esta forma, se compensan las diferencias entre sistemas operativos. Su instalación queda reducida a la ejecución de una sola línea de comando.

- **Plataforma**: la plataforma donde se usan las imágenes son independientes, gracias a la portabilidad de las imágenes se pueden llevar fácilmente de un sistema a otro. El único requisito para ejecutar un contenedor partiendo de una imagen es que el S.O. soporte contenedores.

- **Mínimas pérdidas**: la virtualización con contenedores requiere de 100MB de espacio y pocos minutos de instalación ya sea usando LCX o Docker. La virtualización de hardware provoca una pérdida de rendimiento para el hipervisor y otros S.O.. Al usar contenedores y prescindir de todo esto, las pérdidas son mínimas. El arranque de una máquina virtual suele durar pocos minutos y sus aplicaciones pueden estar disponibles rápidamente.

- **Aislamiento**: cada aplicación se puede ejecutar de forma independiente a diferentes contenedores, de forma que en un mismo sistema se pueden ejecutar aplicaciones con requerimientos diferentes.

- **Administración y automatización únicas**: como en una plataforma como Docker todos los contenedores están ejecutados con herramientas similares, y con ellas será probable automatizar todas las aplicaciones de modo centralizado. Estas soluciones están indicadas ante todo para arquitecturas de servidor en la que los componentes están distribuidos en varios servidores, de forma que se carga con los pesos de instancias diferentes. En estos ámbitos de aplicación, el contenedor Docker dispone de las herramientas con las cuales configurar automatismos. Esto facilita, por ejemplo, iniciar instancias nuevas de forma automática en momentos puntuales de sobrecarga. Google nos ofrece la herramienta de Kubernetes, un software para la orquestación de grandes clusters de contenedores.

2.6 Las pruebas automáticas en CI/CD

La integración continua y el despliegue continuo tienen como objetivo principal que los grupos de desarrollo de software proporcionen a los clientes contenido útil, de esta forma se proporciona valor y consiguen resultados útiles sobre cómo se utilizan sus productos en el mundo real. Gran cantidad de empresas han seguido las prácticas de desarrollo y operaciones DevOps[17] para mantenerse al día con las empresas rivales.

Sin embargo, para proporcionar contenido útil a los clientes primero se han de encontrar y corregir errores más rápidamente, mejorar la calidad y reducir el tiempo necesario para verificar y lanzar nuevas actualizaciones de software. Por este motivo las pruebas en profundidad son esenciales en la integración continua y en el despliegue continuo.

2.6.1 Ventajas de las pruebas automáticas

El testeo es fundamental para conseguir una buena calidad en el software producido y ha sido durante mucho tiempo una parte primordial en el proceso de creación de software [19]. Con el modelo de desarrollo en cascada las pruebas y el fortalecimiento de la calidad del software producido se hacen después de que la aplicación se haya construido y desplegado, con el propósito de comprobar el correcto funcionamiento de acuerdo con las especificaciones.

Con este antiguo modelo de desarrollo se hace más lento el proceso de despliegue y provoca que las personas encargadas del desarrollo no puedan verificar que lo que se ha desarrollado se ejecuta correctamente hasta mucho más tarde, cosa que provoca que se utilice mucho más tiempo para corregir los errores y desplegar de nuevo la aplicación.

Por el contrario, con CI/CD se permiten metodologías ágiles cortas e iterativas, proporcionan rápidamente un resultado de los errores encontrados, cosa que hace que las actualizaciones sean más frecuentes. Una parte esencial de estos cortos ciclos de iteración son pruebas para verificar de forma automática que el nuevo código es bueno y no rompe ninguna parte de la aplicación.

CI/CD supone la confirmación periódica de modificaciones de código en la rama principal de desarrollo, activando el proceso de compilación si los cambios corresponden y probando el software en cada ejecución. Las ventajas de usar CI/CD son más visibles si se incorporan las modificaciones de código al menos una vez al día. Sin embargo, en los grupos de desarrollo tanto grandes como pequeños, realizar manualmente este nivel de prueba equivale a hacer un sobreesfuerzo importante para las personas encargadas de elaborar este trabajo además implica hacer más de una vez el mismo trabajo. Las pruebas automáticas se encargaran de reducir esta gran carga de trabajo.

[17] DevOps son prácticas de desarrollo que tiene como objetivo hacer más rápido el desarrollo de software y proporcionar CI/CD.

Las pruebas automáticas son perfectas para trabajos repetitivos y de esta forma se obtienen resultados más acordes que con las pruebas realizadas manualmente, porque cuando se les pide a los testers que realicen los mismos pasos más de una vez, se les pueden olvidar detalles y probar inconsistencias.

Estas pruebas automáticas son más rápidas que las realizadas manuales, además estas se pueden efectuar de forma paralela, por lo que cuando los plazos de entrega son escasos, la paralelización es un beneficio. Además, la redacción de pruebas automáticas requiere tiempo por adelantado, siempre que los miembros de un grupo de desarrollo empiecen a realizar cambios con regularidad y se desplieguen con más frecuencia, valdrá la pena.

Además, las pruebas automáticas suprimen trabajos repetitivos y tediosos, de tal manera se evita que las personas encargadas del testeo queden agotadas. Estas mismas personas también colaboran en la elaboración de las pruebas trabajando conjuntamente con el equipo de desarrollo.

2.6.2 Las pruebas en el proceso de CI/CD

Las pruebas se llevan a cabo en diferentes fases del proceso de desarrollo. La CI/CD está compuesta de ciclos de retroalimentación cortos, que hacen que el equipo encuentre problemas rápidamente.

Si un problema se detecta de forma tardía, será más difícil de solucionar, ya que puede provocar que se escriba más código erróneo. Para un equipo de desarrollo será conveniente realizar pruebas antes de continuar con un nuevo paso en el proyecto.

Dentro del proceso de CI/CD existen diferentes herramientas que permiten realizar pruebas automáticas, por lo que se pueden ingresar datos de muestra en él y ejecutarse en las diferentes etapas correspondientes, obteniendo resultados después de cada una de ellas. En función de los resultados obtenidos en las pruebas y de la herramienta de CI/CD utilizada, se podrá seleccionar el siguiente paso a ejecutar.

Normalmente las pruebas más rápidas se ejecutan primero, de este modo optimizamos el proceso de CI/CD para sacar el máximo provecho. Permitiendo obtener los resultados erróneos más pronto y de esta forma evitar ejecutar las pruebas más largas y difíciles.

La pirámide de pruebas nos ayudará a elaborar y ejecutar mejor las pruebas automáticas.

2.6.3 Pirámide de Pruebas

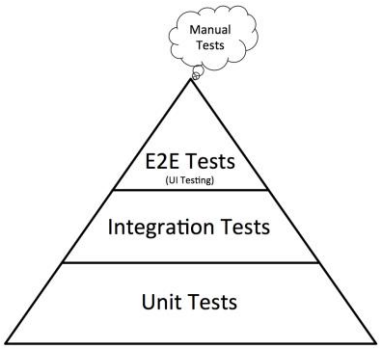


Figura 2.24 Pirámide de Pruebas

Un modo fácil para entender cómo se deben de ejecutar las pruebas en el proceso de CI/CD es la Pirámide de Pruebas, en cuanto al número de pruebas y la prioridad de ejecución. La Pirámide de pruebas fue definida por Mike Cohn [20]. En la base encontraremos las pruebas unitarias, en la parte central las pruebas de integración y en la parte más alta las pruebas de interfaz de usuario.

Puede que no se entienda bien pero, la suposición es clara: empieza con una base fuerte que son las pruebas unitarias automáticas que son fáciles de ejecutar y rápidas, después de pasar a las pruebas de integración y de interfaz que son más difíciles de programar y tienen una ejecución más lenta.

Analizemos los diferentes tipos de pruebas.

Pruebas unitarias

La base de la pirámide de pruebas la forman las pruebas unitarias. Para comprobar que el código funciona correctamente, han sido diseñadas este tipo de pruebas, donde se trata la unidad de funcionamiento más pequeña posible. A medida que se escriben las pruebas, el equipo de desarrollo debe ser responsable y escribirlas cuando se construye el código.

En un equipo de desarrollo que se encuentra trabajando en un proyecto y anteriormente no se han creado pruebas unitarias, escribirlas para todo el código fuente de cero puede resultar una hazaña imposible de abordar. Aunque es recomendable el uso de pruebas unitarias para cubrir un amplio rango, se puede comenzar con algo inicial y ampliarlo posteriormente. La manera de crear pruebas unitarias puede realizarse siguiendo una estrategia donde se añadan estas pruebas a cualquier trozo de código que le pertenezca, y de esta forma asegurar que todo el código quede incluido en estas.

Pruebas de integración

Las pruebas de integración son la fase en que los módulos de software individuales se combinan y prueban como un grupo. Las pruebas de integración se llevan a cabo para evaluar el cumplimiento de un sistema o componente con los requisitos funcionales especificados, como son la base de datos y la aplicación. Ocurre después de las pruebas unitarias y antes de las pruebas interfaz. Las pruebas de integración toman como entrada los módulos que han sido probados por las pruebas unitarias y los agrupa en agregados más grandes.

Podemos dividir las pruebas de integración en pruebas extensas y pruebas limitadas. Con las pruebas extensas se utiliza el componente o servicio real, mientras que con las pruebas limitadas se utiliza un archivo de ejemplo o servidor doble para pruebas. Según lo tedioso que sea nuestro software y del número de servicios que posea, conviene desarrollar pruebas extensas o limitadas ya que estas últimas se ejecutan de manera más rápida.

Pruebas de extremo a extremo

Conocidas también como pruebas E2E, las pruebas extremo a extremo se encargan de revisar el funcionamiento de la interfaz de la aplicación. Generalmente, la prueba de un extremo a extremo es el proceso de probar una pieza de software de principio a fin porque será utilizada por usuarios reales. En una aplicación web, las pruebas consisten en iniciar un navegador web, navegar a la URL correcta, usar la aplicación según lo previsto y comprobar el comportamiento de la misma. Según la pirámide de pruebas estas pruebas son las menos recomendables porque tardan mucho en ejecutarse y son mucho más frágiles.

Toda modificación que se realice en la interfaz del usuario puede provocar alteraciones en estas pruebas, por este motivo pueden resultar molestas en el resultado de las pruebas ya que deben de estar bien actualizadas para no dar errores. Conviene diseñar las pruebas E2E a medida en entornos de prueba que se asemejen lo máximo a la realidad y de esta forma obtener pruebas que se puedan utilizar sin dar problemas.

Pruebas de rendimiento

Estas pruebas no están incluidas en la pirámide de pruebas pero merece la pena incluirlas en el grupo de pruebas automáticas, en especial si los productos desarrollados tienen que tener una gran estabilidad y ser rápidos.

Se incluyen una serie de estrategias de prueba para comprobar el rendimiento del software en un entorno real. Las pruebas de carga examinan el rendimiento del sistema cuando aumenta la demanda, mientras que las pruebas de estrés exceden deliberadamente el uso esperado y las pruebas de asimilación (o resistencia) miden el rendimiento bajo continuas cargas elevadas.

Con este tipo de pruebas, no se pretende únicamente que se confirme que el software funciona correctamente dentro de los valores definidos, sino también confirmar que se comporta de forma correcta cuando se superan estos parámetros.

Entornos de pruebas

Las pruebas de extremo a extremo vistas anteriormente como las pruebas de rendimiento requieren datos de prueba y que sean lo más parecidos a los de producción. Para que el programa de pruebas automatizado brinde confianza en el software que se está probando, es importante que las pruebas siempre se ejecuten de la misma manera. Esto incluye asegurarse de que el entorno de prueba sea

consistente entre ejecuciones (aunque deben actualizarse para adaptarse a la producción cuando se aplican cambios).

La gestión manual del entorno puede ser una trabajo que requiera mucho tiempo, por lo que vale la pena considerar los pasos para automatizar la creación y descomposición del entorno de preproducción para que en cada compilación sea nuevamente creado.

2.6.4 Pruebas manuales

Habitualmente se puede malentender que en un proceso de CI/CD las pruebas manuales realizadas por personas profesionales que entienden el entorno, son sustituidas por las pruebas automáticas. El hecho de que las pruebas sean automáticas sí que deja algo más de tiempo a los componentes del grupo de testeo, pero no les reemplaza. Los testers no necesitan perder tiempo en tareas que se repiten muchas veces, pueden concentrarse en crear casos de prueba, elaborar pruebas automatizadas y aplicar su creatividad y originalidad a las pruebas exploratorias.

Las pruebas automáticas requieren una elaboración metódica de scripts para que se puedan usar, a diferencia de estas, las pruebas manuales solo requieren una ligera base y un documento donde anotar los resultados. El objetivo de las pruebas manuales es buscar errores que las pruebas automáticas hayan pasado por alto. Además sirven para encontrar errores que aún no se han planeado y para los que no se ha escrito ninguna prueba. Tras tomar una decisión de qué parte se va a comprobar, se debe pensar en las nuevas funciones añadidas y también en las partes que pueden afectar estos cambios.

Las pruebas manuales no deben de convertirse en pruebas tediosas y repetitivas, ya que el propósito no es repetir el mismo conjunto de pruebas cada vez. Al detectar un error en las pruebas manuales, se debe resolver, además de tomar un tiempo para escribir una posible prueba automática que se encargará de crear el equipo de desarrollo. De este modo si el problema vuelve a suceder, puede ser detectado en una etapa más temprana del desarrollo. Para utilizar eficazmente el tiempo de los testers, las pruebas manuales solo se deben de realizar después de que hayan pasado todas las pruebas automáticas.

2.6.5 Objetivo de las pruebas

Las pruebas automáticas que forman parte de un proceso de CI/CD tienen como objetivo obtener una respuesta sobre las modificaciones y cambios que se han realizado en el proyecto, así que es esencial entender el resultado de estas pruebas. Los servidores de integración continua generalmente se integran con herramientas de pruebas automáticas para que se puedan ver los resultados en un único sitio. Los grupos de desarrollo obtienen notificaciones automáticas con los resultados de la compilación, estas notificaciones normalmente son a través de correos o plataformas de comunicación como Slack que es usada en la empresa

Para analizar el resultado de cuando falla una prueba nos hace falta saber a qué parte del proyecto corresponde, es por este motivo que las pruebas deben de estar bien documentadas y que con los resultados seamos capaces de saber que parte de código hay que corregir únicamente mirando los valores de la salida, también nos sirven de ayuda las capturas de pantalla de las pruebas E2E. Una forma de documentar bien las pruebas es poner etiquetas a cada comprobación y que cada comprobación únicamente compruebe una cosa, de esta forma serà facil entender porque ha fallado su ejecución. En la CI/CD existen diferentes herramientas que nos ayudan a obtener información adicional sobre los errores y de esta forma nos pueden ayudar a retomar el proceso de compilación más rápido.

Para que un proyecto de CI/CD sea lo más implementable posible es necesario que todo el equipo entienda el valor que nos ofrecen las pruebas automatizadas y de esta forma sean capaces de responder lo más rápido posible a los errores. Así se consigue mantener el software en buen estado.

2.6.6 Importancia del desarrollo paralelo de las pruebas automáticas.

En el proceso de CI/CD las pruebas automáticas realizan un papel muy importante. Para escribir pruebas automáticas se exige la utilización de tiempo y esfuerzo, pero con este trabajo los beneficios de la retroalimentación rápida y la comprensión de la viabilidad del código hacen que las pruebas automáticas valgan la pena. No hay que olvidar las pruebas una vez hayan sido creadas, hay que mantenerlas al día en paralelo a las actualizaciones de la aplicación.

Las pruebas automáticas deben ser parte de la aplicación al igual que el código fuente, eso quiere decir que de la misma forma que se actualiza el código fuente, también se deben actualizar las pruebas para asegurar que continúan siendo funcionales, por lo tanto, requieren de un mantenimiento constante.

Que el código de las pruebas automáticas esté en continuo mantenimiento requiere una cobertura por parte del equipo de desarrollo, para ello los desarrolladores tendrán que hacer pruebas de uso para detectar los errores y de esta forma automatizar un proceso de comprobación del error detectado. Además es aconsejable que se entienda como se trabaja con las pruebas para ayudar a encontrar una solución en caso de detectar algún error.

Con las herramientas que se usan para la CI podemos obtener una variedad de indicadores que nos ayudan a optimizar el proceso. Si las pruebas son poco fiables nos darán una falsa visión y un motivo de preocupación por no saber si el sistema va a fallar o no. Tampoco debemos tener plena confianza en las pruebas, porque siempre nos puede aparecer un error inesperado que tendremos que solucionar lo más rápido posible para poder continuar con la puesta en producción de la aplicación. El objetivo real es ofrecer software funcional a los usuarios de forma regular. La automatización de pruebas logra este objetivo al obtener una rápida y confiable retroalimentación, de esta forma se puede desplegar la aplicación en un entorno de producción tranquilamente.

3. Desarrollo

En este tercer capítulo del trabajo vamos a justificar y analizar las diferentes metodologías y tecnologías que se han elegido en la empresa para la mejora del proceso de desarrollo web.

3.1 Desarrollo web

El desarrollo web hace referencia a la creación y mantenimiento de sitios web. Incluye aspectos tales como el diseño de páginas web, la publicación de estas mismas en repositorios web, la programación web y gestión de bases de datos.

Si bien los términos "desarrollador web" y "diseñador web" se utilizan a menudo como sinónimos, no significan lo mismo. Técnicamente, un diseñador web solo diseña interfaces de sitios web utilizando HTML y CSS. Un desarrollador web puede participar en el diseño del sitio web, pero también puede escribir diferentes scripts para crear funciones específicas en el sitio web. De esta forma, los desarrolladores web pueden ayudar a mantener y actualizar las bases de datos que utilizan los sitios web dinámicos.

El desarrollo web incluye muchos tipos de creación de contenido web. En los últimos años, los sistemas de gestión de contenido como WordPress, Drupal y Joomla[18] se han convertido en medios populares de desarrollo web. Estas herramientas facilitan a cualquier persona la creación y edición de su propio sitio web mediante una interfaz basada en web.

Si bien existen varios métodos para crear sitios web, a menudo existe un compromiso entre la simplicidad y la personalización. Por lo tanto, la mayoría de las grandes empresas no utilizan sistemas de gestión de contenido, sino que cuentan con un equipo de desarrollo web dedicado que diseña y mantiene los sitios web de la empresa.

3.2 Metodología

Con el uso de las metodologías tradicionales como es el modelo en cascada, se buscaba poner un orden al proceso de desarrollo de software y de esta forma conseguir que sea eficaz y previsible.

Las metodologías clásicas intentan exigir disciplina al proceso de desarrollo de software y de esa forma volverlo predecible y eficiente. Para lograr este objetivo, se mantiene un proceso detallado, enfocado en los planes propios de otras empresas de ingeniería. El problema principal de usar este método es que hay demasiadas tareas para seguir la metodología tradicional, lo que retrasa la fase de desarrollo [21].

Otras metodologías más ligeras que intentaban disminuir la probabilidad de sufrir algún fallo por no tener en cuenta el coste, funcionalidad y tiempo en los proyectos de

[18] Software con el que se pueden gestionar webs a través de una interfaz.

desarrollo. Surgieron en los años 90, las denominadas posteriormente, metodologías ágiles. Estas metodologías son una respuesta a las metodologías existentes y tienen como objetivo reducir la burocracia que implica la aplicación de metodologías tradicionales en proyectos pequeños y de mediana escala.

En las metodologías ágiles encontramos dos diferencias esenciales si las comparamos con las metodologías tradicionales:

- Los métodos ágiles son adaptativos y no predictivos.
- Las métodos ágiles son orientadas a las personas y no orientadas a los procesos.

Las metodologías ágiles son adaptativas. Este hecho es de gran importancia ya que contrasta con la predictibilidad buscada por las metodologías tradicionales. Con el enfoque de las metodologías ágiles los cambios son eventos esperados que generan valor para el cliente.

Delante de la necesidad de crear un equipo de trabajo colaborativo donde a su vez cada equipo realice diferentes tareas y que sean capaces de adaptarse a los cambios que surjan o nos propongan los clientes de manera rápida y sin perder mucho tiempo en la planificación para no retrasar el proceso de desarrollo, se ha optado por la metodología de desarrollo ágil.

3.2.1 Metodologías Ágiles

Los métodos ágiles son flexibles y pueden modificarse para adaptarse a la situación real de cada equipo de desarrollo y proyecto. Se reparten en proyectos inferiores por medio de una lista ordenada de particularidades. Cada proyecto se trata por separado, fomentando la división de características en un corto periodo de tiempo. Exigiremos la colaboración de un representante del cliente durante el desarrollo, debido a que la comunicación con él será constante. Los proyectos son inmensamente participativos y se ajustan a los cambios mejor. Una característica anhelada son los cambios en los requerimientos, así como las constantes entregas al cliente y la obtención de feedback con él. Frecuentemente el producto y proceso se mejoran.

Metodologías Ágiles	Metodologías Tradicionales
Basada en heurísticas procedentes de prácticas de desarrollo de software.	Basada en reglas que provienen de los estándares de los entornos de desarrollo.
Está preparada para admitir cambios durante el proyecto..	Con resistencia a las modificaciones.
Reglas definidas internamente por el equipo de desarrollo.	Reglas definidas externamente.
Evolución poco controlada, con pocos principios.	Evolución muy controlada, muchas normas que seguir.
En el equipo de desarrollo también se encuentra el cliente.	El equipo de desarrollo interactúa con el cliente a través de reuniones.
Equipos reducidos (<10)	Equipos grandes (>10)
La arquitectura del software tiene poco énfasis.	La arquitectura del software es esencial.
Contrato flexible e incluso inexistente .	Contrato predefinido.

Figura 3.1 Tabla comparativa entre metodologías ágiles y tradicionales

3.3 Framework web

Un framework web es una herramienta de desarrollo, generalmente definida como una aplicación o un conjunto de módulos, que permiten el desarrollo ágil de aplicaciones a través de las contribuciones de bibliotecas con funciones creadas.

Estos marcos garantizan que el desarrollador no reinvente código constantemente, sino que se centre en el problema que quiere resolver, en lugar de la implementación de las funciones de uso común que otros ya han resuelto.

Los principales objetivos que se persiguen en un framework web son:

- Conseguir que el desarrollo de aplicaciones web sea más rápido.
- Que se use código que ya existe y de esta forma con el uso de patrones fomentar las buenas prácticas de desarrollo.

Por lo tanto, definimos como un framework web, un grupo de componentes que forman un diseño que se puede reutilizar para promover y acelerar el desarrollo de aplicaciones web.

3.3.1 AngularJS

Figura 3.2 Logo AngulasJS

El framework AngularJS es de código abierto desarrollado por Google para promover la creación y programación de aplicaciones web de página única SPA (Single Page Application).

AngularJS mantiene separados por completo el front-end y back-end de la aplicación, evita escribir código que se repite y hace que todo esté más organizado gracias a su modelo MVC [19] para asegurar un desarrollo rápido, al tiempo que permite modificaciones y actualizaciones.

En un sitio web SPA puede parecer que la velocidad de carga resulte lenta la primera vez que se ejecuta, la navegación posterior es completamente instantánea, ya que se ha cargado toda la página de una vez.

Esta es solo una ruta que el servidor debe enviar, y lo que AngularJS hace en segundo plano es cambiar la vista mientras navega para que parezca una Web normal, pero de una manera más dinámica.

Entre las diferentes ventajas que ofrece este framework cabe destacar que es escalable y modular adaptándose a nuestras necesidades, al estar basado en los estándares de componentes web, cuenta con un conjunto de interfaces de programación de aplicaciones (APIs) que permiten la creación de nuevas etiquetas HTML personalizadas que además son reutilizables.

El principal lenguaje de programación de AngularJS es HTML y Javascript, por lo que toda la sintaxis y formas de hacer las cosas con estos lenguajes son iguales de hacer que como si se usaran en otros sitios, lo que aumenta la coherencia y consistencia de la información. Si se quiere añadir un programador nuevo, no le va a costar entender como funciona si conoce estos lenguajes.

Como ya se mencionó, las plantillas de AngularJS almacenan el código de la interfaz de usuario (front-end) y el código de lógica de negocio (back-end) por separado, además de otros beneficios, también pueden hacer un mejor uso de otras herramientas previamente existentes.

Si esto no es suficiente, los editores de código y los entornos de desarrollo integrado (IDE) han proporcionado extensiones para que sea más cómodo usar esta herramienta.

Debido a su programación reactiva, la vista se actualiza automáticamente después de realizar cambios.

[19] Modelo Vista Controlador

3.4 Gestión del código fuente

En esta sección vamos a abordar la primera etapa del flujo de trabajo que se quiere llegar a adoptar. Para tratar esto, analizaremos el funcionamiento del sistema de control de versiones distribuido que se quiere utilizar.

Un VCS almacena archivos de código en los llamados repositorios. Desde un punto de vista práctico, un archivo de código puede entenderse como un archivo de texto plano. Esto es muy importante para poder definir cuál es su unidad atómica, es decir, la parte más pequeña que se puede modificar para entender que el archivo ha sido actualizado. Hoy en día, la mayoría de los VCS utilizan la línea como unidad atómica.

Actualmente la implementación más común se basa en un repositorio distribuido.Si el proyecto es grande, esta arquitectura proporciona a los desarrolladores una mayor independencia, ya que en la primera etapa, sus cambios sólo se guardarán en su repositorio local. El sistema también puede acelerar la operación, porque cuando se trabaja localmente, no es necesario operar con frecuencia a través de la red.

En la arquitectura de repositorios distribuidos, para poner en común el trabajo de todos los desarrolladores, es necesario combinar los cambios realizados sobre sus respectivos repositorios locales en un repositorio remoto compartido entre todos ellos.

3.4.1 GitLab

Figura 3.3 Logo de GitLab

En el año 2011 se creó GitLab, de la mano de Dimitri Saparoschez, quien programó, en lenguaje Ruby[20], y divulgó este software basado en la web. Es un popular VCS que se usa, principalmente, en el ámbito del desarrollo de software. A día de hoy, la mayoría de desarrolladores apuestan por él.

Como ventaja principal de GitLab, tenemos la notable facilidad del uso de metodologías ágiles entre diferentes grupos de desarrolladores, igualmente éstos pueden realizar el trabajo al mismo tiempo y de forma paralela editar funciones. El uso de protocolos en todos los procesos de forma continua garantizan que no se pierdan modificaciones del código ni se sustituyan de forma no intencionada. Además los cambios fijados se pueden deshacer fácilmente.

GitLab es gratuito y de código abierto. Está considerado como uno de los sistemas de control de versiones más utilizados. Además está basado en la herramienta Git vista anteriormente.

¿Cómo funciona?

GitLab se puede instalar en un servidor propio, es una aplicación web con una interfaz gráfica de usuario [22]. La esencia de GitLab la forman proyectos, donde el código a editar se almacena en archivos digitales, llamados repositorios. Estos directorios de

[20] Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos, creado por el programador japonés Yukihiro Matsumot.

proyectos contienen todo el contenido y los archivos del proyecto de software, como archivos HTML, CSS, JavaScript o PHP[21].

Inicialmente, los usuarios implicados en el proyecto descargan su propia copia del repositorio central en sus ordenadores y luego realizan modificaciones de código a través de los llamados commits. Después de editar, las modificaciones se integrarán en el repositorio principal.

Otra importante característica es la ramificación, la cual facilita a los usuarios crear una "rama" que se ramifica desde la parte principal del código y se puede editar independientemente. Esta función es útil para introducir y probar nuevas funciones sin afectar al desarrollo de la rama master.

Gracias al soporte y a la CI, GitLab es una herramienta perfecta para realizar trabajos con los que usar la ramificación y ofrecer tareas útiles como la solicitud de combinaciones y la creación de divisiones.

Las etapas de un archivo en GitLab

Dentro de un repositorio de GitLab, lo primero que tendremos que localizar son las etapas principales que les ocurren a los archivos. Como ejemplo, hasta el momento, solo poseemos el desarrollo de un proyecto que inicialmente nos muestra un gráfico donde se cuentan las entrevistas de ventas de vehículos por día y vemos la urgencia de desarrollar otro gráfico para que nos muestre el número de entrevistas que hace cada vendedor.

Con este fin, iniciamos el desarrollo del módulo. Necesitamos registrar los cambios del archivo para crear uno nuevo y poder volver al archivo eliminado en caso de problemas. Posteriormente, analizaremos algunas de estas tareas. En las siguientes etapas observaremos que sucede con estos archivos:

- Directorio de trabajo.
- Área de preparación.
- Repositorio Git.

En el área de trabajo, encontramos todos los cambios que hicimos para desarrollar el nuevo módulo, que actualmente se halla en el estado “modificado”.

Al sistema debemos informarle de qué archivos queremos cargar o modificar en el repositorio, así entrarán en el estado “listo”, quedando en el área de preparación. Por último, cuando ya estemos seguros y pasemos estos cambios a estado “confirmado”, los encontraremos en nuestro repositorio de GitLab, donde poseeremos la actualización con los nuevos gráficos.

Fases en el repositorio

[21] Lenguajes de programación usados para el desarrollo web.

Como se ha comentado en el apartado anterior, existen algunos comandos con los que hay que lograr completar las fases. Para entender cómo funcionan estos comandos veremos nuevos conceptos de GitLab.

- Add: método que nos deja agregar los cambios efectuados. Es decir, variar el estado de nuestros archivos de “modificado” a “preparado”, y transfiere el archivo del directorio de trabajo al área de preparación.
- Commit: método que realiza el cambio de nuestros archivos en estado “preparado” a “confirmados”, lo que significa que los deja terminados en el repositorio de GitLab.

¿Qué son las ramas y cómo funcionan?

Dentro del repositorio local, vimos las etapas por las que pasan nuestros archivos y los métodos que nos conceden realizar estos pasos.

Ahora, entendamos un nuevo concepto llamado "rama": una línea de desarrollo única en el repositorio, que puede ayudarnos a tener más control. De hecho, un almacén debe tener al menos una rama. Por convención, la rama maestra se denomina "master".

Todo nuestro desarrollo se lleva a cabo en la rama “master”, cuando iniciamos un nuevo desarrollo lo realizaremos en una rama que llamamos "Develop". De esta forma, si algo sale mal, se producirá un error en la rama "Develop", no en la rama “master”. Esto garantizará la integridad de nuestro código.

Una vez finalizado el desarrollo del módulo y luego de verificar que está correctamente diseñado y probado, podremos agregar nuestro código de la rama de desarrollo al código de la rama principal.

3.4.2 GitKraken

El entorno nativo de Git es la línea de comandos. Sólo desde la línea de comandos se encuentra disponible todo el poder de Git. El texto plano no siempre es la mejor opción para todas las tareas y en ocasiones se necesita una representación visual además, algunos usuarios se sienten más cómodos con una interfaz gráfica de apuntar y pulsar.

GitKraken es una potente y elegante interfaz gráfica multiplataforma para git desarrollada con Electron[22]. De forma muy sencilla podemos llevar el completo seguimiento de nuestros repositorios, ver ramas, tags, crear nuevos, todo el historial de nuestro trabajo, commits, etc.

Cómo hemos dicho, GitKraken es multiplataforma, por lo tanto podemos utilizarlo en Windows, Linux y macOS.

[22] Electron es un framework de código abierto creado por Cheng Zhao

Ha sido la plataforma utilizada para ver de forma visual el estado de nuestros proyectos.

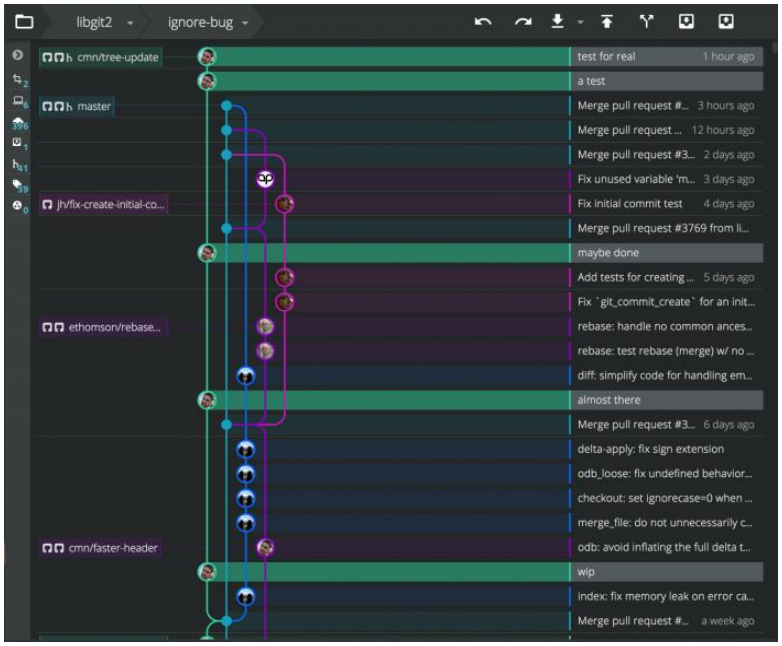


Figura 3.4 Captura de GitKraken

3.5 Integración Continua

En la sección 2.4 se ha descrito el concepto de Integración Continua, pero en esta sección vamos a analizar esta idea, cómo adaptarla a la arquitectura que se quiere llegar a adoptar en la empresa y con qué herramientas contaremos para este fin.

En la empresa se quiere sustituir el proceso de desarrollo anterior por un sistema más moderno que facilite el despliegue de las aplicaciones web que se desarrollan. Por este motivo vamos a empezar a explicar en qué consiste la solución que se quiere llegar a adoptar.

En lugar de que los desarrolladores trabajen de forma aislada y no se integren lo suficiente, se quiere introducir la integración continua para garantizar que los cambios de código y las compilaciones nunca se realicen de forma aislada.

Dado que el entorno de cada desarrollador varía con la modificación del código, esperamos reducir este riesgo durante el proceso de integración.

3.5.1 El Objetivo de la Integración Continua.

En un nivel alto, el Objetivo de CI es [23]:

- Reducir riesgos
- Reducir los procesos manuales repetitivos
- Generar software que se pueda implementar en producción en cualquier momento y en cualquier lugar

- Tener una mejor visibilidad del proyecto
- Establecer una mayor confianza en el producto de software del equipo de desarrollo.

Vamos a analizar estos principios y ver qué valor ofrecen.

Reducir riesgos

Integrando código muchas veces al día, se puede reducir el riesgo en nuestro proyecto. Hacerlo facilita la detección de errores, la comprobación del estado del software y la reducción de las suposiciones.

- **Los errores se detectan y solucionan antes**. Debido a que CI integra y ejecuta pruebas e inspecciones varias veces al día, existe una mayor probabilidad de que se descubran errores, por ejemplo al subir el código al repositorio GitLab en lugar de durante las pruebas automáticas que se realizan más tarde.
- **La salud del software se puede medir**. Al incorporar pruebas e inspecciones continuas en el proceso, los atributos de salud del producto de software, como la complejidad, se pueden rastrear a lo largo del tiempo.
- **Reducir las suposiciones**. Con CI, al construir y probar el software en un entorno limpio utilizando el mismo proyecto y scripts, se pueden reducir las suposiciones (suponer que nuestro código va a funcionar en un entorno nuevo que no tenía las bibliotecas ya instaladas del anterior).

Reducir los procesos manuales repetitivos

Reducir los procesos repetitivos ahorra tiempo, costes y esfuerzo. Estos procesos repetitivos pueden ocurrir en todas las actividades del proyecto, incluida la compilación de código, la integración de la base de datos, las pruebas automáticas, las pruebas manuales, en la implementación y la retroalimentación. Al usar CI, se tiene una mayor capacidad para garantizar las siguientes características:

- El proceso se ejecuta de la misma manera cada vez.
- Se sigue un proceso ordenado. Por ejemplo, se pueden ejecutar las pruebas unitarias, antes de ejecutar el despliegue.
- Los procesos se ejecutarán cada vez que se produzca una confirmación en la rama master del repositorio de control de versiones.

Esto facilita:

- La reducción de la mano de obra en procesos repetitivos, esto libera a los desarrolladores para que realicen más trabajos de provecho y que utilicen más la reflexión.

- Que los miembros del equipo (que no quieran desarrollar pruebas) vean que la automatización es provechosa y de este modo contribuyan al desarrollo de las pruebas.

La generación de software que se pueda implementar en producción.

La CI puede permitirle lanzar software que se puede desplegar en producción en cualquier momento. Desde una perspectiva externa, este es el beneficio más obvio de CI. Podemos hablar infinitamente de las mejoras de la calidad del software y la reducción de riesgos en CI, pero el software desplegable es el activo más real para los agentes externos, como son los clientes y usuarios. Con CI, se realizan pequeños cambios en el código fuente e integran estos cambios con el resto del código de forma regular. Si hay algún problema, se informa a los miembros del desarrollo y las correcciones se aplican al software inmediatamente. Los proyectos que no integran la CI no podrán probar el software hasta que no se vaya a entregar. Esto puede retrasar una versión, retrasar la reparación de errores, causar nuevos errores, y de forma muy grave, puede significar el final de un proyecto.

Tener una mejor visibilidad del proyecto

CI proporciona la capacidad de detectar tendencias y tomar decisiones que sean efectivas, ayuda a proporcionar nuevas mejoras. Los proyectos sufren cuando no hay patrones que respaldan las decisiones a tomar, por lo que todos los miembros ofrecen sus mejores ideas de mejora. Normalmente, los desarrolladores del proyecto recopilan esta información manualmente, lo que hace que el esfuerzo sea mayor e inoportuno.

- **Decisiones efectivas:** Un sistema de CI puede proporcionar información puntual sobre el estado de la compilación y métricas de calidad. Algunos sistemas de CI también pueden mostrar tasas de defectos.

- **Detección de tendencias:** Dado que las integraciones ocurren con frecuencia en un sistema de CI, la capacidad de detectar tendencias es el éxito o el fracaso de compilación, de esta forma se puede mejorar la calidad general del proyecto.

Establecer una mayor confianza en el producto

En general, el uso eficaz de las prácticas de CI puede proporcionar una mayor confianza en la producción de un producto software. Con cada compilación, el equipo sabe que se ejecutan pruebas contra el software para verificar el funcionamiento, que se cumplen los estándares de diseño y la codificación del proyecto.

Sin integraciones frecuentes, algunos equipos pueden sentirse asustados porque no conocen qué repercusión tendrán los cambios en el código. Dado que un sistema de CI puede informar cuando algo ha salido mal, los desarrolladores y otros miembros del equipo tendrán más confianza para realizar cambios. Debido a que la CI fomenta que exista un único punto desde donde se construyen toda la aplicación, existe una mayor confianza con el producto, en el caso de nuestro proyecto, con Jenkins.

3.5.2 Prácticas a adoptar en CI

Para que CI funcione de manera eficaz en un proyecto, los desarrolladores deben cambiar sus hábitos habituales de desarrollo de software del día a día. Los desarrolladores deben confirmar el código con más frecuencia, hacer que sea una prioridad reparar las compilaciones rotas, escribir pruebas automáticas y no obtener ni enviar código roto hacia el repositorio de control de versiones.

Las prácticas a adoptar que recomendamos requieren algo de conocimientos previos, así se podrán conseguir las ventajas que veremos en esta sección.

Hay siete prácticas que funcionan bien para los equipos que ejecutan CI en un proyecto.

- Confirmar código con frecuencia.
- No cometas código roto.
- Repare las compilaciones rotas de inmediato.
- Escribe pruebas de desarrollador automáticas.
- Todas las pruebas e inspecciones deben pasar.
- Ejecutar compilaciones privadas.
- Evite que se rompa el código.

Las siguientes secciones cubren cada práctica con mayor detalle.

Confirmar código con frecuencia

- **Hacer pequeños cambios.** Tratar de no cambiar muchos componentes a la vez. Elegir una pequeña tarea, escribir las pruebas, el código fuente, ejecutar las pruebas y luego enviar el código al repositorio de control de versiones.
- **Compromiso después de cada tarea.** Se requiere que los desarrolladores confirmen el código a medida que completan cada tarea.
- **Evitar que todos hagan cambios a la misma hora**. Se producirán muchos más errores si se suben todos los cambios al mismo tiempo ya que las colisiones serán mayores y más difíciles de gestionar.

No comprometa el código roto

No se debe enviar código que no funciona al repositorio de control de versiones. La máxima mitigación para este riesgo es tener un script de compilación bien factorizado que compile y pruebe el código repetidamente.

Repare las compilaciones rotas de inmediato

Una compilación rota es cualquier acción que impide que la compilación se realice exitosamente. Esto puede ser un error de compilación, una prueba automática fallida, un problema con la base de datos o una implementación fallida. Cuando se trabaja con

un entorno de CI, estos problemas deben solucionarse de inmediato. Afortunadamente, en un entorno de CI, cada error se descubre de forma incremental y, por lo tanto, es probable que sea muy pequeño.

Escribir pruebas de desarrollador automáticas

Una compilación debe estar completamente automatizada. Escribir las pruebas para frameworks ya existentes como Karma o Protractor para que se puedan automatizar fácilmente.

Todas las pruebas e inspecciones deben pasar

En un entorno de CI, el 100% de las pruebas automáticas de un proyecto deben pasar para que la compilación pase. Las pruebas automáticas son tan importantes como la compilación. Todo el mundo sabe que el código que no se compila no funcionará; por lo tanto, el código que tenga errores de prueba tampoco funcionará.

Ejecutar compilaciones privadas

Para evitar compilaciones rotas, los desarrolladores deben emular una compilación real en un entorno de prueba. En primer lugar es preferible de manera local y después de completar sus pruebas unitarias, hacer la compilación donde se puede ejecutar en un entorno de prueba.

Evite obtener código roto

Cuando la compilación no funciona, no se debe consultar el código más reciente del repositorio de control de versiones. De lo contrario, debe dedicar tiempo a desarrollar una prueba para encontrar el error que se ha producido en la compilación, solo para poder compilar y probar el código nuevamente y que este error no se produzca.

3.5.3 Problemas a evitar

La CI tiene numerosos beneficios, pero también prevé muchas preocupaciones que hay que evitar.

- **Aumento de los gastos en el mantenimiento**. Esta suele ser una impresión equívoca, porque la necesidad de integrar, probar, inspeccionar e implementar existe independientemente de si se está utilizando CI. Administrar un sistema de CI robusto es mejor que administrar procesos manuales.

- **Demasiados cambios**. Algunos pueden sentir que hay demasiados procesos que deben cambiar para lograr CI en un proyecto. Un enfoque incremental de CI es más eficaz.

- **Demasiadas construcciones fallidas**. Normalmente, esto ocurre cuando los desarrolladores no están realizando compilaciones privadas antes de enviar el código a un repositorio de control de versiones. Podría ser que un desarrollador se olvide de registrar un archivo o fallar en algunas pruebas.

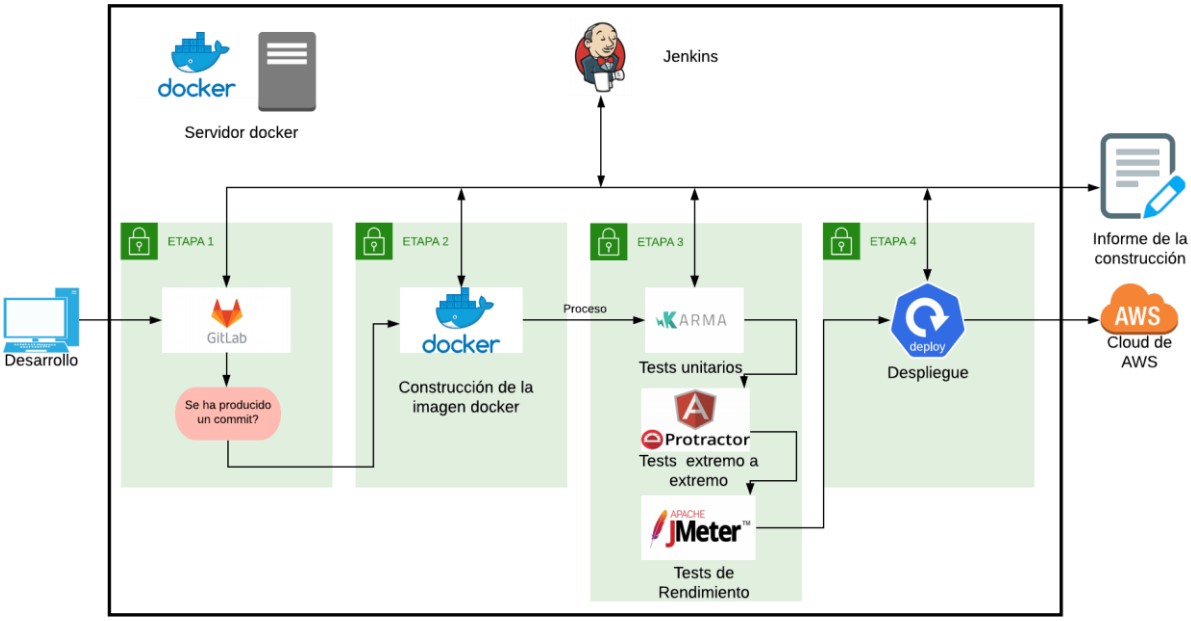
- **Costos adicionales de hardware / software**. Para utilizar CI de manera eficaz, se debe usar una máquina de integración que esté separada, esto

supone un gasto mínimo en comparación a los costos que nos pueden causar los problemas detectados más adelante en el ciclo de vida del desarrollo.

3.5.4 Flujo de trabajo

La siguiente imagen muestra cómo funciona todo el flujo de trabajo de integración continua que se quiere llegar adoptar en la empresa. Más adelante lo analizaremos con más detalle.

Figura 3.5 Diagrama de la solución en la empresa



3.5.5 Funcionalidad de Docker

Utilizaremos Docker para englobar las aplicaciones que forman parte de nuestro sistema de integración continua y entrega continua, así podrán funcionar en cualquier ámbito.

Con la intención de que nuestro sistema pueda funcionar en una gran variedad de entornos diferentes y sea ampliamente portable, todas las aplicaciones que lo componen estarán encapsuladas con Docker.

Tendremos que configurar una imagen Docker que contenga un servidor Jenkins con todos los plugins y las tareas necesarias para realizar las siguientes funciones:

- Compilar.
- Crear una imagen Docker.
- Pasar tests unitarios, extremo a extremo y tests de rendimiento.
- Desplegar.
- Enviar correos electrónicos.

Para que el pipeline funcione correctamente tendremos que instalar todas las librerías y programas para poder compilar los lenguajes de programación escogidos.

3.5.6 Integración Contínua con Jenkins

Lo primero para empezar a usar la integración continua será crear una imagen Docker que contenga el servidor Jenkins. Esta imagen deberá ser capaz de compilar distintos lenguajes de programación, por lo que tendremos que instalar las librerías y programas necesarios para hacerlo. Esta imagen, además, tendrá que poder usar comandos, ya sea para crear las imágenes de la aplicación.

Entonces, según el flujo de trabajo anterior, generalmente así es como funciona el proceso de integración continua [24].

1. Primero, un desarrollador confirma el código en el repositorio de control de versiones.
2. Mientras tanto, el servidor de GitLab al detectar que se ha subido nuevo código en la rama maestra, disparará el inicio de nuestro pipeline de Jenkins.
3. El servidor de integración continua recupera la última copia del código del repositorio y luego ejecuta un script de compilación, que integra el software y comenzará a preparar una nueva imagen Docker.
4. Si lo construido es correcto, entonces Jenkins implementa la construcción de la aplicación en el servidor de prueba.
5. Luego, se llevan a cabo pruebas de ese proyecto. Si las pruebas tienen éxito, el código está listo para implementarse en el servidor de ensayo o de producción.
6. Jenkins genera comentarios enviando por correo electrónico los resultados de la ejecución a los miembros especificados del proyecto. Si falla la compilación, se notificará al equipo afectado.
7. Jenkins continuará esperando a ser gatillado por GitLab si se suben cambios en el repositorio de control de versiones y repetir todo el proceso.

3.5.7 Ventajas de Jenkins

- Es fácil tanto de utilizar como de instalar, es de código abierto y no requiere instalación de complementos extra.
- Es gratuito.
- Puede ser modificado y ampliado de forma fácil. Utiliza código de manera inmediata y gracias a él se pueden obtener reportes de las construcciones.
- Jenkins proporciona herramientas para integraciones continuas y despliegue continuo.
- Está disponible para la mayoría de sistemas operativos como Linux, macOS o Windows.
- Dispone de una gran variedad de plugins. Esto hace que Jenkins sea más flexible facilitando la construcción, implementación y automatización.

- Tiene grandes comunidades en línea con equipos ágiles, lo que proporciona un gran soporte.
- La detección de errores hace que el tiempo que utiliza el desarrollador sea menor al no tener que corregir errores a gran escala si se corrigen de forma temprana.
- Los problemas de integración son menores ya que todo el proceso de integración se puede automatizar. Esto nos permite ahorrar dinero y tiempo durante el ciclo de vida de un proyecto.

3.6 Configuración del entorno

En esta sección se pretende explicar la construcción del entorno de integración continua que se ha descrito anteriormente.

3.6.1 Instalación de Docker en nuestro servidor Linux

El servidor Linux que se ha decidido utilizar usa la distribución CentOS. Un sistema operativo de código abierto, basado en la distribución Red Hat Enterprise Linux, operándose de manera similar, y cuyo objetivo es ofrecer al usuario un software de "clase empresarial" gratuito.

Para instalar Jenkins lo primero que se ha hecho es una instalación de Docker [25].

```
$ sudo yum-config-manager \ --add-repo \
    https://download.docker.com/Linux/centos/docker-ce.repo
```

Figura 3.6 Comando para la instalación de Docker en CentOS

La instalación de Docker Engine, nos permitirá administrar el motor de Docker mediante el terminal de comandos y de esta forma utilizar las imágenes, archivos y contenedores de Docker.

```
$ sudo yum install docker-ce docker-ce-cli containerd.io
```

Figura 3.7 Comando para la instalación de Docker Engine

3.6.2 Instalación de Docker Compose

Para poder crear y ejecutar imágenes debemos instalar la herramienta Docker Compose.

Docker Compose es una herramienta que nos ayudará a definir y compartir aplicaciones en diferentes contenedores. Con Compose, se usa un archivo YAML para configurar los servicios de la aplicación, con un único comando crea, inicia o borra la aplicación.

Una ventaja que podemos obtener utilizando Compose es que un usuario puede clonar e iniciar la aplicación en cualquier momento. Para poderse iniciar por un segundo usuario, previamente debe de estar definida la pila de la aplicación en el fichero YAML que incorporaremos en nuestro repositorio.

Para la instalación de Docker Compose debemos tener en cuenta la versión a instalar, que se puede consultar desde la web de Docker[25].

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/(version a instalar)/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Figura 3.8 Comando para la instalación de Docker Compose

A continuación agregamos permisos de ejecución al binario.

```
$ sudo chmod +x /usr/local/bin/docker-compose
```

Figura 3.9 Comando para agregar permisos a Docker Compose

3.6.3 Instalación de Jenkins

Una vez ya tenemos instalado Docker, la instalación de imágenes dentro de Docker es muy sencilla.

Para la instalación de Jenkins hemos utilizado una imagen ya creada que hemos encontrado dentro de Docker Hub.

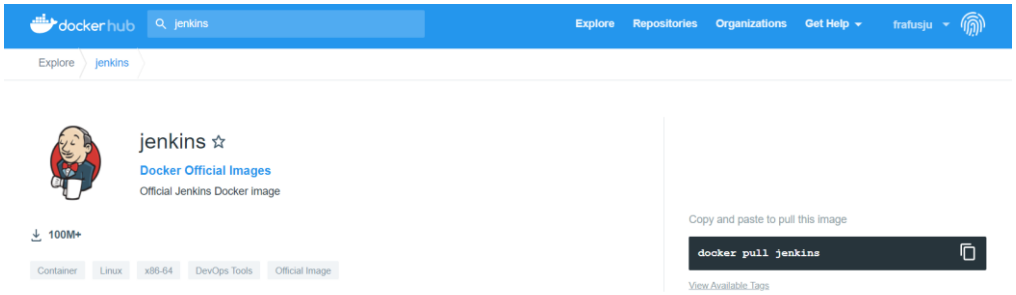


Figura 3.10 Captura de la imagen de Jenkins a descargar desde Docker Hub

La única instrucción que debemos utilizar para descargar la imagen y poderla utilizar posteriormente es la siguiente:

```
$ docker pull jenkins
```

Figura 3.11 Comando para la instalación de Jenkins desde Docker

Para su instalación vamos a crear un archivo docker-compose.yml donde vamos a especificar el contenido del servicio que queremos.

```
services:
  jenkins:
    container_name: jenkins
    image: jenkins
    ports:
      - "8080:8080"
    volumes:
      - $PWD/jenkins_home:/var/jenkins_home
    networks:
      - net
networks:
  net:
```

Figura 3.12 Contenido de docker-compose.yml

Analizamos el archivo y vemos que consiste en la creación de un servicio llamado jenkins, donde el contenedor de la imagen que vamos a utilizar también lo llamaremos jenkins.

La imagen que se va a utilizar para su creación es la descargada con la instrucción que hemos mencionado anteriormente: jenkins/jenkins. A continuación especificamos el puerto donde queremos que esté escuchando nuestro servicio, la ruta donde alojar sus ficheros y la red donde debe estar nuestro servicio.

Después de crear el servicio de Jenkins, debemos seguir con la instalación para crear el primer usuario.

3.6.4 Instalación de GitLab

Al igual que hemos hecho anteriormente para instalar Jenkins, nos descargaremos la imagen de GitLab desde Docker Hub.

```
$ docker pull gitlab/gitlab-ce
```

Figura 3.13 Comando para la instalación de GitLab desde Docker

Para su instalación lo que tenemos que hacer es crear un nuevo servicio en nuestro archivo docker-compose.yml

En el archivo tendremos que añadir el nuevo servicio con su respectivo nombre, git en este caso. El contenedor que se va a encargar de correr la imagen de git lo llamaremos git-server y va a utilizar la imagen que hemos descargado desde Docker Hub. Seguidamente, igual que hemos hecho con Jenkins, le especificamos los puertos donde queremos que esté escuchando y la ruta donde se deben alojar sus ficheros.

```
git:
    container_name: git-server
    hostname: gitlab.example.com
    ports:
      - "443:443"
      - "80:80"
    volumes:
      - "/srv/gitlab/config:/etc/gitlab"
      - "/srv/gitlab/logs:/var/log/gitlab"
      - "/srv/gitlab/data:/var/opt/gitlab"
    image: gitlab/gitlab-ce
    networks:
      - net
```

Figura 3.14 Servicio a añadir dentro de docker-compose.yml

3.6.5 Conexión de GitLab y Jenkins

Para empezar con la conexión de GitLab y Jenkins primero debemos de crear una pareja de llaves SSH. Una vez generadas las llaves, accedemos a GitLab y nos dirigimos a Settings/SSH Keys y guardamos la clave pública que hemos generado.

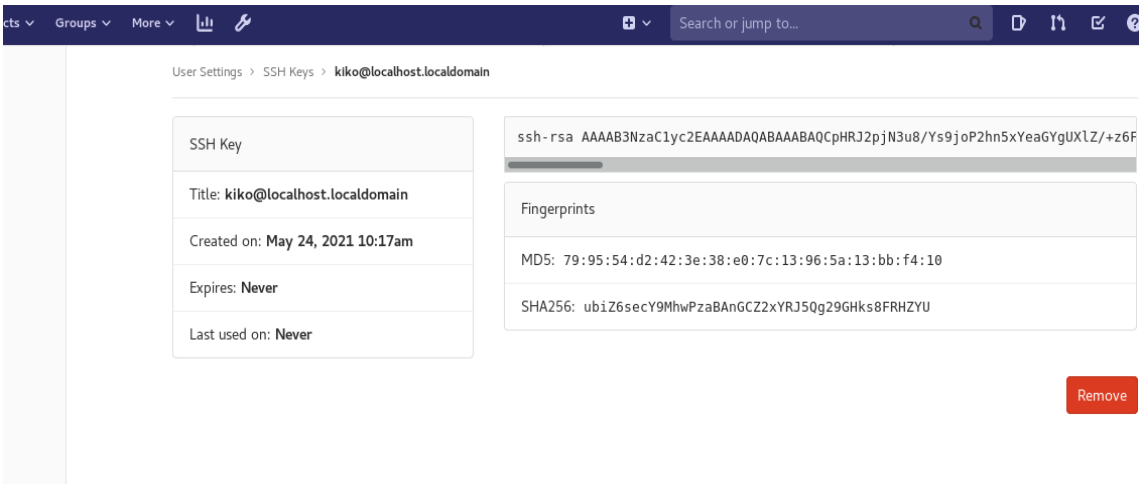


Figura 3.15 Configuración de la conexión entre GitLab y Jenkins

Posteriormente debemos acceder a Settings/ Access Tokens y generar un token con las opciones de: api+read_user+read-repository.

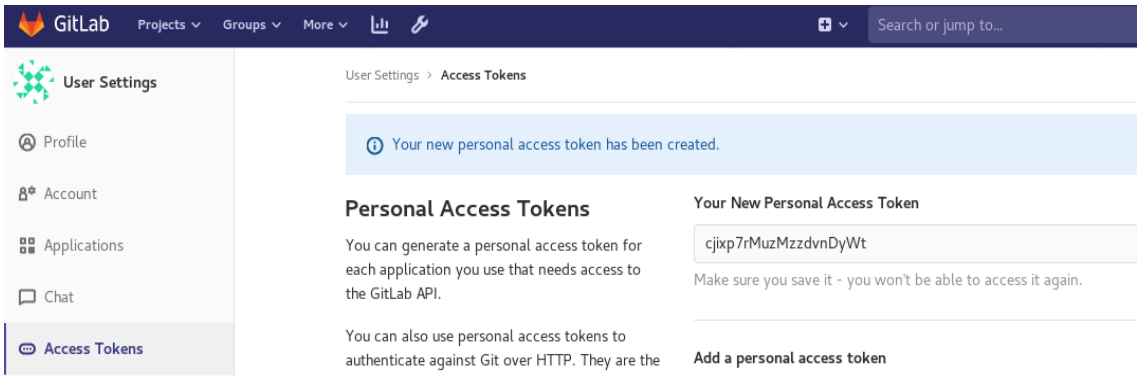


Figura 3.16 Configuración de la conexión entre GitLab y Jenkins 2

Ahora nos dirigimos a la configuración de Jenkins y al igual que con GitLab empezamos con la configuración de las llaves SSH. Para ello, accedemos a la sección de credenciales en la administración de Jenkins y debemos crear una credencial con la llave SSH privada.

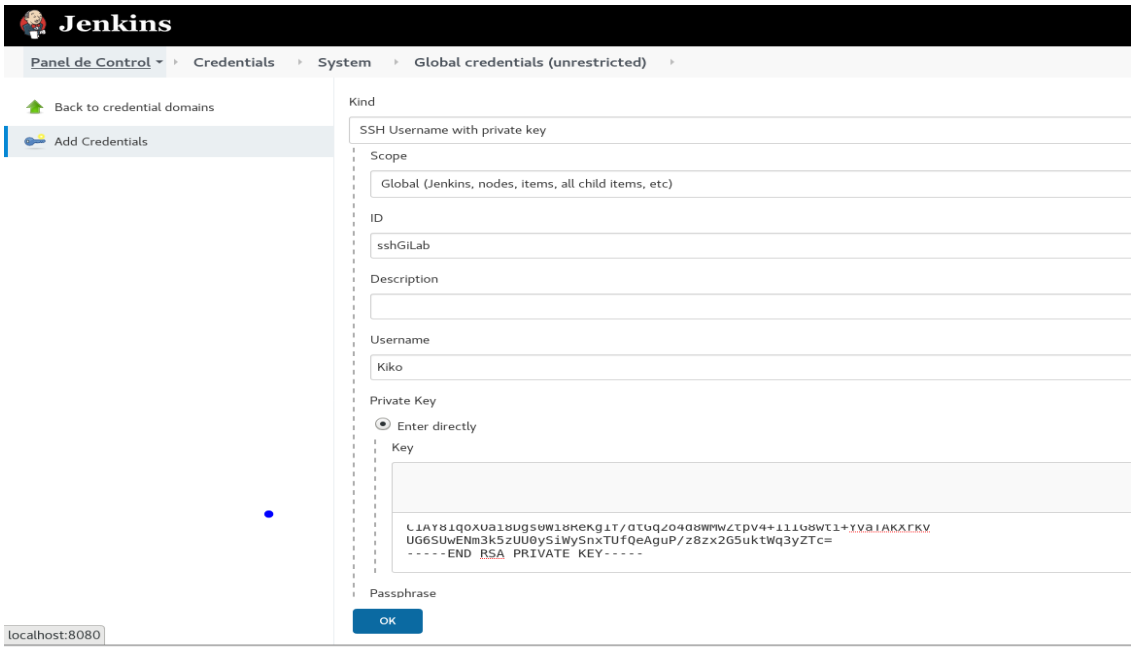


Figura 3.17 Configuración de la conexión entre GitLab y Jenkins 3

También añadimos el token que hemos obtenido en GitLab como credencial. Para añadir el Token, previamente debemos tener el plugin de GitLab instalado en Jenkins.

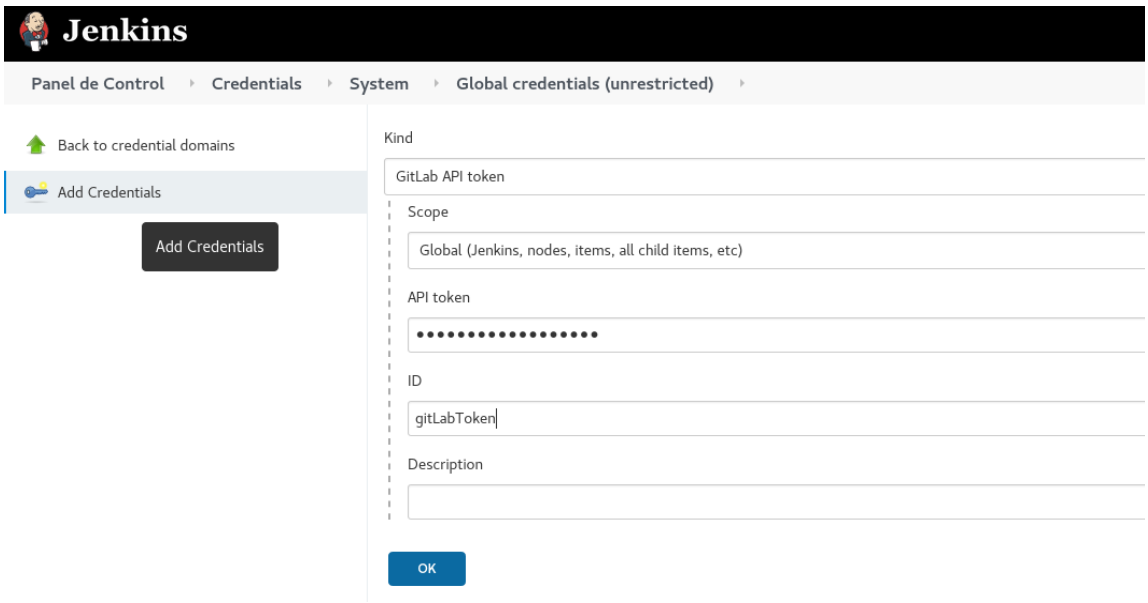


Figura 3.18 Configuración de la conexión entre GitLab y Jenkins 4

Una vez que hemos configurado las credenciales de acceso a la API podemos configurar la conexión entre las dos plataformas. Para ello accedemos desde Jenkins a la administración y desde la administración a configuración.

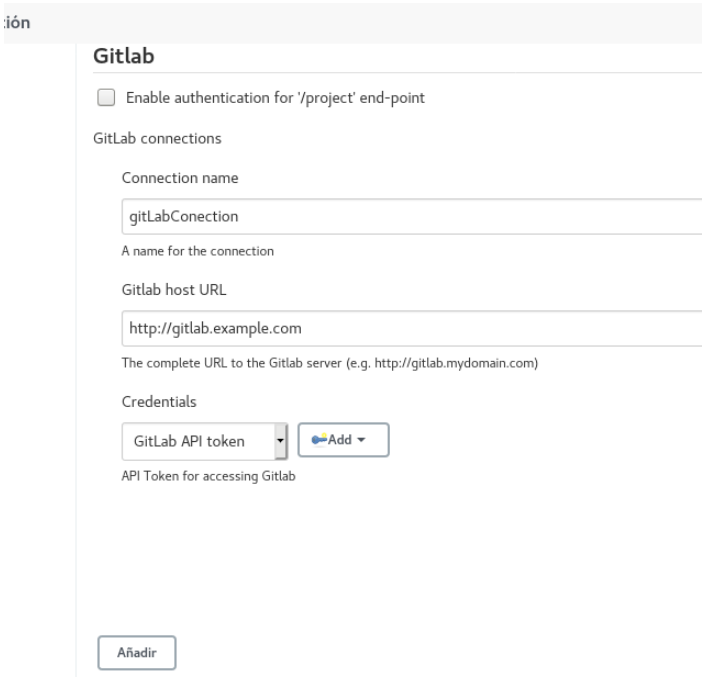


Figura 3.19 Configuración de la conexión entre GitLab y Jenkins 5

Una vez ya tenemos la conexión creada, el próximo paso a seguir ya sería en la configuración de un pipeline, donde elegiremos cuando se va a empezar a ejecutar nuestro pipeline. Próximamente se detallarán los pasos a seguir en cada etapa del proceso de CI.

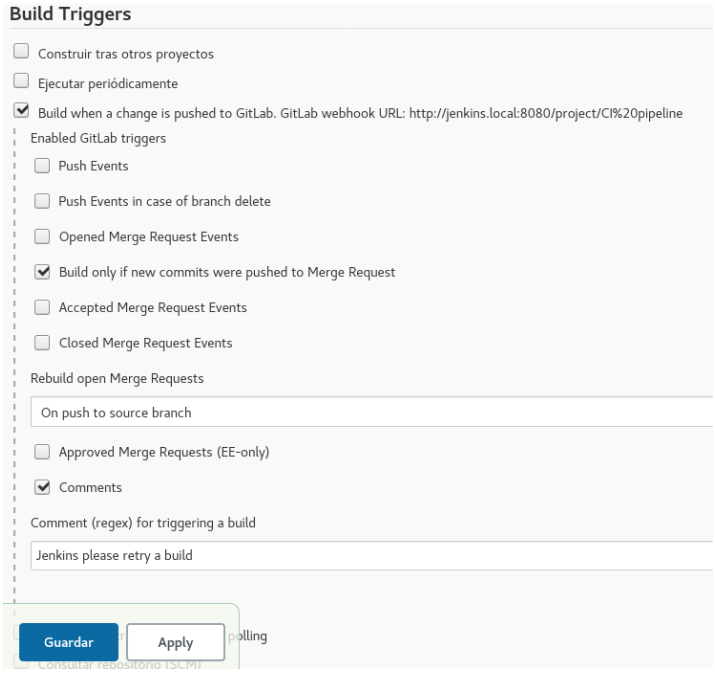


Figura 3.20 Configuración de la conexión entre GitLab y Jenkins 6

Por último nos queda configurar el evento que queremos que desencadene el proceso de CI. Para ello tenemos que configurar un webhook[23] desde nuestro proyecto de GitLab, Settings/Integrations. Desde aquí debemos seleccionar los eventos que queremos que desencadenen el proceso de CI.

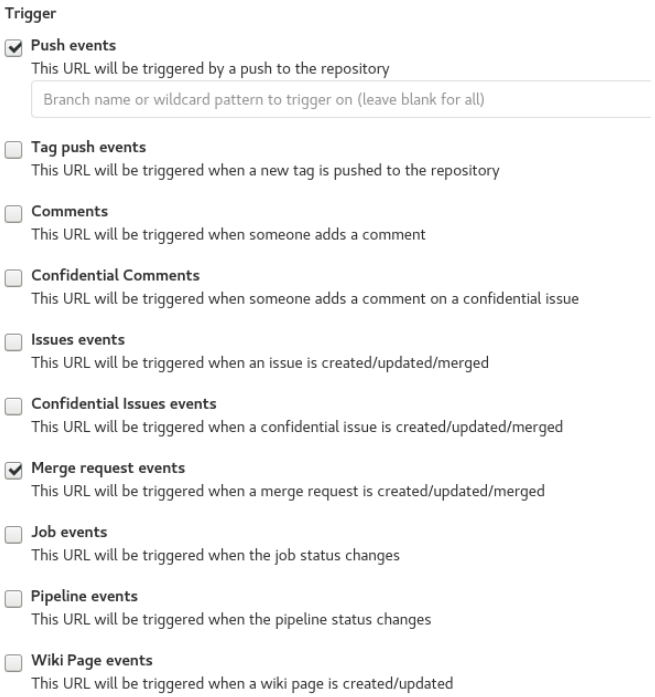


Figura 3.21 Configuración de la conexión entre GitLab y Jenkins 7

3.6.6 Automatizar la creación de imágenes

Como anteriormente para la integración de GitLab y Jenkins ya hemos creado un pipeline que empezaba su ejecución después de que hiciera un merge request. Para automatizar la creación de imágenes voy a explicar como se ha construido este nuevo pipeline.

Para poder ejecutar el pipeline que se describe posteriormente, previamente hemos tenido que instalar Docker dentro de Jenkins.

En primer lugar se ha elegido que lea nuestros pasos de ejecución desde un Jenkinsfile que hemos creado y está alojado en nuestro repositorio de GitLab.

[23] Instrucción que lanza una retrollamada HTTP, una solicitud HTTP POST que interviene cuando ocurre algo

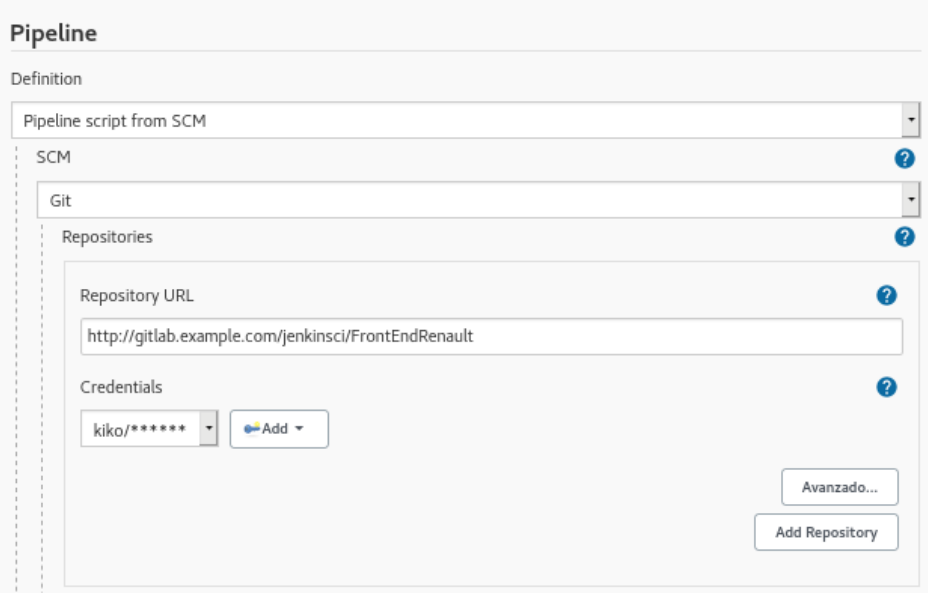


Figura 3.22 Configuración de GitLab

```
pipeline {
    agent {
        docker {
            image 'node:12-alpine3.12'
            args '-v $HOME/.m2:/root/.m2'
        }
    }
    stages {
        stage('checkout') {
            steps {
             git 'http://gitlab.example.com/jenkinsci/FrontEndTest'
            }
        }
    }
}
```

Figura 3.23 Primeras etapas de nuestro pipeline

Como nuestro proyecto de AngularJS a su vez es un proyecto que utiliza Node.js para poder ejecutar Javascript en el servidor, seleccionamos una imagen Docker que contiene Node.js. Además para evitar que con cada ejecución se descarguen de nuevo las dependencias de node, a través de los argumentos le decimos que lo almacene en la caché de nuestro pipeline.

Seguidamente el primer paso con el que debemos seguir es la descarga del código fuente del proyecto que está alojado en nuestro GitLab.

```
stage('Build Project') {
    sh 'cd FrontEnd && npm install && npm run build'
}
stage('Build image') {
    sh 'cp /tmp/Dockerfile .'
    docker.withRegistry('https://odecgandia.com:5000') {
      def angular_img = docker.build("FrontEnd:${env.BUILD_ID}")
      angular_img.push()
   }
}
stage('Deploy image on test enviroment') {
    docker.image("FrontEnd:${env.BUILD_ID}").run('-p 8888:80')
}
```

Figura 3.24 Etapas de la construcción de nuestro código desde el pipeline

Continuando con nuestro pipeline el siguiente paso que hemos hecho para poder construir nuestra nueva imagen, es la instalación y construcción del proyecto de front-end que se encontraba en nuestro repositorio.

A continuación, se construye la imagen y se sube a un registro de Docker donde se almacenarán nuestras imágenes y finalmente se despliega para poder continuar con las pruebas.

3.7 Pruebas en aplicaciones AngularJS

El código erróneo, los defectos de la aplicación y las malas prácticas de desarrollo pueden generar aplicaciones poco fiables. Escribir buenas pruebas ayudará a detectar este tipo de problemas y evitará que afecten negativamente a nuestra aplicación. Es vital que se pruebe minuciosamente la aplicación si se desea que sea sostenible y compatible durante los próximos años. El propósito principal de escribir pruebas es ayudar a proteger contra la rotura de la funcionalidad de la aplicación cuando se tenga que agregar nuevas funciones o corregir errores más adelante.

3.7.1 Descripción de las pruebas en AngularJS

La mayoría de las pruebas en AngularJS podemos encontrar hacen referencia a dos tipos de pruebas: pruebas unitarias y pruebas E2E [27].

En una aplicación AngularJS se pueden probar los siguientes conceptos:

- Componentes: trozos de código que puede usar para encapsular cierta funcionalidad que luego puede reutilizar en toda la aplicación. Los componentes son tipos de directivas (consulte el siguiente concepto), excepto que incluyen una vista o una plantilla HTML.

- Directivas: se utilizan para manipular elementos que existen en el documento o puede agregar elementos o eliminarlos del documento. Ejemplos de instrucciones incluidas con Angular son ngFor, ngIf, y ngShow.

- Tuberías: se utilizan para transformar datos. Por ejemplo, supongamos que desea convertir un número entero en moneda. Se usaría una tubería para convertir 15 en $ 15.00.

- Servicios: Se utilizarán los servicios para recuperar datos y luego inyectarlos en sus componentes.

- Enrutamiento: permite a los usuarios navegar de una vista a la siguiente mientras realizan tareas en la aplicación web.

3.7.2 Pruebas unitarias

Las pruebas unitarias sirven para probar la funcionalidad de partes o unidades básicas de código. Cada prueba unitaria sólo debe probar una parte del código fuente. Puede probar funciones, métodos, objetos, tipos, valores y más con pruebas unitarias. Las ventajas de usar pruebas unitarias son que tienden a ser rápidas, confiables y repetibles si las escribe correctamente y las ejecuta en el entorno adecuado.

Jasmine es una suite de testing que sigue la metodología Behavior Driven Development.

Karma es el test-runner, es decir, el módulo que permite automatizar algunas de las tareas de las suites de testing, como Jasmine. Karma, además, ha sido desarrollado directamente por el equipo de Angular, lo cual, en cierto modo, nos da alguna garantía de que va a seguir existiendo de aquí a un tiempo, por lo que parece una buena opción.

Con estos dos elementos tenemos preparado nuestro entorno para poder añadir tests a nuestra aplicación.

```
it('should define animate class after delaying timeout ', function() {
      $timeout.flush();
      expect(vm.classAnimation).toEqual('rubberBand');
 });
```

Figura 3.25 Ejemplo de prueba unitaria

Todo lo que se está haciendo en el código es verificar que el valor de una clase después de un timeout es el valor que queremos.

```
stage('Unit Test') {
    sh 'export CHROME_BIN=/usr/bin/chromium-browser && cd FrontEnd/tests && ng test
--code-coverage'
 }
```

Figura 3.26 Código para lanzar los tests unitarios desde el pipeline

Este trozo de código es la continuación del pipeline de Jenkins que se encargará de ejecutar las pruebas unitarias de Jasmine.

3.7.3 Pruebas E2E

Las pruebas E2E sirven para probar la funcionalidad de una aplicación simulando el comportamiento de un usuario final. Por ejemplo, es posible que tenga una verificación de prueba E2E si un modal aparece correctamente después de enviar un formulario o una página muestra ciertos elementos al cargar la página, como botones o texto.

Las pruebas E2E hacen un buen trabajo con las aplicaciones de prueba desde el punto de vista del usuario final, pero pueden ejecutarse lentamente y esa lentitud puede ser la fuente de falsos positivos que fallan en las pruebas debido a problemas de tiempo de espera. Los problemas de sincronización de las pruebas E2E hacen que sea preferible escribir pruebas unitarias en lugar de pruebas E2E siempre que sea posible.

Para escribir las pruebas E2E, se usará el marco de prueba Protractor E2E desarrollado por el equipo de Angular.

Protractor es un marco de prueba de código abierto de extremo a extremo para aplicaciones Angular y AngulaJS. Fue construido por Google por encima de WebDriver[24].

```
it('(1) HOME-Cockpit: Comprueba que hay 24 widgets en 2 grupos (24 widgetcharts)',
function() {
            expect(element.all(by.css('.widget-item')).count()).toBe(24);
            expect(element.all(by.tagName('odecui-widget-group'))
                .get(0).all(by.css('.widget-item')).count()).toBe(16);
            expect(element.all(by.tagName('odecui-widget-group'))
              .get(1).all(by.css('.widget-item')).count()).toBe(8);
});
```

Figura 3.27 Ejemplo de prueba de Protractor

Para ja ejecución de los tests E2E desde el pipeline de Jenkins se usará el siguiente código:

```
stage('E2E Tests') {
    sh 'protractor test/protractor.conf.js --
baseUrl'http://odecgandia.com/FrontEnd/Test'
}
```

Figura 3.28 Código de nuestro pipeline que ejecuta los tests E2E de protractor

[24] WebDriver es una herramienta para automatizar los test de aplicaciones

3.7.4 Pruebas unitarias frente E2E

Característica	Pruebas unitarias	Pruebas E2E
Velocidad	Suelen ser más rápidos que las pruebas E2E.	Suelen ser más lentas que las pruebas unitarias.
Fiabilidad	Suelen ser más fiables que las pruebas E2E.	Las pruebas pueden ser inestables y fallar porque pueden agotarse durante la ejecución.
Ayudando a hacer cumplir la calidad del código	Escribir pruebas puede ayudar a identificar código innecesariamente complejo que puede ser difícil de probar.	Las pruebas del navegador no ayudarán a escribir un mejor código porque está probando la aplicación como un todo desde el exterior.
Rentabilidad	Más rentable debido al tiempo del desarrollador para escribir pruebas, ejecución de pruebas y confiabilidad.	Menos rentable porque lleva más tiempo escribir las pruebas, la ejecución de las pruebas es lenta y las pruebas pueden ser inestables.
Imitando las interacciones del usuario	Las pruebas pueden imitar las interacciones del usuario, pero pueden ser difíciles de usar para verificar interacciones complejas.	Imitar las interacciones del usuario es el fuerte de las pruebas E2E y para lo que están hechas.

Figura 3.29 Tabla comparativa entre Pruebas unitarias i E2E

Analicemos cada una de estas características una por una:

- **Velocidad***:* debido a que las pruebas unitarias operan en pequeños fragmentos de código, pueden ejecutarse rápidamente. Las pruebas E2E se basan en pruebas a través de un navegador, por lo que tienden a ser más lentas.

- **Fiabilidad***:* debido a que las pruebas E2E tienden a involucrar más dependencias e interacciones complejas, pueden ser inestables, lo que puede dar lugar a falsos positivos. La ejecución de pruebas unitarias rara vez da como resultado falsos positivos. Si una prueba unitaria bien redactada falla, puede confiar en que hay un problema con el código.

- **Ayudando a hacer cumplir la calidad del código:** uno de los principales beneficios de escribir pruebas es que ayuda a reforzar la calidad del código. Las pruebas unitarias pueden ser difíciles de probar. Como regla general, si le resulta difícil escribir pruebas unitarias, su código puede ser demasiado complejo.

Escribir pruebas E2E no le ayudará a escribir código de mejor calidad per se. Debido a que las pruebas de E2E prueban desde el punto de vista del navegador, no prueban directamente su código.

- **Rentabilidad***:* debido a que las pruebas E2E tardan más en ejecutarse y pueden fallar en momentos aleatorios, ese tiempo conlleva un costo. También puede llevar más tiempo escribir tales pruebas, porque pueden basarse en otras interacciones complejas que pueden fallar, por lo que los costos de desarrollo también pueden ser más altos cuando se trata de escribir pruebas E2E.

- **Imitando las interacciones del usuario***:* con la interfaz de usuario es donde brillan las pruebas E2E. Con Protractor, se puede escribir y ejecutar pruebas como si un usuario real estuviera interactuando con la interfaz de usuario. Puede simular las interacciones del usuario mediante pruebas unitarias, pero probablemente será más fácil escribir pruebas E2E para ese propósito porque para eso están hechas.

Es importante tener ambos tipos de pruebas para probar a fondo sus aplicaciones. Puede probar gran parte de la funcionalidad que realizaría un usuario escribiendo pruebas unitarias, pero debe probar la funcionalidad clave con las pruebas E2E.

3.7.5 Pruebas de rendimiento con JMeter

Obtener un buen rendimiento del sistema es fundamental , para ello las pruebas deben de comenzar al inicio del desarrollo del software porque cuando más tarde se detecten los errores, el coste será más elevado ya que el trabajo y tiempo que se tendrá que emplear en solucionar el problema será más grande.

Nuestro entorno de pruebas debe de ser lo más parecido al de producción y no cruzarlo con el de desarrollo, de esta forma los resultados que obtendremos serán más fiables.

Estos son los diferentes tipos de prueba de rendimiento:

Prueba de carga

Una prueba de carga por regla general se realiza para contemplar el funcionamiento de una aplicación ante una cuantía previsible de peticiones. Un ejemplo de este tipo de carga puede ser el número de usuarios que se encuentren utilizando la aplicación de forma concurrente y realizando diferentes peticiones mientras la carga va aumentando. Este tipo de prueba nos mostrará el tiempo de respuesta de todas las peticiones importantes de la aplicación.

Prueba de estrés

Se utiliza generalmente para disrumpir el servicio en la aplicación. Se van duplicando el número de personas que se incorporan a ella, para efectuar una prueba de carga a la espera de que esta deje de funcionar correctamente. Con estas pruebas se determina lo sólida que puede resultar la aplicación en los momentos de más trabajo. Todo ello facilita a los administradores el comprobar si la aplicación rinde lo bastante si la carga real es mayor que la carga esperada.

Prueba de estabilidad

Habitualmente se hacen para diagnosticar si la aplicación es capaz de soportar de forma continuada una carga prolongada y de esta forma poder comprobar si hay algún fallo que provoque el uso de más recursos en la aplicación.

Prueba de picos

La prueba de picos, como indica su nombre, contempla la conducta del sistema al modificarse el número de personas que están utilizando la aplicación, tanto cuando baja el número de usuarios como cuando se realizan alteraciones rotundas en su carga. Se recomienda que esta prueba se realice de forma automática utilizando software. Este software debe permitir variar el número de usuarios mientras el equipo de desarrollo monitoriza estas pruebas para comprobar su correcto funcionamiento y resultados obtenidos.

En nuestro caso, utilizaremos pruebas de carga que hemos creado previamente en el programa de JMeter.

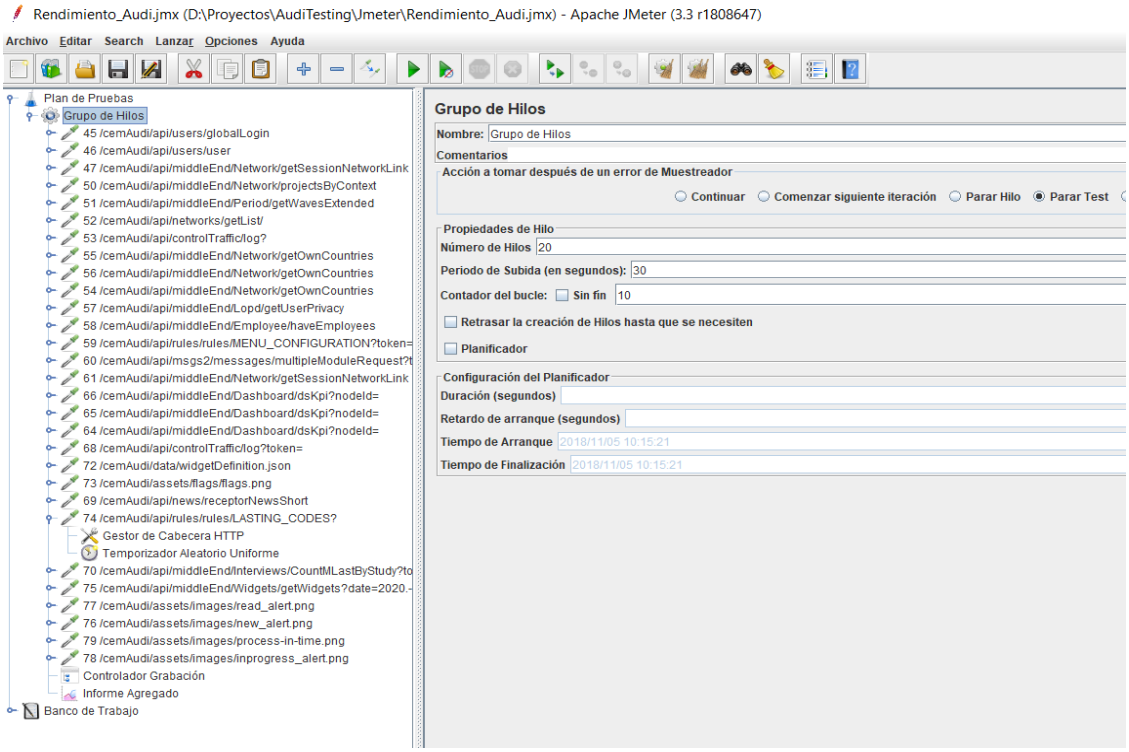


Figura 3.30 Captura de Jmeter con las peticiones a probar

Las pruebas se lanzarán desde Jenkins a través del siguiente código:

```
stage ('JMeter Test')
  sh 'jmeter/apache-jmeter-3.3/bin/jmeter.sh -n -t /tests/FicheroTest.jmx
}
```

Figura 3.31 Código de nuestro pipeline que ejecuta las pruebas de JMeter

3.8 Despliegue de la imagen

Para el despliegue de las imágenes se ha decidido usar el proveedor de Cloud público Amazon Web Services (AWS) [28].

3.8.1 Amazon Web Services

Amazon Web Services es una colección de servicios de computación remota que juntos conforman una plataforma de computación en la nube, ofrecida a través de Internet. Los más fundamentales y conocidos de estos servicios son Amazon EC2[25] y Amazon S3[26] pero hay muchos más.

3.8.2 Elastic Container Registry

Amazon Elastic Container Registry (ECR) es un registro de contenedores de Docker totalmente administrado que permite a los desarrolladores almacenar, administrar, compartir e implementar fácilmente imágenes y máquinas de contenedores en cualquier lugar. Con la herramienta de Amazon ECR, ya no se necesita usar el repositorio de contenedores ni preocuparse por expandir la infraestructura. Amazon ECR aloja sus imágenes en una arquitectura de alto rendimiento y alta disponibilidad, lo que le permite implementar imágenes de manera confiable para aplicaciones de contenedor. Puede compartir software contenedor de forma privada dentro de su organización, o puede compartir software contenedor públicamente a nivel mundial para que cualquiera pueda descubrirlo y descargarlo.

Beneficios:

- Reducir el esfuerzo con un registro completamente administrado.
- Descargar y compartir de forma segura imágenes de contenedores.
- Brindar acceso rápido y de alta disponibilidad.
- Simplificar el flujo de trabajo de implementación.

Para el despliegue de las imágenes desde el pipeline de Jenkins se ha usado el siguiente código:

```
stage('Deploy image on AWS') {
    steps{
        script{
            docker.withRegistry("$DOCKER_AWS_SERVER", "ecr:eu-central-1:" +
            "$JENKINS_AWS_CREDENTIAL) {
            dockerImage.push()
            }
        }
    }
}
```

Figura 3.32 Código de nuestro pipeline que hace el deploy final en el servidor de AWS

[25] Amazon Elastic Compute Cloud es una plataforma en la nube que permite a los usuarios alquilar computadores virtuales en los cuales pueden ejecutar sus propias aplicaciones.
[26]Servicio ofrecido por Amazon Web Services que proporciona almacenamiento de objetos a través de una interfaz de servicio web.

4. Contribuciones personales

En este cuarto capítulo vamos a analizar en detalle cómo se han desarrollado las pruebas de nuestros proyectos. Esta es la parte donde más he trabajado en la empresa ya que estuve en el equipo de testeo.

Como ya hemos visto anteriormente en la sección 2.6 las pruebas son esenciales en un proyecto de integración continua, ayudan a obtener un software estable y funcional a un proyecto que continuamente se le están haciendo cambios.

4.1 Realización de las pruebas manuales

Una prueba manual es un tipo de comprobación que se realiza sobre un software donde los testers realizan las comprobaciones de los casos de uso manualmente sin usar ninguna herramienta de test automática.

La prueba manual es la más primitiva de todos los tipos de prueba y ayuda a encontrar errores en el sistema de software. Cualquier cambio en nuestra aplicación debe probarse manualmente antes de automatizar las pruebas. La prueba manual es más costosa pero es necesaria para verificar la viabilidad de la automatización.

Las pruebas manuales, antes de implementar los tests automáticos, eran nuestra principal herramienta para detectar los errores en nuestros despliegues.

Previamente se levantaba un servidor de prueba con el código a probar antes de ser desplegado en los servidores de producción.

Estas pruebas consisten en la realización de cuestionarios que se entregaban a través de documentos de texto o excel y que previamente habían sido redactados por el equipo de desarrollo.

Las persona encargada de realizar las pruebas debía ir rellenando fila por fila el estado de las tareas a comprobar, donde además de las novedades incorporadas, se comprobaba toda la web. En caso de detectar un error la persona que detectaba el error debía informar al desarrollador que se había encargado de realizar la tarea para que solucionara el problema y volvieran a realizar las pruebas pertinentes.

El tiempo que se perdía entre la detección de errores, informar al desarrollador para corregir los errores y volver realizar el despliegue en el servidor de prueba para volverse a probar, era un tiempo muy grande.

Ante la necesidad de reducir estos tiempos, y evitar la comprobación de toda las web, se propuso el desarrollo de los tests automáticos. Esto no implicaba dejar de hacer pruebas manuales, pero sí que se reducían la cantidad de pruebas a realizar.

A continuación exponemos un ejemplo de cuestionario.

[1] HOME:COCKPIT		TEST STATUS
USER	TEST	
xxx/xxx	Verify there are 3 Project labels and After Sales jump	OK
	Verify Audi Sales jump.	OK
	Verify Audi Used Cards jump.	OK
	Verify the color of the first Project label.	OK
	Verify the color of the second Project label.	OK
	Verify the color of the third Project label.	OK
	Verify there are 4 widgets in the Aftersales Audi Project.	OK
	Verify there are 4 widgets in the Sales Audi Project.	OK
	Verify there are 4 widgets in the UsedCars Audi Project.	OK
	Verify the Widgets of Aftersales Audi display Top 5 countries.	
	Verify the Widgets of Sales Audi Audi display Top 5 countries.	
	Verify the Widgets of UsedCars Audi display Top 5 countries.	
	Verify "Show all Countries" button works.	
	Verify the Widgets display 6M by default for Aftersales.	
[2] HOME:DASHBOARD		
USER	TEST	
xxx/xxx	Verify Dispersion Map Widget is displayed.	
	Open the KPI Combo and Verify there is a Global group and 4 KPIs to Choose.	
	Take a snapshot.	
	Change the filter params and verify filter view pills match them.	
	Click any point in the chart and verify leads you to country Level.	

Figura 4.1 Tabla de ejemplo de cuestionario

4.2 Realización de las pruebas E2E

El framework con el que se han escrito las pruebas E2E es Jasmine, el mismo que se ha usado para las pruebas unitarias. Sin embargo, para ejecutar estas pruebas se va a utilizar Protractor.

Protractor es un programa en Node.js, que utiliza Selenium para permitir a las pruebas interactuar con el navegador. La idea de las pruebas E2E es interactuar con la aplicación de la misma manera que haría un usuario en su navegador.

De esta manera, es posible probar de principio a fin una funcionalidad, definiendo una sucesión de interacciones a realizar. Protractor ejecuta a continuación esta sucesión de acciones, en los navegadores definidos, lo que es muy práctico, porque esto permite identificar automáticamente las diferencias entre los navegadores.

Las pruebas E2E no sustituyen a las pruebas unitarias por muchas razones, sin embargo son complementarias. Estos dos tipos de pruebas son muy importantes.

4.2.1 Configuración de Protractor

El siguiente paso es configurar Protractor para lo que creamos el fichero protractor.pro.conf.js en la raíz del proyecto con el siguiente contenido:

```
'use strict';
var HierarchicalHTMLReporter = require('protractor-html-hierarchical-reporter');
exports.config = {
  capabilities: {
    'browserName': 'chrome'
  },
  specs: ['tests/dailyTestPro.e2e.js'],
  rootElement: 'html',
  resultJsonOutputFile: 'e2e.json',
  framework: 'jasmine',
    jasmineNodeOpts: {
    showColors: false,
    defaultTimeoutInterval: 300000,
    isVerbose: true,
    includeStackTrace: false
  },
  onPrepare: function() {
      jasmine.getEnv().addReporter(
        new HierarchicalHTMLReporter({
          savePath: 'e2e/reports/dailyTestPro/',
          filePrefix: 'index'
        })
      );
  },
  allScriptsTimeout: 400000
};
```

Figura 4.2 Código de configuración de Protractor

En este fichero le indicamos la configuración a utilizar, estamos indicando en qué navegador vamos a ejecutar las pruebas, en este caso, Chrome, dónde van a residir nuestros ficheros de tests dentro de la estructura del proyecto y la configuración de Jasmine, para que muestre colores y establecer un timeout máximo de espera.

Además se ha añadido la configuración de **protractor html hierarchical reporter** que como su nombre indica es una herramienta para crear reportes. Con él vamos a obtener capturas de imágenes de los resultados de la ejecución.

Para ejecutar los tests de manera local utilizamos la siguiente instrucción donde indicamos el fichero de configuración y la URL donde se quiere que se realicen las pruebas.

```
protractor e2e/protractor.pro.conf.js --baseUrl https://odecgandia.com:1234
```

Figura 4.3 Comando para la ejecución de Protractor

4.2.2 Redacción de los tests de Protractor

La redacción de los tests ha consistido en convertir las pruebas manuales que hemos visto anteriormente, en código: Para ello se ha usado el framework Jasmine[29].

```
describe('HOME:COCKPIT ::', function(){
    it('Verify there are 3 Project labels and Audi After Sales jump.', function() {
        helpers.log('-----------------------------------');
        helpers.log('(1)-Verify there are 3 Project labels and After Sales jump');
        helpers.clickLink('[uib-tooltip="Jump to After Sales"]')
        .then(function(){
            expect(element.all(by.css('.button-link')).count()).toBe(3);
        });
    });
    it('Verify Audi Sales jump.', function() {
        helpers.log('(2)-Verify Audi Sales jump.');
        helpers.scrollToTop();
        helpers.clickLink('[uib-tooltip="Jump to Sales"]')
        .then(function(){
            expect(element.all(by.css('.button-link')).count()).toBe(3);
        });
    });
    it('Verify Audi Used Cars jump.', function() {
        helpers.log('(4)-Verify Audi Used Cars jump.');
        helpers.clickLink('[uib-tooltip="Jump to Used Cars"]')
        .then(function(){
            expect(element.all(by.css('.button-link')).count()).toBe(3);
        });
    });
    it('Verify the color of the first Project label.', function() {
        helpers.log('(5)-Verify the color of the first Project label.');
        expect(element.all(by.css('.scroll-buttons .btn-to-scroll')).get(0)
            .getCssValue('background-color')).toEqual('rgba(187, 10, 48, 1)');
    });
});
```

Figura 4.4 Instrucciones de una prueba de E2E

Además, para facilitar la construcción de las pruebas, se ha creado unas funciones de ayuda, estas funciones estarán disponibles en un fichero llamado “helpers”.

A continuación, mostramos la lista de las funciones que se han creado para facilitar la escritura de los tests, estas funciones evitan crear código repetitivo, por este motivo se ha decidido crear esta solución.

```
module.exports = {
    login: login,
    clickLink: clickLink,
    clickLinkByText:clickLinkByText,
    askLocation: askLocation,
    scrollIntoView: scrollIntoView,
    scrollToTop: scrollToTop,
    goToThroughMenu: goToThroughMenu,
    goToThroughMenuText : goToThroughMenuText,
    verifyNodeSelected: verifyNodeSelected,
    verifyTextNodeSelected: verifyTextNodeSelected,
    select2ByText: select2ByText,
    newTabUrl: newTabUrl,
    goToSettings: goToSettings,
    waitUntilExist: waitUntilExist,
    log:log,
    waitForElement: waitForElement,
    setFilterCloseOptions:setFilterCloseOptions,
    setFilter:setFilter,
    setNodeFilter:setNodeFilter,
    getScale:getScale
};
```

Figura 4.5 Listado de instrucciones de ayuda del archivo helpers.js

4.2.3 Resultado de los tests de Protractor

Como hemos dicho anteriormente el resultado de las pruebas que queremos obtener es un reporte HTML donde además de ver los errores podremos ver las capturas de la navegación y ver de forma visual y con más detalle los errores.

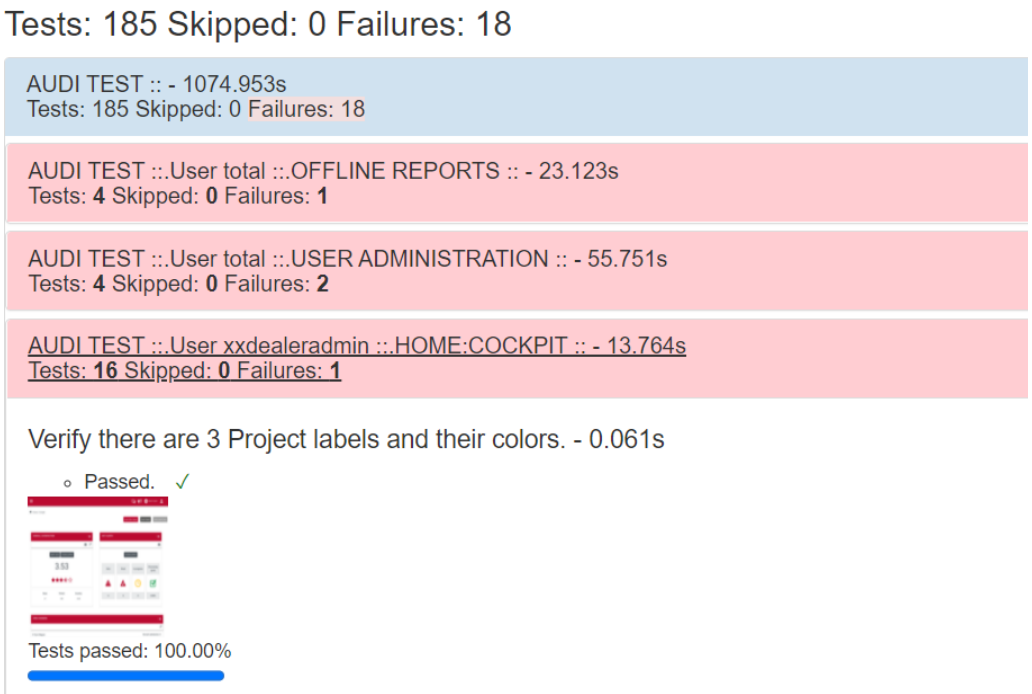


Figura 4.6 Report de Protractor con HTMLHierarchicalReporter

Estos reportes también serán enviados a través de un correo desde el pipeline de Jenkins donde adjuntamos una URL donde se encuentran alojados los ficheros correspondientes a los reportes de la ejecución.

```
catch (err) {
        currentBuild.result = "FAILURE"
            mail body: "La construcción del proyecto ha fallado, aquí puedes ver el
            resultado de la ejecución: ${env.BUILD_URL}" ,
            from: 'jenkins@odec.com',
            subject: 'Resultado de la construcción en Jenkins',
            to: 'frafusju@inf.upv.es'
        throw err
    }
```

Figura 4.7 Código para enviar los reportes

4.3 Pruebas de JMeter

4.3.1 Grabación de peticiones con JMeter

En primer lugar, para obtener el listado de las peticiones que queremos probar, usaremos la herramienta Servidor Proxy HTTP de JMeter, mediante esta herramienta podemos iniciar una grabación que irá obteniendo un listado de peticiones [30].

En segundo lugar, para poder realizar la captura de peticiones, debemos navegar con un navegador web por la web de la que queremos obtener las peticiones.

En nuestro caso, para realizar la grabación de peticiones, iniciamos el servidor proxy desde JMeter, y con el navegador Chrome desde una pestaña de incognito, accedemos a la web desde la que queremos obtener las peticiones y navegamos a través de ella por los lugares que se quieren probar.

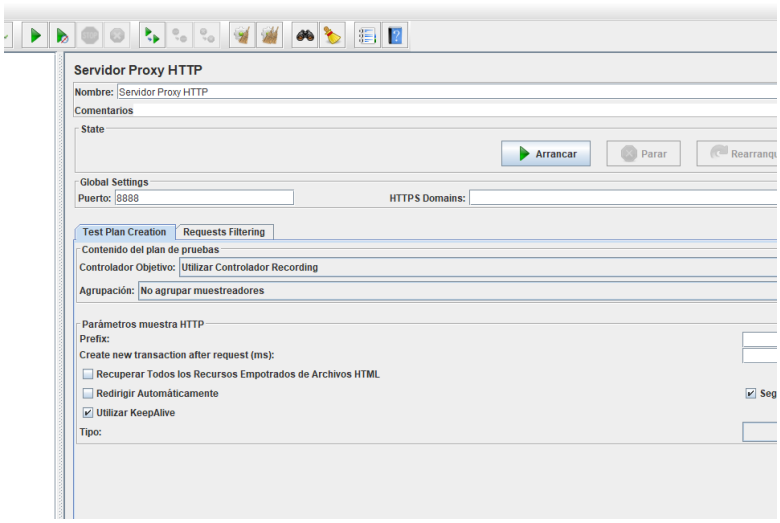


Figura 4.8 Servidor Proxy HTTP de JMeter

Una vez hemos obtenido el listado de peticiones, el siguiente paso a configurar es el paso de datos entre peticiones. Para ello hemos usado las herramientas de JSON Extractor para las peticiones que devuelven un resultado JSON o la herramienta Extractor de Expresiones Regulares para las funciones que devuelven una expresiones regulares. De esta forma, usando una herramienta u otra vamos a poder extraer los valores de la respuesta y almacenar en una variable el resultado para poder usarlo posteriormente en el resto de peticiones a probar.

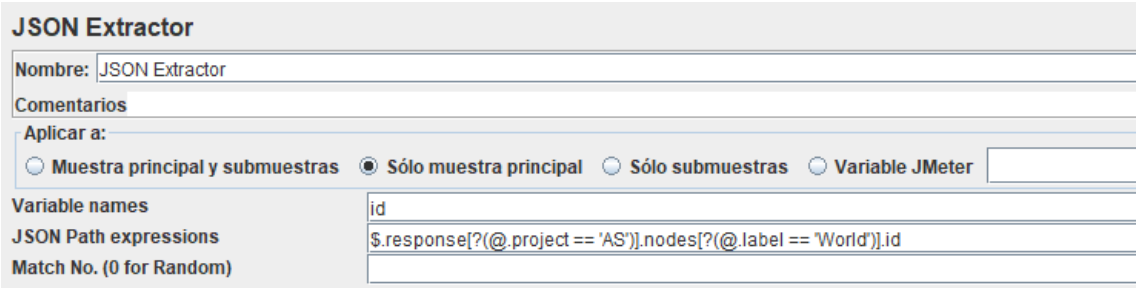


Figura 4.9 JSON Extractor de Jmeter

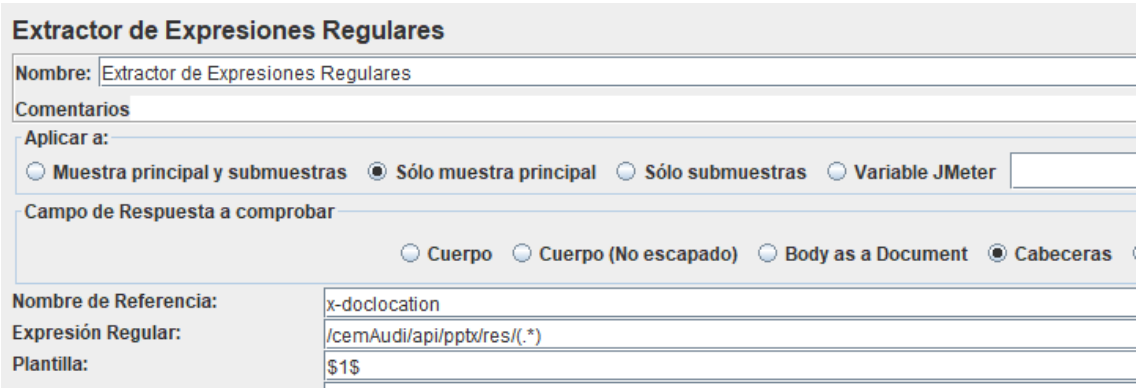


Figura 4.10 Extractor de Expresiones Regulares de JMeter

Hay dos propósitos por los que hemos decidio usar JMeter:

- Prueba de carga para ver que la web, con un número normal de usuarios, aguanta con fluidez sin bajar el rendimiento.
  Para este tipo de prueba, en la configuración de la herramienta Grupo de Hilos de JMeter, vamos a hacer que se creen 20 hilos, que correspondía a 20 usuarios, estos usuarios se crearían de forma progresiva cada 20 segundos.

- Prueba de descarga de ficheros, a través de estas pruebas se comprueba que la descarga de ficheros desde la web funciona correctamente. Para este tipo de prueba, en la configuración de la herramienta Grupo de Hilos de JMeter, solamente se crea un hilo que como si correspondiera a un usuario.

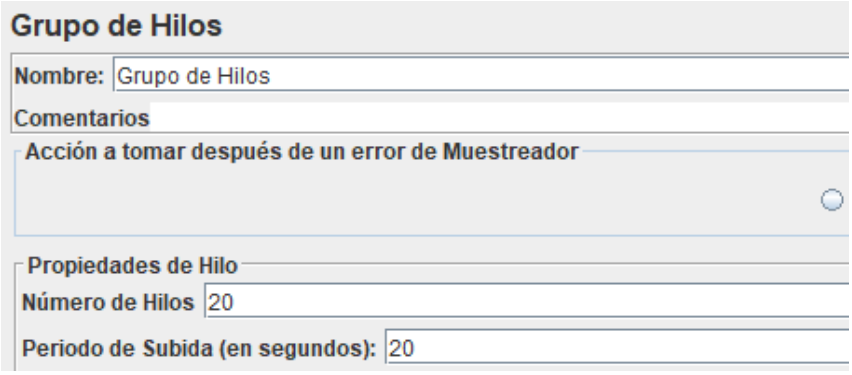


Figura 4.11 Configuración de Grupo de Hilos de JMeter

4.3.3 Otras ejecuciones de JMeter

Además de utilizar los tests de carga, como prueba de testeo en el el proceso de CI desde Jenkins la cual ya hemos descrito anteriormente en la sección 3.7.5. Utilizaremos estos tests para comprobar los tiempos de respuesta de las peticiones desde diferentes países.

Estos entornos de prueba, corresponden a diferentes máquinas remotas que se encuentran ubicadas físicamente en otros países. De esta forma se podía comparar los tiempos de la respuesta de las peticiones desde diferentes países.

La ejecución de estas pruebas se ha programado para que se realicen cada 20 minutos desde las máquinas remotas. De esta forma podemos obtener la media de los tiempos de respuesta de todas las ejecuciones durante un día.

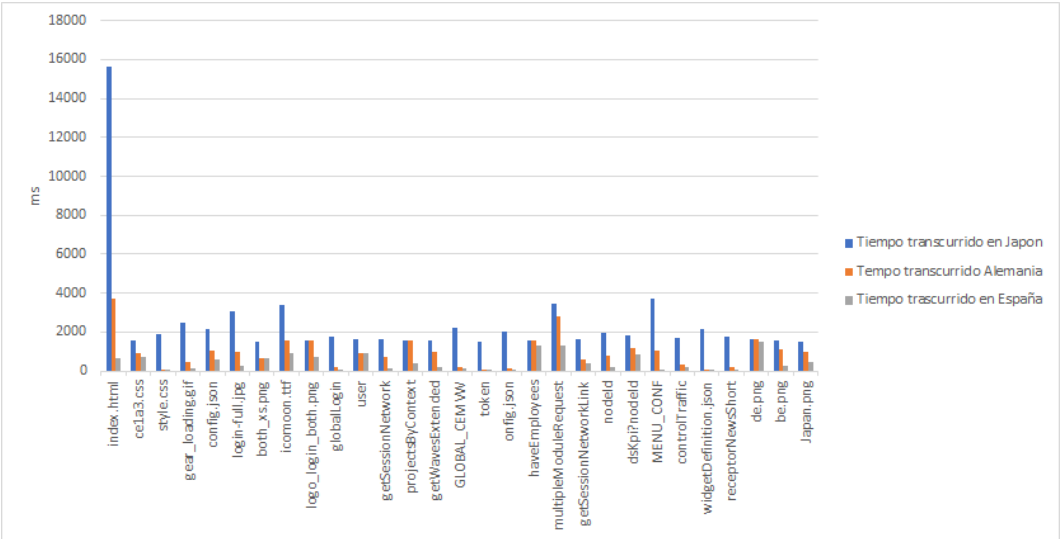


Figura 4.12 Resultados de las ejecuciones de JMeter

Como vemos en el gráfico anterior se detectaron problemas en las peticiones que se hacen con las máquinas remotas ubicadas en Japón, los tiempos de respuesta obtenidos en las peticiones individualmente son mayores fuera de la UE por este motivo, se puso en estudio para mejorar este problema en un futuro.

5. Conclusión y Trabajos Futuros

5.1 Conclusión

Este trabajo ha tenido como objetivo mejorar el entorno de desarrollo usado para el desarrollo de aplicaciones web AngularJS. Anteriormente en la empresa se sufrían diferentes problemas a la hora de programar y desplegar las aplicaciones web. Además la metodología utilizada hacía falta mejorarla, ya que los tiempos entre desarrollo y corrección de errores eran muy elevados por que al desplegar la aplicación se tenía que probar la mayoría de funcionalidades manualmente.

Con una metodología ágil donde la CI está en funcionamiento se ha puesto fin a la mayoría de problemas que encontrábamos. Con esta metodología, se pretende que el equipo de desarrollo siga las mismas pautas y patrones de trabajo a la hora de desarrollar aplicaciones independientemente del proyecto en el que se esté trabajando.

Anteriormente el desarrollo de software era mucho más lento que el actual, donde las comprobaciones y el despliegue se hacían de forma mucho más manual. Todo el desarrollo ha pasado a ser colaborativo, donde intervienen diferentes personas al mismo tiempo y se ha automatizado al máximo el proceso de testeo y despliegue del software.

En el caso de detectar algún fallo posterior al despliegue el coste era elevado ya que todas las comprobaciones manuales de la aplicación se tenían que repetir, en algunas ocasiones varias veces hasta que se solucionara el problema. Con la puesta en funcionamiento del testeo automático estos problemas se han visto solucionados lo que produce una disminución de tiempo y una reducción del número de fallos.

5.2 Trabajos futuros

Un trabajo pendiente que se quiere realizar, es la migración de AngularJS a Angular, esto requiere gran trabajo ya que se debe de migrar pieza por pieza.

Se conoce que AngularJS tiene problemas de rendimiento como de estructura, además la versión 1.7 es la última y pronto dejará de tener soporte. Por este motivo Google lanzó Angular para poner solución a estos problemas. Angular usa lenguaje

Existen diferentes maneras de migrar aplicaciones a AngularJs. Se puede hacer mediante herramientas como ngUpgrade, Angular Elements o desarrollando nuestros propios componentes. Se puede migrar sin parar el desarrollo, pero no va a ser un trabajo rápido.

Otra tarea a solucionar son los problemas de latencia fuera de Europa. Mediante las pruebas de JMeter ejecutadas desde Japón y Alemania, se pudo ver que los tiempos

de respuesta fuera de la UE eran mucho más elevados a la media obtenida desde Europa.

Para solucionar este problema se propuso la propuesta de replicar los servidores en otros países mediante el uso de Kubernetes.

Bibliografía

[1] ODEC. (2018). ODEC. Servicios. from https://www.odec.es/es/servicios/

[2] Ruiz, J.Jesus Maria Zavala. "¿Por Qué Fracasan los Proyectos de Software?; Un Enfoque Organizacional." Congreso Nacional de Software Libre 2004. https://www.researchgate.net/publication/283546859_Por_Que_Fracasan_los_Proyectos_de_Software_Un_Enfoque_Organizacional.

[3] GABRIEL MANCUZO. "Modelo cascada y espiral: comparación para elegir el adecuado." Compara Software, 2021, https://blog.comparasoftware.com/modelo-cascada-y-espiral/.

[4]María N. Moreno García. (2002). Modelos de proceso del software. In (pp. 7-43). Departamento de Informática y Automática Universidad de Salamanca Universidad de Salamanca.Edicio

[5] WiseRatel. (2020, Julio 5). Git – Sistemas De Control De Versiones. Swift By Coding. Retrieved Julio 07, 2021, from https://swiftbycoding.dev/git/sistemas-de-control-de-versiones/

[6] Tichy, W. F. (1985). RCS—A system for version control. Software: Practice and Experience, 15(7), 637-654.

[7] Apache® Subversion®. Apache Subversion. from https://subversion.apache.org/

[8] Wingerd, L. (2005). Practical Perforce. " O'Reilly Media, Inc.".

[9] Techtva. (2010, Septiembre 9). Git, Mercurial and Bazaar – A Comparison. Retrieved Julio 7, 2021, from http://www.techtatva.com/2010/09/git-mercurial-and-bazaar-a-comparison/

[10] Git. (2014). Pro Git. (2nd ed.). Chacon, S., & Straub, B. https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Una-breve-historia-de-Git

[11] O'Sullivan, B. (2009). Mercurial: The Definitive Guide: The Definitive Guide. " O'Reilly Media, Inc.".

[12] Bazaar. History Of Bazaar. http://wiki.bazaar.canonical.com/HistoryOfBazaar

[13] Red Hat. ¿Qué son la integración/distribución continuas (CI/CD)? https://www.redhat.com/es/topics/devops/what-is-ci-cd

[14] Cloud Bees. What is Jenkins. https://www.cloudbees.com/jenkins/what-is-jenkins

[15] Travis, C. I., & Hroncok, M. (2018). Travis ci. Source: https://travis-ci. org, 17.

[16] Henn Idan. (2017, Octubre 24). Jenkins vs Travis CI vs Circle CI vs TeamCity vs Codeship vs GitLab CI vs Bamboo. https://www.overops.com/blog/jenkins-vs-travis-ci-vs-circle-ci-vs-teamcity-vs-codeship-vs-gitlab-ci-vs-bamboo/

[17] IONOS by 1&1. (2019, Junio 9). *Virtualización con contenedores Docker: alternativas*. https://www.ionos.es/digitalguide/servidores/know-how/alternativas-a-los-contenedores-en-docker/

[18] Jangla, K. (2018). Docker Basics. In Accelerating Development Velocity Using Docker (pp. 27-53). Apress, Berkeley, CA.

[19] Jet Brains. (n.d.). Automated Testing for CI/CD. https://www.jetbrains.com/teamcity/ci-cd-guide/automated-testing/

[20] Cohn, M. (2010). *Succeeding with agile: software development using Scrum*. Pearson Education.

[21] Montero, B. M., Cevallos, H. V., & Cuesta, J. D. (2018). Metodologías ágiles frente a las tradicionales en el proceso de desarrollo de software. *Espirales revista multidisciplinaria de investigación*, *2*(17).

[22] Ricardo Caicedo. (2016, Mayo 21). *¿Qué es GIT y cómo funciona?* https://www.pragma.com.co/academia/lecciones/que-es-git-y-como-funciona

[23] Duvall, P. M., Glover, A., & Matyas, S. (2007). Continuous integration : improving software quality and reducing risk (1st edition). Addison Wesley.

[24] Leszko, R. (2017). Continuous delivery with Docker and Jenkins : delivering software at scale (1st edition). Packt.

[25] Docker Docs. Install Docker Engine on CentOS. https://docs.docker.com/engine/install/centos/

[26] Docker Docs. Install Docker Compose. https://docs.docker.com/compose/install/

[27] Cohn, C., Nishina, C., Palmer, J., & Giambalvo, M. (2019). Testing Angular Applications [electronic resource] (1st edition). Manning Publications.

[28] Amazon Web Services. Available in: http://aws.amazon.com/es/ec2

[29] Jasmine.. Jasmine Behavior-Driven JavaScript. https://jasmine.github.io/pages/docs_home.html

[30] Apache JMeter. User's Manual. https://jmeter.apache.org/usermanual/index.html