Integración y entrega continua (CI/CD) con Jenkins

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Autor: Juan Carlos Mercedes Brito

Consultor: Miguel Martín Mateo

Enero 2022

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

INTRODUCCIÓN

1.1 Motivación

Durante el ciclo de vida del desarrollo de software, muchas empresas o proyectos, continúan utilizando tecnologías o herramientas anticuadas, provocando con ello que el equipo de desarrollo, y el de operaciones, realicen tareas repetitivas. Además, antes de llegar a su etapa final, debe atravesar diferentes etapas (o entornos, como son el de desarrollo, el de pruebas, operaciones, producción, etc.), lo que lo puede provocar que éste sea más proclive a fallas o modificación, y no funcionar correctamente o como se esperaba a la hora de desplegarlo en su entorno final.

Para poder mitigar estos retrasos o fallos, estos proyectos deben adoptar el uso de nuevas metodologías y procedimientos, en los que se automaticen procesos que faciliten el flujo de trabajo, consiguiendo así reducir tiempos, costes, riesgos y garantizar la calidad final del producto, requisito indispensable.

El objetivo técnico de la Integración Continua es establecer una forma coherente y automatizada de crear, empaquetar y probar aplicaciones. Con la consistencia del proceso de integración, es más probable que los equipos realicen cambios de código con mayor frecuencia, lo que conduce a una mejor colaboración y calidad del software. El Despliegue Continuo comienza donde termina la integración continua. El Despliegue Continuo (CD) automatiza el despliegue de aplicaciones a diferentes entornos de infraestructura. La mayoría de los equipos trabajan con varios entornos distintos a los de producción, como los entornos de desarrollo y pruebas, y el Despliegue Continuo garantiza que haya una forma automatizada de enviarles cambios de código.

1.2 Objetivo

Este proyecto pretende explicar de una manera clara y expositiva, la importancia y beneficios que tiene para una organización dedicada al desarrollo de software, la integración de técnicas de Integración Continua y Entrega Continua, ya que éstas ayudan a optimizar todas las tareas que conllevan los procesos del ciclo de vida del software, agilizando el despliegue a producción, garantizando un producto final de mayor calidad y de menor coste económico. Se describen las ventajas de implementar estas técnicas, se especifican las diferentes herramientas utilizadas en este ámbito, y se profundiza en el uso de Jenkins, una de las herramientas de integración continua más ampliamente utilizada.

Implementando estas técnicas, se automatizan tareas como la obtención y compilación del código fuente, tests unitarios, de integración e incluso funcionales, despliegues, así como la generación de paquetes RPM del servicio o del producto, desplegarlo en la nube o subirlo a un repositorio para que el cliente pueda disponer de él y realizar sus validaciones o propósito final, convergiendo así en un sistema más ágil. Además se realizará una introducción a la metodología *agile* debido a su analogía con la integración continua.

A continuación, se plasma un caso práctico/real en el que se muestran todos los conceptos explicados, aplicando CI/CD con Jenkins, en la que se exhibe como se automatiza las diferentes tareas que engloban desde la fase de desarrollo hasta la fase final (producción), proporcionando información a los diferentes integrantes que participan en la creación del software.

Finalmente se formulan las conclusiones y beneficios de adoptar esta metodología, en comparación con otras ya obsoletas, en base a los resultados obtenidos tras implantar el sistema de Integración Continua.

ESTADO DEL ARTE

2.1 Antecedente

Inicialmente para el desarrollo de software se utilizaban conjuntos de procedimientos o instrucciones que seguían una secuencia lógica, donde cada etapa dependía de que se culmine la etapa anterior. Seguir este paradigma implica que las etapas realizadas no eran autónomas de las siguientes, sino que se crea una dependencia estructural y en caso de un error en alguna de ellas, se retrasaría todo el proyecto [1].

Algunas de estas metodologías clásicas son:

- Modelo Lineal o Cascada Pura (Waterfall).

Este modelo es un proceso de desarrollo secuencial, en el que el desarrollo de software se concibe como un conjunto de etapas que se ejecutan una tras otra. Concibe las fases de desarrollo como procesos independientes en el tiempo, es decir, no se pueden realizar de manera simultánea; cada fase empieza cuando se ha terminado la fase anterior y para poder pasar a otra fase es necesario haber conseguido todos los objetivos de la etapa previa. Las etapas de este paradigma se desarrollan en forma secuencial y cuando se detecta algún error en alguna etapa, lo más probable será abandonar todo lo avanzado y regresar a la etapa primera de análisis de requisitos del sistema; pues, aunque la vuelta atrás por etapas es posible, ésta demanda mucho esfuerzo y puede terminar en el colapso, como se denota en la siguiente imagen:

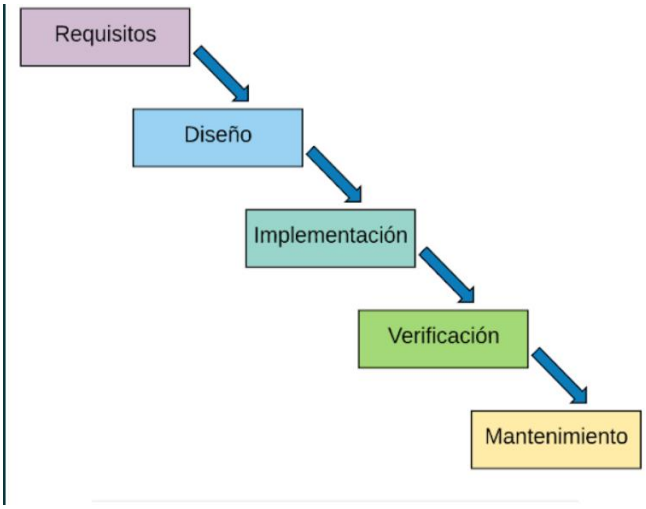


Ilustración 1. Las etapas del modelo en cascada

- Modelo evolutivo.

Este enfoque aúna las actividades de especificación, desarrollo y validación. Son modelos iterativos que permiten desarrollar versiones cada vez más completas, hasta llegar al objetivo final marcado por el cliente. Un sistema inicial se desarrolla rápidamente a partir de especificaciones abstractas. Éste se refina basándose en las peticiones del cliente para producir un sistema que satisfaga sus necesidades [2].

- Modelo basado en componentes.

Este modelo se basada en la reutilización al desarrollo de sistemas de software, es decir, permite reutilizar un número significativo de código preelaborado que permiten realizar diversas tareas concretas, conllevando a diversos beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo y el mayor retorno sobre la inversión. Se enfoca, básicamente, en integrar estos componentes en el sistema más que en desarrollarlos desde cero.

- Prototipos

Este paradigma de construcción comienza con la recolección de requisitos. El desarrollador y el cliente encuentran y definen los objetivos globales para el software, identifican los requisitos conocidos, y las áreas del esquema en donde es obligatoria más definición. Consiste en construir rápida y económicamente un sistema experimental para que lo evalúen los usuarios finales. Interactuando con el prototipo, los usuarios pueden darse una mejor idea de sus requerimientos de información. Este modelo resulta una alternativa para el desarrollo rápido de aplicaciones de software; pues el analista acorta en tiempo entre la determinación de los requerimientos de información y la entrega de un sistema funcional, además que el usuario podrá modificar y depurar sus requerimientos conforme avance el desarrollo del proyecto.

- Espiral

La forma de espiral representa una iteración (repetición) de procesos que, a medida que se van entregando prototipos y éstos son revisados por los clientes o usuarios finales, el tiempo empleado para desarrollar la próxima versión es cada vez mayor. Cada división recibe el nombre de región de tareas. el software se desarrolla en una serie de versiones incrementales. Durante las primeras iteraciones la versión incremental podría ser un modelo en papel o un prototipo, y durante las últimas iteraciones se producen versiones cada vez más completas del sistema diseñado. Una característica clave del desarrollo en espiral es la minimización de los riesgos en el desarrollo de software, lo que podría resultar en un aumento de los costes totales, más esfuerzo y un lanzamiento retardado.

- Incremental

El modelo incremental es una unión de las mejores funcionalidades del modelo de cascada y del modelo de prototipos. A medida que se presenta un prototipo se produce un “incremento”, que es una iteración del proceso anterior, pero aplicando las experiencias aprendidas del proceso anterior. A diferencia del modelo de prototipos, los prototipos de este modelo están orientados a ser operacionales en cada incremento y no ser solo una “previa” de cómo sería el sistema en su versión final. Un ejemplo gráfico se muestra en la siguiente imagen:

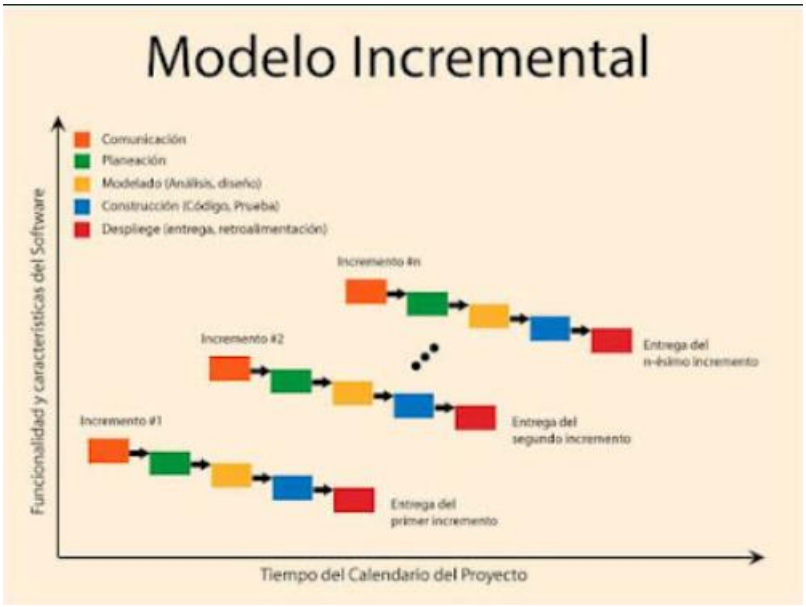


Ilustración 2. Modelo incremental

- Modelo de desarrollo concurrente

Es un modelo de tipo de red donde todas las personas actúan simultáneamente o al mismo tiempo. Provee una meta-descripción del proceso del software. Tiene la capacidad de describir las múltiples actividades del software ocurriendo simultáneamente. La mayoría de los modelos de procesos de desarrollo del software son dirigido por el tiempo; cuanto más tarde sea, más atrás se encontrará en el proceso de desarrollo. Ese modelo está dirigido por las necesidades del usuario, las decisiones de la gestión y los resultados de las revisiones [3].

2.2 Contexto

Como respuesta a las limitaciones de modelos clásicos de desarrollo de software surgen las metodologías Ágiles. Las metodologías Ágiles son aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno. Las metodologías ágiles son aquellas que básicamente se basan en interactuar directamente con el cliente desarrollando y entregando avances parciales del trabajo hasta tener la versión final del mismo. Se fundamentan en la mejora continua. La rapidez y la flexibilidad. La comunicación entre los integrantes pasa a ser el centro de atención, y se establecen hitos de entregas frecuentes de tareas pequeñas y concretas [4].

Las principales características de las metodologías Ágiles se pueden enumerar como:

- Mejora la calidad del Producto: La interacción entre el cliente y los desarrolladores tiene como objetivo crear un proyecto que cumpla las necesidades justas del cliente.
- Mejora la motivación e implicación del equipo de trabajo: Es importante escuchar las opiniones tanto del cliente como de los desarrolladores incluidos en el equipo, toda opinión es útil para la realización del proyecto.
- Retroalimentación más rápida: En este tipo de metodologías es común trabajar activamente con el cliente, escuchando sus opiniones y mostrándole avances constantes del proyecto.
- Ahorrar tiempo en costes: En estas metodologías se toma en cuenta mucho el hecho de mantenerse dentro del presupuesto y dentro de los tiempos de entrega.
- Se trabaja con mayor velocidad y eficiencia: Cada cierto periodo de tiempo corto se entrega una muestra de los avances del proyecto en versiones funcionales, lo que permite corregir errores e implementar mejoras de acuerdo con comentarios del equipo o cliente, además de mejorar así la calidad y eficiencia de trabajo.
- Eliminación de características innecesarias del producto: Al escuchar constantemente las opiniones del cliente se pueden eliminar características o necesidades que realmente no son necesarias o prioritarias en el desarrollo del proyecto.
- Alertar rápidamente tanto de errores como de problemas: Se detectan fácilmente situaciones como errores o bugs, excesos de presupuesto o tiempos de desarrollo [2].

En la siguiente imagen se muestra de forma gráfica las características del modelo *agile:*

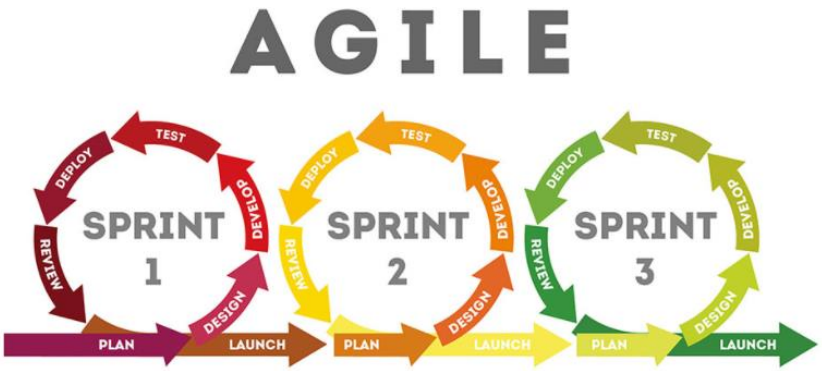


Ilustración 3. Características del modelo Agile

En la actualidad existen diferentes modos de metodologías de desarrollo ágil, entre las principales se puede enumerar:

- Metodología Scrum

Se puede considerar como la metodología ágil más ampliamente implantada [5]. Se trata de un método indicado para solventar problemas complejos y se basa en procesos empíricos de control. Esto significa que las decisiones se toman en función de la información existente y de la propia experiencia. Eso sí, cuenta con dos tipos de enfoque:

- Iterativo: en cada sprint se genera una nueva versión del producto que mejora la versión del sprint anterior. Se trata de ir refinando y mejorando las propiedades del producto conforme avanza el proyecto.
- Incremental: en cada periodo de tiempo corto se van añadiendo nuevas características al producto.

Se trata de un modelo ágil y flexible que tiene por objetivo controlar y planificar proyectos con un gran volumen de cambios de última hora, en donde la incertidumbre sea elevada.

Se centra en ajustar sus resultados y responder a las exigencias reales y exactas del cliente. De ahí, que se vaya revisando cada entregable, ya que los requerimientos van variando a corto plazo.

En cuanto a los distintos elementos que conforman la metodología se puede distinguir los tiempos asignados, la definición de hecho, el ciclo de Scrum, los productos y los distintos tipos de reuniones. Todo ello en base a sus pilares básicos y valores fundamentales. Por otra parte, en un proyecto ágil bajo metodología Scrum se distinguen tres roles: Product Owner, Scrum Máster y el equipo de desarrollo. Para llevarla a cabo hay que tener en cuenta: el punto de partida es una lista que debe priorizarse; Los bloques de tiempos o hitos hasta dura cada entregable, se le denomina “sprints”, y suelen ser de entre dos y cuatros semanas; y al finalizar cada sprint, se presenta los resultados para ser aprobados o rechazados. En la siguiente imagen se muestran todos integrantes, así como las diferentes ceremonias y los diferentes componentes en este modelo:

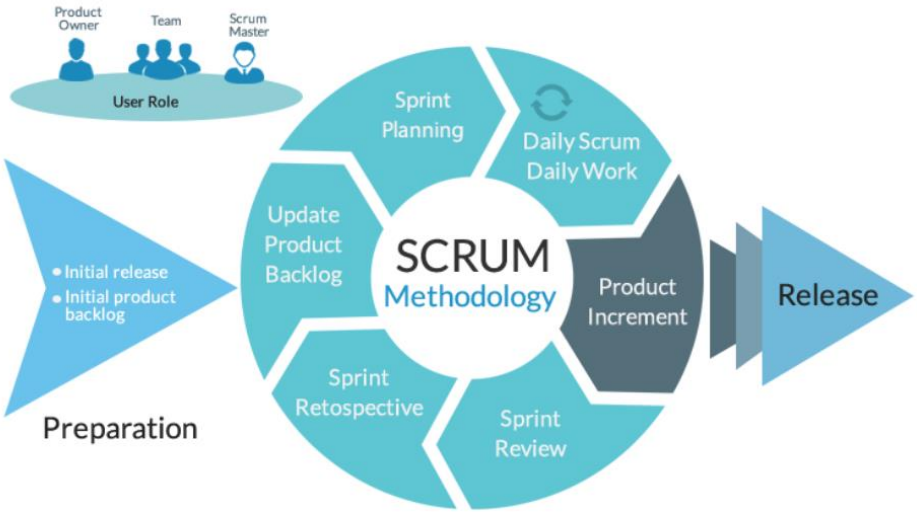


Ilustración 4. Metodología Scrum

- Metodología XP

Es una metodología ágil exclusiva para el desarrollo de software. Sus siglas provienen de Extreme Programming y, al igual que Scrum, contempla cambios frecuentes e iteraciones relativas a cortos periodos de tiempo.

En este caso se distinguen cuatro roles: líder ágil o coach, cliente, programador y tester. Con respecto a sus valores, XP recoge la simplicidad, la comunicación, el feedback, la motivación y el respeto como sus principales premisas. Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Las características fundamentales del método son:

- Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.

- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación. Por ejemplo, las herramientas de prueba JUnit orientada a Java, DUnit orientada a Delphi, NUnit para la plataforma.NET o PHPUnit para PHP. Estas tres últimas inspiradas en JUnit, la cual, a su vez, se inspiró en SUnit, el primer framework orientado a realizar tests, realizado para el lenguaje de programación Smalltalk.

- Programación en parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. La mayor calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata.

- Frecuente integración del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.

- Corrección de todos los errores antes de añadir nueva funcionalidad. Hacer entregas frecuentes.

- Refactorización del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad, pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.

- Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.

- Simplicidad en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo [6].

A continuación, se ejemplifica de forma gráfica las diferentes etapas en la programación XP:

Ilustración 5. Etapas de la programación XP

- Kanban

Esta metodología consiste en la organización del trabajo diario en base a un panel de tareas. No propone cambios en las prácticas de ingeniería ni una nueva definición de proceso o estilo de trabajo. En cambio, se diseña para evitar la sobreproducción y para asegurarse de que los componentes pasan de un subproceso al siguiente en el orden adecuado.

Así se desarrolla un sistema de relleno que controla las cantidades producidas para reponer los componentes solo cuando sea necesario. Eso sí, en lugar de utilizar Kanban específicos, también se pueden poner en marcha otros sistemas reutilizables, como contenedores, palets o bandas codificadas.

La metodología Kanban se basa en una serie de principios que la diferencian del resto de metodologías conocidas como ágiles:

Calidad garantizada. Todo lo que se hace debe salir bien a la primera, no hay margen de error. De aquí a que en Kanban no se premie la rapidez, sino la calidad final de las tareas realizadas. Esto se basa en el hecho que muchas veces cuesta más arreglarlo después que hacerlo bien a la primera.

Reducción del desperdicio. Kanban se basa en hacer solamente lo justo y necesario, pero hacerlo bien. Esto supone la reducción de todo aquello que es superficial o secundario (principio YAGNI).

Mejora continua. Kanban no es simplemente un método de gestión, sino también un sistema de mejora en el desarrollo de proyectos, según los objetivos a alcanzar.

Flexibilidad. Lo siguiente a realizar se decide del backlog (o tareas pendientes acumuladas), pudiéndose priorizar aquellas tareas entrantes según las necesidades del momento (capacidad de dar respuesta a tareas imprevistas).

- Lean

Es considerada tanto una metodología de trabajo como una filosofía centrada en maximizar el valor del cliente y minimizar el desperdicio. Esto traducido a los procesos de fabricación “pull” radica en producir solo lo necesario y en el momento adecuado. Tiene su origen en la empresaToyota, aplicada con mucho éxito en Japón y ahora muy famosa en el mundo del software, muchas veces bajo el término de Lean Software Development.

En este contexto, podemos decir que el Lean Start Up consiste en extender la metodología al lanzamiento de nuevas empresas al mercado. Los principios son los mismos que los de la filosofía Lean. Sin embargo, resalta todas las actividades que aportan valor al negocio al mismo tiempo que elimina las que no lo hagan.

Esta metodología incluye siete principios:

- Eliminar desperdicios o restos. Se refiera a código generado que ofrece funcionalidades no deseadas o necesarias, retrasos en el proceso de desarrollo de software, mala toma de requisitos, problemas con la comunicación interna, documentación excesiva o poco detallada.

- Amplificar el aprendizaje. Se refiere a la importancia que todos los miembros del equipo de desarrollo trabajen con una mentalidad de aprendizaje continuo. El hecho de que un desarrollador trabaje con una tecnología o lenguaje concreto no quiere decir que no pueda aprender de otros compañeros o proyectos.

- Tomar decisiones lo más tarde posible. Este principio que a priori puede parecer malo desde un punto de vista tradicional (ciclo de vida en cascada) en la filosofía Lean es primordial. Los requisitos de los clientes pueden cambiar de un día para otro, bien por cambios en las necesidades o bien por una mala definición de estos. Los requisitos suelen ser sustituidos por user stories que están más cerca de la necesidad real. Por este motivo se puede esperar a construir el software hasta que la user story esté definida claramente y sin ambigüedades.

- Entregar lo antes posible. Las entregas de software son más frecuentes incluyendo features alineadas con las user stories. Por este motivo, cada entrega incluirá funcionalidades que necesitan los usuarios lo antes posible basados en prioridades, impacto, valor o cualquier otro motivo.

- Potenciar el equipo. Facilitar que los desarrolladores participen en la toma de decisiones de tiempos asociados a tareas, priorización de estas y demás hacen que los miembros del equipo se sientan parte importante en él. Los desarrolladores saben de primera mano qué tareas cuestan más, cuales menos y qué implicaciones tienen el ciclo de vida del proyecto.

- Crear la integridad. Contar con un buen sistema de integración continua que incluya pruebas automatizadas, builds, pruebas de usabilidad son críticas para que un software sea fácil de mantener, de mejorar y de reutilizar. Con esto se evita añadir desperdicios a dicho software e intentar aprovechar lo aprendido de proyectos anteriores.

- Visualizar todo el conjunto. Analizar las interacciones del software con el resto de los sistemas dentro de la compañía permitirá estudiar posibles mejoras y cambios que redunden en una mejor experiencia de usuario y aporten un mayor valor para el cliente y para el equipo del proyecto.

FDD (Feature Drive Development)

Metodología FDD (Feature Driven Development). Es una metodología ágil para el desarrollo de sistemas, basado en la calidad del software, que incluye un monitoreo constante del proyecto. Esta metodología se enfoca en iteraciones cortas que permite entregas tangibles del producto en corto periodo de tiempo que como máximo son de dos semanas. La metodología ágil FDD contempla la figura del jefe de proyecto y una fase de arquitectura, está orientada a equipos más grandes, con más personas que aquellos a los que normalmente se aplican otras metodologías ágiles como Scrum.

FDD es una metodología dirigida por modelos, y de iteraciones cortas. FDD define 5 procesos: Proceso 1 – Desarrollar el modelo global (Develop overall model), Proceso 2 – Construir una lista de características (Build feature list), Proceso 3 – Planificar (Plan by feature), Proceso 4 – Diseñar (Design by feature) y Proceso 5 – Construir (Build by feature). Los 3 primeros pueden considerarse la “iteración cero”, aunque en FDD no le llaman así, y los consideran “procesos iníciales”.

No hace énfasis en la obtención de los requerimientos sino en cómo se realizan las fases de diseño y construcción. Se preocupa por la calidad, por lo que incluye un monitoreo constante del proyecto. Ayuda a contrarrestar situaciones como el exceso en el presupuesto, fallas en el programa o el hecho de entregar menos de lo deseado. Propone tener etapas de cierre cada dos semanas. Se obtienen resultados periódicos y tangibles. Se basa en un proceso iterativo con iteraciones cortas que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitorear. Define claramente entregas tangibles y formas de evaluación del progreso del proyecto [7].

El equipo de trabajo está estructurado en jerarquías, siempre debe haber un jefe de proyecto, y aunque es un proceso considerado ligero también incluye documentación (la mínima necesaria para que algún nuevo integrante pueda entender el desarrollo de inmediato); Un director del proyecto que es el líder administrativo y financiero del proyecto; El arquitecto jefe, que realiza el diseño global del sistema y ejecución de todas las etapas; El director de desarrollo que lleva las actividades de desarrollo; Programador jefe que analiza los requerimientos y diseña el proyecto. Selecciona las funcionalidades a desarrollar de la última fase del FDD; El propietario de clases que es responsable del desarrollo de las clases que se le asignaron como propias, participa en la decisión de que clase será incluida en la lista de funcionalidades de la próxima iteración; Expertos de dominio, que puede ser un usuario, un cliente, analista o una mezcla de estos. Poseen el conocimiento de los requerimientos del sistema.

TDD (Test Driven Development)

Es un proceso de desarrollo que consiste en codificar pruebas, desarrollar y refactorizar de forma continua el código construido.

La idea principal de esta metodología es realizar de forma inicial las pruebas unitarias para el código que se tiene que implementar. Es decir, primero se codifica la prueba y, posteriormente, se desarrolla la lógica de negocio.

Para conseguirlo, se escriben líneas de código y se prueba su comportamiento en el conjunto del programa, incluso cuando se sabe que la prueba a realizar es incompleta o dará algún tipo de fallo. Los errores que arroje el propio programa darán las pautas de las siguientes líneas de código que se debe implementar, y así sucesivamente hasta obtener un resultado final. Se desarrolla una parte del programa, se prueba el código, si este es funcional se pasa a ver si se puede optimizar, si no, se pasa a la corrección de errores.

El proceso TDD se puede simplificar a los siguientes pasos:

- Se escribe una prueba que recoja los requisitos.
- Se ejecuta la prueba. Esta debe fallar, en caso contrario es que no se está desarrollando bien, por lo tanto, no es válida.
- Se escribe la mínima cantidad de código necesaria para que el test pase.
- Se vuelve a ejecutar la prueba, esta debe correr exitosamente.
- Se recomienda refactorizar el código escrito, ya que cualquier cambio que se haga, se va a estar seguros de que el código va a funcionar si los tests son favorables.
- Se repete el punto uno para el siguiente requisito.

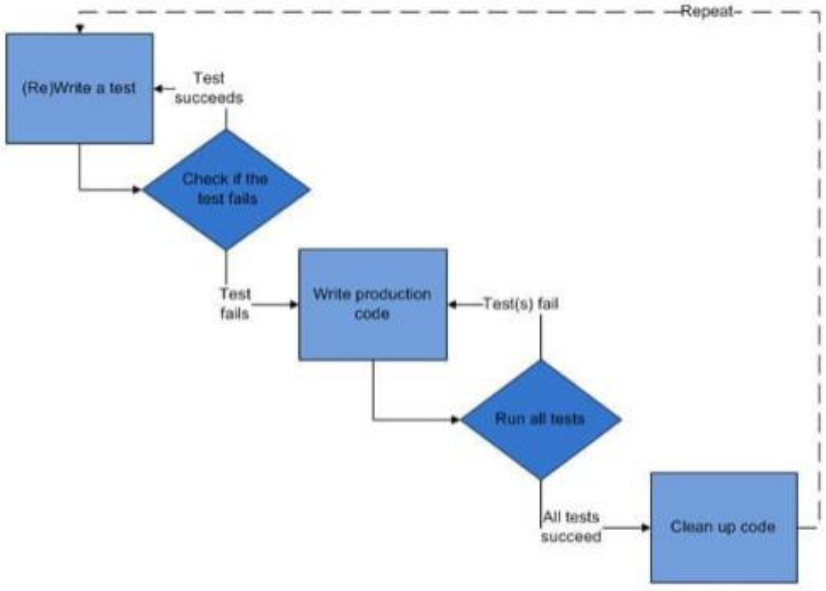


Ilustración 6. Proceso TDD

INTEGRACIÓN Y DESPLIEGUE CONTINUO

3.1 Integración Continua

Todas las metodologías ágiles mencionadas en los apartados anteriores tienen un denominador común: la integración y despliegue continuo.

La integración continua (*continuous integration*) es una práctica de ingeniería de software que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. Se entiendo por integración, la compilación y ejecución de pruebas de todo un proyecto. Esta práctica de desarrollo forma parte del proceso de desarrollo XP.

El proceso suele ser: cada cierto tiempo (horas), descargarse las fuentes desde el control de versiones (por ejemplo CVS, Git, Subversion, Mercurial o Microsoft Visual SourceSafe) compilarlo, ejecutar pruebas y generar informes.

Cada integración se verifica mediante una compilación automatizada (incluida la prueba) para detectar errores de integración lo más rápido posible. Muchos equipos encuentran que este enfoque conduce a problemas de integración significativamente reducidos y permite que un equipo desarrolle software cohesivo más rápidamente [9].

La Integración Continua se centra en aportar una realimentación casi de forma instantánea del estado, o evolución, de cada entrega de código al repositorio, mediante la automatización del proceso de construcción del software, dando la posibilidad de detectar errores de integración en las primeras etapas de la construcción del software, aumentando notablemente la calidad el software entregado.

En la siguiente imagen se muestra de una forma resumida las fases más importantes llevadas a cabo durante la integración continua:

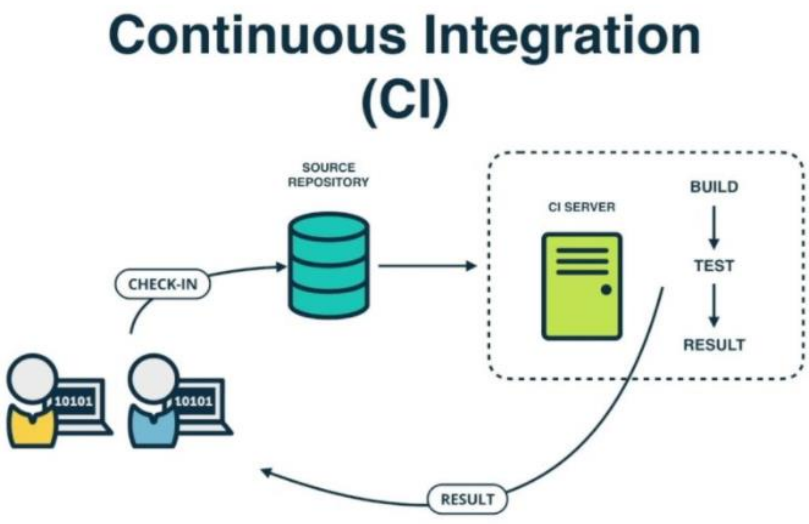


Ilustración 7. Integración continua

En el proceso de desarrollo en la que al menos existen dos desarrolladores, al embarcarse en un cambio, un desarrollador toma una copia del código base actual sobre el que trabajar. A medida que otros desarrolladores envían el código modificado al repositorio de código fuente, esta copia deja de reflejar gradualmente el código del repositorio. No solo se puede cambiar la base del código existente, sino que se puede agregar nuevo código, así como nuevas bibliotecas y otros recursos que crean dependencias y posibles conflictos. A medida que crece el código, el desarrollo en una rama sin que esta vuelva a fusionarse con la rama principal será mayor y por tanto incrementará el riesgo de múltiples conflictos de integración y fallas cuando la rama de un desarrollador finalmente se vuelva a fusionar con la rama principal. Cuando los desarrolladores envían código al repositorio, primero deben actualizar su código para reflejar los cambios en el repositorio desde que tomaron su copia.

Cuantos más cambios contenga el repositorio, más trabajo deben hacer los desarrolladores antes de enviar sus propios cambios. Con el tiempo, el repositorio puede volverse tan diferente de las líneas de base de los desarrolladores que ingresan, a lo que a veces se conoce como "infierno de fusión" o "infierno de integración", donde el tiempo que se tarda en integrar supera el tiempo que se tarda en hacer sus cambios originales.

Flujo de trabajo

En el flujo de trabajo de integración continua se pueden distinguir las siguientes fases:

- Ejecutar pruebas de forma local. Está destinado a ser utilizado en combinación con pruebas unitarias automatizadas escritas a través de las prácticas de desarrollo impulsado por pruebas. Esto se hace ejecutando y pasando todas las pruebas unitarias en el entorno local del desarrollador antes de mezclarse con la rama principal. Esto ayuda a evitar que el trabajo en curso de un desarrollador rompa la copia de otro desarrollador.

- Compilar código en CI. Un servidor de compilación compila el código periódicamente o incluso después de cada confirmación e informa los resultados a los desarrolladores. El uso de servidores de compilación se introdujo fuera de la comunidad de XP (programación extrema) y muchas organizaciones han adoptado CI sin adoptar toda la metodología XP.

- Ejecutar pruebas en CI. Además de las pruebas unitarias automatizadas, las organizaciones que utilizan CI deben utilizar un servidor de compilación para implementar procesos continuos de aplicación del control de calidad, en general. Además de ejecutar las pruebas unitarias y de integración, dichos procesos ejecutan análisis estáticos adicionales, miden y perfilan el rendimiento, extraen y dan formato a la documentación del código fuente y facilitan los procesos manuales de control de calidad. Un ejemplo en este sentido es el servicio Travis CI, que tiene como objetivo mejorar la calidad del software y reducir el tiempo necesario para entregarlo, reemplazando la práctica tradicional de aplicar el control de calidad después de completar todo el desarrollo. Esto es muy similar a la idea original de integrar con más frecuencia para facilitar la integración, solo se aplica a los procesos de control de calidad.

- Implementar un artefacto de CI. La integración continua a menudo se entrelaza con la entrega o la implementación continuas en lo que se denomina canalización de CI / CD. La "entrega continua" asegura que el software registrado en la rama principal esté siempre en un estado que se pueda implementar para los usuarios y el "despliegue" hace que el proceso de implementación sea completamente automatizado.

Buenas prácticas

Se enumeran las mejores prácticas sugeridas por varios autores sobre cómo lograr una integración continua y cómo automatizar esta práctica. La automatización de la compilación es una práctica recomendada en sí misma.

La integración continua debe ocurrir con la frecuencia suficiente para que no quede ninguna ventana intermedia entre la confirmación y la compilación, y de manera que no puedan surgir errores sin que los desarrolladores los noten y los corrijan de inmediato. La práctica normal es activar estas compilaciones por cada confirmación en un repositorio, en lugar de una compilación programada periódicamente. De esta manera, los eventos de confirmación se "eliminan" para evitar compilaciones innecesarias entre una serie de confirmaciones rápidas. Muchas herramientas automatizadas ofrecen esta programación automáticamente.

Otro factor es la necesidad de un sistema de control de versiones que admita confirmaciones atómicas; es decir, todos los cambios de un desarrollador pueden verse como una única operación de confirmación.

Para lograr estos objetivos, la integración continua se basa en los siguientes principios:

- Mantener un único repositorio de código. Esta práctica aboga por el uso de un sistema de control de revisiones para el código fuente del proyecto. Todos los artefactos necesarios para construir el proyecto deben colocarse en el repositorio. En esta práctica y en la comunidad de control de revisiones, la convención es que el sistema debe poder compilarse a partir de una caja nueva y no requerir dependencias adicionales. El defensor de la programación extrema, Martin Fowler, también menciona que cuando la ramificación es compatible con herramientas, su uso debe minimizarse. En cambio, se prefiere que los cambios se integren en lugar de que se mantengan simultáneamente varias versiones del software. La rama principal debe ser el lugar para la versión de trabajo del software.

- Automatiza la compilación. Un solo comando debe tener la capacidad de construir el sistema. Muchas herramientas de construcción, como *make*, existen desde hace muchos años. Otras herramientas más recientes se utilizan con frecuencia en entornos de integración continua. La automatización de la compilación debe incluir la automatización de la integración, que a menudo incluye la implementación en un entorno similar al de producción. En muchos casos, el script de compilación no solo compila binarios, sino que también genera documentación, páginas de sitios web, estadísticas y medios de distribución.

- Realice la autocomprobación de la compilación. Una vez que el código está construido, todas las pruebas deben ejecutarse para confirmar que se comporta como los desarrolladores esperan que se comporte.

- Todos realizan *commits* a la rama principal todos los días. Al realiza *commits* (confirmación) con regularidad, se puede reducir el número de cambios conflictivos. Registrar el trabajo de una semana corre el riesgo de entrar en conflicto con otras funciones y puede ser muy difícil de resolver. Los pequeños conflictos tempranos en un área del sistema hacen que los miembros del equipo se comuniquen sobre el cambio que están realizando. Confirmar todos los cambios al menos una vez al día, generalmente se considera parte de la definición de Integración Continua. Además, generalmente se recomienda realizar una compilación nocturna.

- Cada *commit* (a la rama principal) debería ser compilada. El sistema debe crear confirmaciones con la versión de trabajo actual para verificar que se integren correctamente. Una práctica común es utilizar la integración continúa automatizada, aunque esto se puede hacer manualmente. La Integración Continúa Automatizada emplea un servidor para monitorear el sistema de control de revisiones en busca de cambios y luego ejecutar automáticamente el proceso de compilación.

- Cada *commit* de corrección de errores debe venir con un caso de prueba. Al corregir un error, es una buena práctica impulsar un caso de prueba que reproduzca el error. Esto evita que se revierta la corrección y que vuelva a aparecer el error, lo que se conoce como regresión. Los investigadores han propuesto automatizar esta tarea: si una confirmación de corrección de errores no contiene un caso de prueba, se puede generar a partir de las pruebas ya existentes.

- Mantener la compilación en el menor tiempo posible. La compilación debe completarse rápidamente, de modo que, si hay un problema con la integración, se identifique rápidamente.

- Realizar pruebas en un clon del entorno de producción. Tener un entorno de prueba puede provocar fallas en los sistemas probados cuando se implementan en el entorno de producción porque el entorno de producción puede diferir del entorno de prueba de manera significativa. Sin embargo, construir una réplica de un entorno de producción tiene un costo prohibitivo. En su lugar, el entorno de prueba o un entorno de preproducción debe construirse para que sea una versión escalable del entorno de producción para aliviar los costos y, al mismo tiempo, mantener la composición y los matices de la pila de tecnología. Dentro de estos entornos de prueba, la virtualización de servicios se usa comúnmente para obtener acceso bajo demanda de ciertas dependencias que son demasiado complejas para configurar en un laboratorio de pruebas virtual [8][9].

3.2 Despliegue continuo

El despliegue continuo (también denominado implementación) consiste en empaquetar e implementar lo construido y testeado en la integración continua, permitiendo una entrega de software automatizada (sin intervención humana) con un menor coste, mayor rapidez y gran fiabilidad [10].

El despliegue continuo es una estrategia de desarrollo de software en la que los cambios de código de una aplicación se publican automáticamente en el entorno de producción. Esta automatización se basa en una serie de pruebas predefinidas. Una vez que las nuevas actualizaciones pasan esas pruebas, el sistema envía las actualizaciones directamente a los usuarios del software.

El despliegue continuo ofrece varias ventajas para las empresas que quieren escalar su porfolio de TI y aplicaciones. En primer lugar, agiliza el tiempo de comercialización al eliminar el desfase entre la codificación y el valor del cliente, que suele ser de días, semanas o incluso meses.

Para lograrlo, se deben automatizar las pruebas de regresión, de modo que se eliminen los altos costes de las pruebas de regresión manual. Los sistemas que las organizaciones han puesto en marcha para gestionar grandes paquetes de cambios de producción, incluidas las reuniones de planificación y aprobación de publicaciones, también se pueden eliminar en la mayoría de los cambios.

El despliegue continuo amplía un poco más la automatización y elimina la necesidad de intervención manual. Las pruebas y los desarrolladores se consideran lo suficientemente fiables como para que no sea necesario aprobar la publicación en producción. Si se pasan las pruebas, se considera que el código nuevo está aprobado y se inicia el despliegue en producción.

El despliegue continuo es el resultado natural de una entrega continua bien realizada. Al final, la aprobación manual aporta un valor mínimo o nulo, y solo ralentiza el proceso. Llegados a ese punto, se suprime, y la entrega continua pasa a ser un despliegue continuo.

Para que funcione la automatización de los procesos de despliegue, todos los desarrolladores que trabajan en un proyecto necesitan una forma eficaz de comunicar los cambios que se producen. Y eso es lo que posibilita la integración continua.

Para desarrollar y desplegar continuamente mejoras de software de alta calidad, los desarrolladores deben utilizar las herramientas adecuadas para crear prácticas eficaces de DevOps. Hacerlo no solo garantiza una comunicación eficaz entre los departamentos de desarrollo y operaciones, sino que también minimiza o elimina los errores en el conducto de entrega de software.

Estas son algunas de las herramientas más cruciales utilizadas en un flujo de trabajo de despliegue continuo:

- Control de versiones: el control de versiones facilita la integración continua mediante el seguimiento de las revisiones de los activos de un proyecto determinado. También conocido como control de "origen" o "revisión", el control de versiones ayuda a mejorar la visibilidad de las actualizaciones y los cambios de un proyecto, además de facilitar la colaboración de los equipos independientemente de dónde y cuándo trabajen.

- Revisión de código: tan sencilla como parece, la "revisión de código" es el proceso de utilizar herramientas para probar el código fuente actual. Las revisiones de código permiten mejorar la integridad del software al encontrar errores en la codificación y ayudan a los desarrolladores a resolver estos problemas antes de desplegar las actualizaciones.

- Integración continua (CI): la CI es un componente indispensable del despliegue continuo y desempeña un papel muy importante en la minimización de los obstáculos para el desarrollo cuando varios desarrolladores trabajan en el mismo proyecto. Existe una gran variedad de herramientas de CI propietarias y de código abierto, y cada una de ellas atiende las complejidades exclusivas de los despliegues de software empresarial.

- Gestión de configuración: la gestión de la configuración es la estrategia y la disciplina de asegurarse de que todo el software y el hardware mantienen un estado coherente. Esto incluye la configuración y la automatización adecuadas de todos los servidores, el almacenamiento, la red y el software.

- Automatización de publicaciones: la automatización de las publicaciones de aplicaciones (o la orquestación de publicaciones de aplicaciones) es muy importante al automatizar todas las actividades necesarias para impulsar el despliegue continuo. Las herramientas de orquestación conectan los procesos entre sí para garantizar que los desarrolladores sigan todos los pasos necesarios antes de enviar nuevos cambios a producción. Estas herramientas trabajan estrechamente con los procesos de gestión de configuración para garantizar que todos los entornos del proyecto se suministren correctamente y puedan ofrecer su máximo nivel de rendimiento.

- Supervisión de infraestructuras: al utilizar un modelo de despliegue continuo, es importante poder visualizar los datos que residen en los entornos de prueba. Las herramientas de supervisión de la infraestructura le ayudan a analizar el rendimiento de las aplicaciones para ver si los cambios realizados tienen un impacto positivo o negativo [11].

La implementación continua es el siguiente paso más allá de la entrega continua, donde no solo se está creando de manera constante un paquete implementable, sino que en realidad lo está implementando de manera constante.

Implementar constantemente el código en producción a medida que se completan las funciones y tan pronto como se cumplen los criterios de publicación para esas funciones. Ese criterio de lanzamiento depende de su situación, y puede estar ejecutando algunas pruebas automatizadas, revisiones de código, pruebas de carga, verificación manual por parte de una persona de control de calidad o una parte interesada comercial. Una vez más, los criterios específicos pueden variar, pero la idea clave es tener un flujo constante que impulse los cambios en la producción, que siempre haga avanzar el código y mantenga el flujo lo más corto posible de manera realista [13].

En la siguiente imagen se muestra de forma gráfica, el proceso de despliegue continuo:

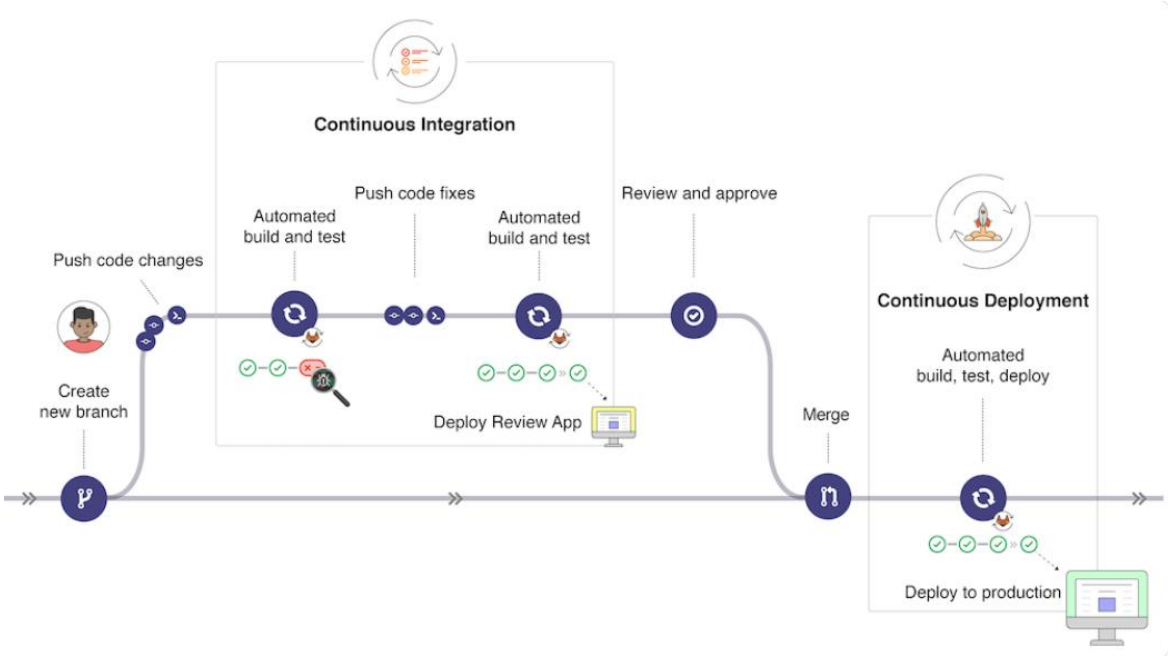


Ilustración 8. Proceso de integración y despliegue continuo

3.3 Entrega continua

Un paso intermedio entre la integración continua y el despliegue a producción, se encuentra la Entrega continúa (o distribución), que es una práctica de desarrollo de software en la que el software se crea de forma que se puede publicar en producción en cualquier momento dado. Para ello, los modelos de entrega continua utilizan entornos de prueba similares a la producción. Las nuevas compilaciones realizadas en una solución de entrega continua se despliegan automáticamente en un entorno de pruebas automáticas de control de calidad que buscan errores e incoherencias. Una vez que el código pasa todas las pruebas, en la entrega continua se requiere intervención humana para aprobar los despliegues en producción. A continuación, el despliegue en sí se realiza mediante automatización.

Después de la automatización de los diseños y las pruebas de unidad e integración de la CI, la distribución continua automatiza la liberación de ese código validado hacia un repositorio. Por eso, para que el proceso de distribución continua sea eficaz, es importante que la CI ya esté incorporada a su canal de desarrollo. El objetivo de la distribución continua es tener una base de código que pueda implementarse en un entorno de producción en cualquier momento.

En la distribución continua, cada etapa (desde la fusión de los cambios en el código hasta la distribución de los diseños listos para la producción) implica la automatización de las pruebas y de la liberación de código. Al final de este proceso, el equipo de operaciones puede implementar una aplicación para que llegue a la etapa de producción de forma rápida y sencilla [12].

La entrega continua es otra práctica de DevOps que se enfoca en entregar cualquier cambio en el código (actualizaciones, correcciones de errores e incluso nuevas funciones) a los usuarios de la manera más rápida y segura posible. La idea es poder lanzar cambios bajo demanda en cualquier punto del ciclo de vida del desarrollo a través de una combinación de automatización y cambios culturales. En la siguiente imagen se muestra las fases que integran la entrega continua:

Ilustración 9. Fases de la Entrega Continua

3.4 Beneficios

De entre los múltiples beneficios de la integración y despliegue continuo se pueden citar:

- Mejora la productividad de desarrollo. La integración continua mejora la productividad del equipo al liberar a los desarrolladores de las tareas manuales y fomentar comportamientos que ayudan a reducir la cantidad de errores y bugs enviados a los clientes.

- Garantiza una calidad constante y un código liberable en todo momento. Gracias a la realización de pruebas más frecuentes, el equipo puede descubrir y arreglar los errores antes de que se conviertan en problemas más graves.

- Se entregan las actualizaciones con mayor rapidez. La integración continua le permite a su equipo entregar actualizaciones a los clientes con mayor rapidez y frecuencia [13]. Pueden obtener las funciones que solicitan en un período mínimo de tiempo e incluso solicitar cambios en la funcionalidad.

- Con el despliegue continuo, se automatiza todo el proceso de entrega de software al usuario, eliminando la acción manual o intervención humana necesaria en la entrega continua.

- Ejecución inmediata de las pruebas unitarias.

- Monitorización continúa de las métricas de calidad del proyecto.

- Y como resultado de los puntos anteriores, se incluye como beneficio: tiempo de comercialización más rápido, menores costos, productos mejorados y una mejor cooperación entre equipos.

3.5 Principales herramientas para implementar la Integración Continua

Para desarrollar y desplegar continuamente mejoras de software de alta calidad, los desarrolladores deben utilizar las herramientas adecuadas para crear prácticas eficaces de DevOps. Hacerlo no solo garantiza una comunicación eficaz entre los departamentos de desarrollo y operaciones, sino que también minimiza o elimina los errores en el conducto de entrega de software.

Estas son algunas de las herramientas más cruciales utilizadas en un flujo de trabajo de despliegue continuo [11]:

Sistema de control de versiones. Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación. Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión dando lugar a los llamados sistemas de control de versiones o VCS. Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas. Ejemplos de este tipo de herramientas son entre otros: CVS, Subversion, SourceSafe, ClearCase, Darcs, Bazaar, Plastic
SCM, Git, SCCS, Mercurial, Perforce, Fossil SCM, Team Foundation Server. Un sistema de control de versiones debe proporcionar mecanismos de almacenamiento de los elementos que deba gestionar (ej. archivos de texto, imágenes, documentación...), posibilidad de realizar cambios sobre los elementos almacenados (ej. modificaciones parciales, añadir, borrar, renombrar o mover elementos) y un registro de históricos de las acciones realizadas con cada elemento o conjunto de elementos (normalmente pudiendo volver o extraer un estado anterior del producto).

- **GIT**: es un software de control de versiones que se centra en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código. Entre las características más relevantes se encuentran [15]:

- Fuerte apoyo al desarrollo no lineal, por ende, rapidez en la gestión de ramas y mezclado de diferentes versiones.
- Gestión distribuida. Git le da a cada programador una copia local del historial del desarrollo entero, y los cambios se propagan entre los repositorios locales. Los cambios se importan como ramas adicionales y pueden ser fusionados en la misma manera que se hace con la rama local.
- Los almacenes de información pueden publicarse por HTTP, FTP, rsync o mediante un protocolo nativo, ya sea a través de una conexión TCP/IP simple o a través de cifrado SSH. Git también puede emular servidores CVS.
- Gestión eficiente de proyectos grandes, dada la rapidez de gestión de diferencias entre archivos, entre otras mejoras de optimización de velocidad de ejecución.

- Todas las versiones previas a un cambio determinado implican la notificación de un cambio posterior en cualquiera de ellas a ese cambio.

- **GitLab**: es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado bajo una Licencia de código abierto. GitLab es una suite completa que permite gestionar, administrar, crear y conectar los repositorios con diferentes aplicaciones y hacer todo tipo de integraciones con ellas, ofreciendo un ambiente y una plataforma en cual se puede realizar las varias etapas de su SDLC/ADLC y DevOps [16]. Gitlab es una nueva herramienta de CI que ofrece una experiencia completa de DevOps. Gitlab se creó con la intención de mejorar la experiencia general de Github, ofreciendo una interfaz de usuario actual y es compatible con contenedores, alojamiento local o en la nube, pruebas de seguridad continuas y una interfaz de usuario sencilla.

- **GitHub**: es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de ordenador [17]. Es un portal para gestionar las aplicaciones que utilizan el sistema Git. Además de permitir revisar el código y descargar las diferentes versiones de una aplicación, la plataforma también hace las veces de red social conectando desarrolladores con usuarios para que estos puedan colaborar mejorando la aplicación. Ofrece las mejores características de este tipo de servicios sin perder la simplicidad, y es una de las más utilizadas por los desarrolladores.

Además, existen varios clientes GUI para gestionar los repositorios git. Algunos de los más utilizados: SourceTree, Qgit, GitUp o GitKraken.

Herramientas de construcción de proyectos. Para hacer frente a la problemática de gestión de dependencias y automatización de la compilación existen varias herramientas que gestionan estos procesos. Inicialmente, *make* era la única herramienta disponible para automatizar la construcción de programas, además de las soluciones que se desarrollaron por cuenta propia. Sin embargo, esta solución no proporcionaba todos los requisitos en cuanto a optimización y aparecieron nuevas alternativas:

- **Apache Ant:** es una biblioteca de Java y una herramienta de línea de comandos cuya misión es impulsar los procesos descritos en los archivos de compilación como objetivos y puntos de extensión que dependen unos de otros. El principal uso conocido de Ant es la construcción de aplicaciones Java. Ant proporciona una serie de tareas integradas que permiten compilar, ensamblar, probar y ejecutar aplicaciones Java. Ant también se puede utilizar de forma eficaz para crear aplicaciones que no sean Java, por ejemplo, aplicaciones C o C ++. De manera más general, Ant se puede utilizar para pilotar cualquier tipo de proceso que pueda describirse en términos de objetivos y tareas [18].

- **Apache Maven**: es una herramienta de gestión de proyectos de desarrollo utilizada principalmente en el entorno de computación Java utilizando conceptos provenientes de Apache Ant. Es bastante personalizable, permitiendo finalizar tareas complejas de forma rápida y reutilizando los resultados de ejecuciones pasadas. La configuración de un proyecto se basa en un fichero XML en el cual se declaran los requerimientos para la construcción. Maven añade principalmente las siguientes funcionalidades a Apache Ant: gestión de repositorios, gestión de dependencias y gestión del ciclo de vida. El archivo de configuración de Maven, que contiene instrucciones de administración de dependencia y compilación, se llama por convención *pom.xml*. Además, Maven también prescribe una estructura de proyecto estricta [19].

- **Gradle:** es un sistema de automatización de construcción de código de software que construye sobre los conceptos de Apache Ant y Apache Maven e introduce un lenguaje específico del dominio (DSL) basado en *Groovy* en vez de la forma XML utilizada por Apache Maven para declarar la configuración de proyecto. *Gradle* utiliza un grafo acíclico dirigido ("DAG") para determinar el orden en el que las tareas pueden ser ejecutadas [18]. Las ventajas con respecto a Maven, es que Gradle usa el lenguaje DSL en vez de XML (por tanto, los scripts son más cortos y limpios) y mejora en cuanto a la gestión del ciclo de vida, ya que añade la capacidad de soportar todo el proceso de vida del software. El tiempo de construcción de Gradle es más corto y rápido, se basa en la tarea mediante la cual se realiza el trabajo, admite compilaciones incrementales de la clase java y tiene soporte en la mayoría de las herramientas de Integración continua [20].

Gestores de integración continua. En la actualidad, se ofrece una gran variedad de servidores para la integración continua. Todas tienen como objetivo ayudar al desarrollador en la implementación de la integración continua, y lo hacen de diferentes modos y con la ayuda de características distintas. Pero estas herramientas no solo se diferencian unas de otras en cuanto a sus características, sino que también existe una gran variedad en lo que respecta a precios y licencias. Mientras que muchas de ellas son de código abierto y se encuentran disponibles de forma gratuita, otros fabricantes ofrecen herramientas comerciales. A continuación, te ofrecemos un resumen de las más utilizadas y examinamos sus características y funciones:

- **Jenkins**: es probablemente una de las herramientas de integración continua más conocidas del mercado. Este software escrito en Java ha continuado desarrollándose constantemente desde el año 2005 (entonces, bajo el nombre de Hudson) y cuenta en la actualidad con numerosas funciones que asisten no solo en la integración continua, sino también en el despliegue y la entrega continua. En la próxima sección se describe más en profundidad.

- **Travis CI:** es un servicio de integración continúa alojado que se utiliza para crear y probar proyectos de software alojados en GitHub y Bitbucket. Esta herramienta puede configurarse con un sencillo archivo YAML que se guarda en el directorio raíz del proyecto. GitHub informa a Travis CI de todos los cambios efectuados en el repositorio y mantiene el proyecto actualizado.

  Se puede destacar que Travis CI es:

- Programado en Ruby
- Multiplataforma
- Funciona con GitHub
- Se configura con un archivo YAML
- Gratuita para proyectos de código abierto
- Precio para proyectos comerciales: entre 69 y 489 dólares/mes
- De código abierto (licencia MIT)

- **Bamboo:** es una herramienta de integración continua y despliegue que reúne compilaciones, pruebas y versiones automatizadas en un solo flujo de trabajo. Esta herramienta no solo sirve de ayuda en la integración continua, sino también para funciones de despliegue y gestión de lanzamientos. Funciona a través de una interfaz web. A destacar por esta herramienta:

- Escrito en Java
- Multiplataforma
- Fácil integración de otros productos Atlassian
- Gran cantidad de addons
- Realización de varias pruebas al mismo tiempo
- Interfaz web y API REST
- Gratuita para proyectos de código libre, ONG y centros escolares
- De lo contrario, pago único de entre 10 y 126 500 dólares, dependiendo del número de servidores utilizados

- **GitLab CI**:forma parte del conocido sistema de control de versiones GitLab. Además de integración continua, ofrece despliegue y entrega continua. Al igual que con Travis CI, la configuración de GitLab CI se lleva a cabo con un archivo YAML. A destacar por esta herramienta:

- Forma parte de GitLab
- Programado en Ruby y Go
- Configuración con un archivo YAML
- Asiste también en la entrega y el despliegue continuo
- Open Core
- Alojamiento propio o en la nube
- Versión gratuita con pocas funciones
- Precio para otras versiones, entre 4 y 99 dólares/mes por usuario

- **CircleCI:** la herramienta de integración continua CircleCI funciona tanto con GitHub como con Bitbucket. En las fases de prueba, pueden emplearse tanto contenedores como máquinas virtuales. CircleCI confiere mucha importancia a la ejecución de procesos de desarrollo sin interferencias, por lo que arroja de forma automática builds compatibles con otros entornos. Sus características son:

- Configuración con un archivo YAML
- Soporta también el despliegue continuo
- Alojamiento propio o en la nube
- Se ejecuta en contenedores Docker, máquinas virtuales Linux y MacOS
- Gratuita para un contenedor

- **Codeship:** es un servicio de entrega continúa alojado en la nube centrado en la fiabilidad, velocidad y simplicidad. Entre sus características:

- Interfaz web en la versión básica
- Archivos de configuración en el repositorio en la versión profesional
- Asistencia Docker en la versión profesional
- Gratuita para 100 compilaciones al mes en un pipeline de prueba
- Precio entre 75 y 1500 dólares/mes

- **TeamCity:** es una solución de integración e implementación continuas de uso general que admite gran flexibilidad para todo tipo de flujos de trabajo y prácticas de desarrollo. Destaca por su *gated commits* (evita que los desarrolladores rompan las fuentes en un sistema de control de versiones ejecutando la compilación de forma remota para los cambios locales antes de la confirmación). Se puede mencionar que es:

- Escrito en Java
- Multiplataforma
- Gated Commits
- Gratuito para 100 builds con 3 agentes de compilación
- Pago único de entre 299 euros y 21 999 euros
- Con 50 % de descuento para startups y gratuita para proyectos de código abierto

JENKINS

4.1 Descripción

Jenkins es un servidor automatizado de integración continua de código abierto capaz de organizar una cadena de acciones que ayudan a lograr el proceso de integración continua de manera automatizada. Es un servidor de automatización de código abierto autónomo que se puede utilizar para automatizar todo tipo de tareas relacionadas con la creación, prueba y entrega o implementación de software [22][23].

Jenkins puede instalarse a través de paquetes de sistema nativos, Docker o incluso ejecutarse de forma independiente en cualquier máquina que tenga instalado Java Runtime Environment (JRE).

Jenkins está completamente escrito en Java y es una aplicación conocida y reconocida por DevOps de todo el mundo, más de 300.000 instalaciones y creciendo día a día. La razón por la que Jenkins se hizo tan popular es porque se encarga de supervisar las tareas repetitivas que surgen dentro del desarrollo de un proyecto. Jenkins prueba continuamente las compilaciones de este y será capaz de mostrar los errores que aparezcan a lo largo de las primeras etapas del desarrollo.

Al usar Jenkins, las compañías de software pueden acelerar su proceso de desarrollo del código, ya que Jenkins puede automatizar, agilizar y aumentar el ritmo de toda la compilación y las pruebas de los proyectos. Además, Jenkins puede ser implementado a lo largo de todo el ciclo de vida completo del desarrollo, desde la fase de construcción inicial, la fase de pruebas, en la documentación del software, en su implementación y en todas las demás etapas existentes dentro del ciclo de vida del software.

Ilustración 10. Logotipo característico de Jenkins

Ventajas de usar Jenkins

- Jenkins está siendo administrada por la comunidad, que es muy abierta. Todos los meses, celebran reuniones públicas y reciben aportes del público para el desarrollo del proyecto Jenkins.
- Hasta ahora, alrededor de 280 entradas están cerradas, y el proyecto publica un lanzamiento estable cada tres meses.
- A medida que la tecnología crece, también lo hace Jenkins. Hasta ahora, Jenkins tiene alrededor de 320 complementos publicados en su base de datos de complementos. Con los complementos, Jenkins se vuelve aún más potente y rico en funciones.
- Jenkins también admite arquitectura basada en la nube para que pueda implementar Jenkins en plataformas basadas en la nube.
- La razón por la que Jenkins se hizo popular es que fue creado por un desarrollador para desarrolladores.

Desventajas de usar Jenkins

- Aunque Jenkins es una herramienta muy poderosa, tiene sus defectos.
- Su interfaz está desactualizada y no es fácil de usar en comparación con las tendencias actuales de la interfaz de usuario.
- Aunque Jenkins es amado por muchos desarrolladores, no es tan fácil mantenerlo porque Jenkins se ejecuta en un servidor y requiere algunas habilidades como administrador del servidor para monitorear su actividad.
- Una de las razones por las cuales muchas personas no implementan Jenkins se debe a su dificultad para instalar y configurar Jenkins.
- Las integraciones continuas se rompen regularmente debido a algunos pequeños cambios de configuración. La integración continua se detendrá y, por lo tanto, requiere cierta atención del desarrollador.

Compilaciones

Las compilaciones se pueden activar por varios medios, por ejemplo:

- un webhook que se activa al enviar confirmaciones en un sistema de control de versiones
- programación a través de un mecanismo similar a un cron
- solicitando una URL de compilación específica
- después de que las otras compilaciones en la cola se hayan completado
- invocado por otras compilaciones

Plugins

Jenkins fue creado, originariamente, para gestionar proyectos que utilizan java como lenguaje de programación. Los complementos (plugins) para Jenkins extienden su uso a proyectos escritos en lenguajes distintos de Java. Hay complementos disponibles para integrar Jenkins con la mayoría de los sistemas de control de versiones y bases de datos. Muchas herramientas de compilación son compatibles a través de sus respectivos complementos. Los complementos también pueden cambiar la apariencia de Jenkins o agregar nuevas funciones. Hay un conjunto de complementos dedicados a las pruebas unitarias que generan informes de prueba en varios formatos (por ejemplo, JUnit incluido con Jenkins, MSTest, NUnit, etc.) y admiten pruebas automatizadas. Las compilaciones pueden generar informes de prueba en varios formatos y Jenkins puede mostrar los informes y generar tendencias y representarlos en la GUI. Algunos de los complementos más utilizados son:

- Notificación vía. Permite configurar notificaciones por correo electrónico para los resultados de las compilaciones. Jenkins enviará correos electrónicos a los destinatarios especificados cada vez que ocurra un evento importante, como puede ser una compilación fallida o una compilación inestable.
- Gestión de credenciales. Permite almacenar credenciales en Jenkins. Proporciona una API estandarizada para que otros complementos almacenen y recuperen diferentes tipos de credenciales.
- Supervisión de trabajos externos. Agrega la capacidad de monitorear el resultado de trabajos ejecutados externamente.
- Agentes SSH. Este complemento permite administrar agentes que se ejecutan en máquinas a través de SSH.
- Javadoc. Este complemento agrega compatibilidad con Javadoc a Jenkins. El complemento permite la opción de "Publicar Javadoc" como una acción posterior a la compilación [].

Seguridad

Jenkins se utiliza en todas partes, desde estaciones de trabajo en intranets corporativas hasta servidores de alta potencia conectados a la Internet pública. Para respaldar de manera segura esta amplia variedad de perfiles de seguridad y amenazas, Jenkins ofrece muchas opciones de configuración para habilitar, personalizar o deshabilitar varias funciones de seguridad [23]. La seguridad de Jenkins depende de dos factores: control de acceso y protección contra amenazas externas. El control de acceso se puede personalizar de dos formas: autenticación de usuario y autorización. También se admite la protección contra amenazas externas, como ataques CSRF y compilaciones maliciosas.

4.2 Descarga e instalación[1]

Descarga

El proyecto Jenkins produce dos líneas de lanzamiento: estable (LTS) y regular (semanal). Dependiendo de las necesidades de la organización, se puede preferir una sobre la otra.

- Estable (LTS)
  Las líneas de base de lanzamiento de soporte a largo plazo (LTS) se eligen cada 12 semanas a partir del flujo de lanzamientos regulares. Cada 4 semanas se publican versiones estables que incluyen versiones posteriores que incluyen corrección de errores y mayor seguridad.

- Lanzamientos regulares (semanales)
  Esta línea de lanzamiento ofrece correcciones de errores y nuevas funciones rápidamente a los usuarios y desarrolladores de complementos que las necesitan. Por lo general, se realiza una entrega semanal.

Jenkins se distribuye como archivos WAR, paquetes nativos, instaladores e imágenes de Docker. La ruta es: https://www.jenkins.io/download/

Instalación

Jenkins se puede instalar en cualquier plataforma Windows, Linux, en contenedores Docker, Kubernetes, MacOS, ficheros WAR, Solaris, OmniOS, SmartOS, entre otros. Jenkins generalmente se ejecuta como una aplicación independiente en su propio proceso con el servidor de aplicaciones / contenedor de servlets Java integrado (Jetty). Jenkins también se puede ejecutar como un servlet en diferentes contenedores de servlets de Java, como Apache Tomcat o GlassFish. Antes de su instalación, tiene una serie de requisitos a nivel de hardware y software:

- Configuración mínima de hardware:
  - 256 MB de RAM
  - 1 GB de espacio libre en disco, si Jenkins se ejecuta como contenedor Docker, se recomienda un mínimo de 10 GB
- Configuración de hardware recomendada para equipos pequeños:
  - 1 GB + RAM
  - 50 GB + espacio libre en disco
- Requisitos de Software:
  - Java 8-Java Runtime Environment (JRE) o un Java Development Kit (JDK). *Nota: si ejecuta Jenkins como contenedor de Docker, no es necesario que cumpla con este requisito.*

[1] Ver anexo A

4.3 Configuración

La mayoría de los cambios de configuración de Jenkins se pueden realizar a través de la interfaz de usuario de Jenkins o mediante la configuración como complemento de código. Hay algunos valores de configuración que solo se pueden modificar mientras se inicia Jenkins. Esta sección describe esas configuraciones y cómo puede usarla.

La inicialización de Jenkins también se puede controlar mediante parámetros de tiempo de ejecución pasados como argumentos. Los argumentos de la línea de comandos pueden ajustar la red, la seguridad, la supervisión y otras configuraciones. Por otra parte, Jenkins ignora los parámetros de la línea de comandos que no comprende en lugar de producir un error, por lo que hay que tener cuidado al utilizar los parámetros de la línea de comandos y asegurarse de tener la ortografía correcta.

Algunos comportamientos de Jenkins están configurados con propiedades de Java. Las propiedades de Java se establecen desde la línea de comandos que inició Jenkins. Las asignaciones de propiedad utilizan el formulario: - *DsomeName=someValue* para asignar el valor *someValue* a la propiedad nombrada *someName*. Por ejemplo, para asignar el valor *true* a una propiedad testName, el argumento de la línea de comando sería -DtestName=true.

Jenkins dispone de varias opciones de configuración. Para acceder al menú de configuración es necesario seleccionar la opción "*Manage Jenkins*" y en la siguiente pantalla, seleccionar la opción "*Configure System*".

Pantalla de Configuración del sistema:

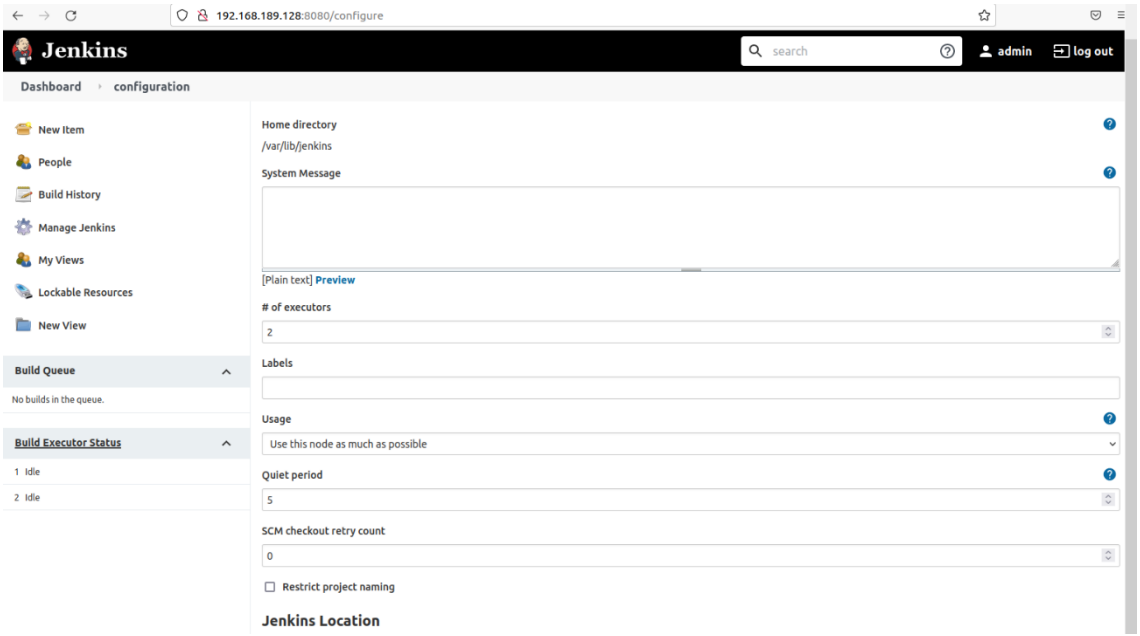


Ilustración 11. Jenkins. Configuración del sistema

Algunas de las opciones más importantes:

- Directorio JENKINS_HOME. Jenkins necesita algo de espacio en el disco para realizar compilaciones y mantener archivos. Puede verificar esta ubicación desde la pantalla de configuración de Jenkins (en este ejemplo: /var/lib/.jenkins). De forma predeterminada, se establece en ~ / .jenkins, pero se puede cambiar de la siguiente manera:
- Estableciendo la variable de entorno "JENKINS_HOME" en el nuevo directorio de inicio antes de iniciar el contenedor de servlet.
- Estableciendo la propiedad del sistema "JENKINS_HOME" en el contenedor de servlets.
- Estableciendo la entrada de entorno JNDI "JENKINS_HOME" en el nuevo directorio.

También se puede cambiar esta ubicación después de haber usado Jenkins por un tiempo. Para hacerlo, es necesario detener todas las tareas de Jenkins, y mover los contenidos del antiguo JENKINS_HOME al nuevo, estableciendo el nuevo JENKINS_HOME y reiniciando Jenkins.

- Número de ejecutores. Esto hace referencia al número total de ejecuciones de tareas concurrentes que pueden tener lugar en la máquina de Jenkins. Puede ser cambiado basado en requerimientos. Se recomienda mantener este número igual al número de CPU’s en las máquinas para un mejor rendimiento.

- Variables de entorno. Esta opción se utiliza para añadir variables de entornos personalizadas las cuales se aplicarán a todos las tareas (jobs). Son pares clave-valor a las que se puede acceder y utilizar en compilaciones siempre que sea necesario. URL de Jenkins. Por defecto apunta al localhost. Esta opción también es configurable (como se ha visto en apartados anteriores), en el wizard inicial o configuración inicial de Jenkins. Si ya se tiene configurado un nombre de dominio, se puede establecer, en caso contrario, la url con la IP de la máquina. Esto ayudará a configurar el esclavo y al enviar enlaces usando el correo electrónico, ya que puede acceder directamente a la URL de Jenkins usando la variable de entorno JENKINS_URL, a la que se puede acceder como ${JENKINS_URL}.

- Notificaciones vía Email. En esta opción se puede configurar los ajustes de un servidor SMTP para envíos de email a una lista de remitentes, cuando se realiza una tarea especifica en Jenkins.

4.4 Gestión

Para realizar la administración de Jenkins, en la pantalla principal, es necesario selección la opción “Manage Jenkins” en el lado izquierdo del menú, como aparece en la siguiente imagen:

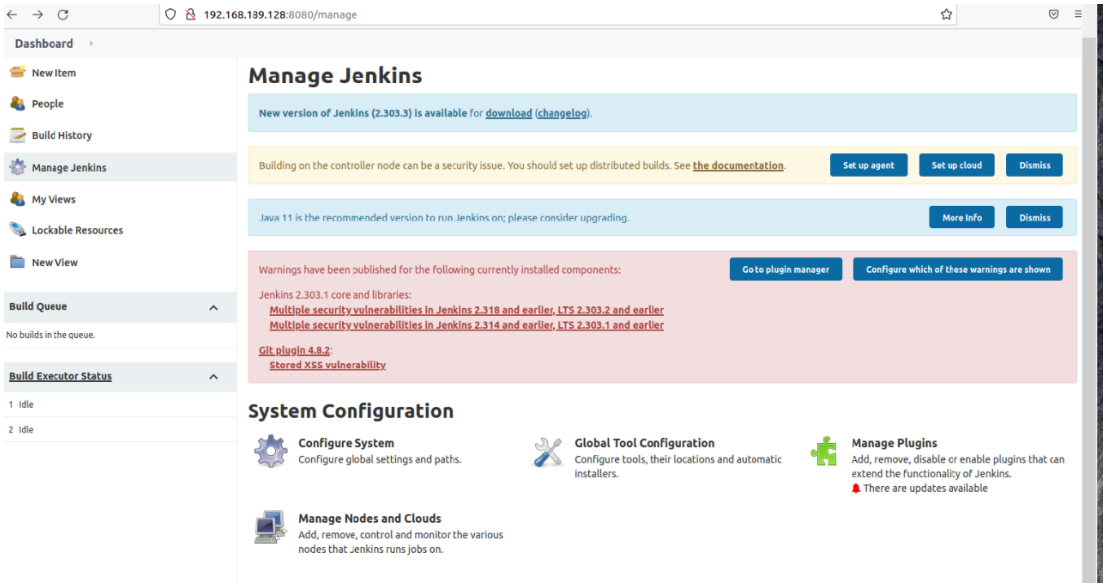


Ilustración 12. Jenkins. Gestión (parte 1)

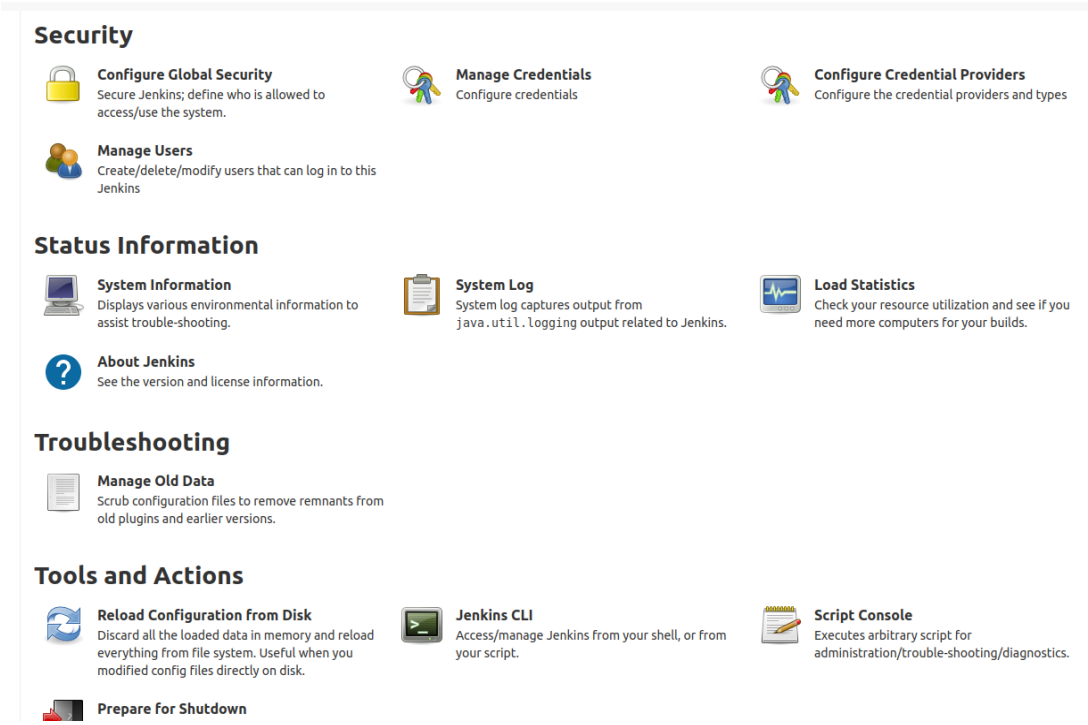


Ilustración 13. Jenkins. Gestión (parte 2)

En este menú se aprecian todas las opciones de configuración que ofrece Jenkins.

- Configuración del sistema. En esta opción se puede administrar las rutas de las diversas herramientas, que utilizaran las compilaciones, como los JDK, la versión de Ant y Maven, GIT, así como las opciones de seguridad, los servidores de correo electrónico y otros detalles de configuración de todo el sistema. Además, se puede añadir, borrar, habilitar o deshabilitar plugins que pueden extender la funcionalidad de Jenkins, y en las últimas versiones, se puede gestionar y monitorizar los nodos de ejecución de las tareas, así como gestionar su implementación en la nube.

- Seguridad. Define quien está permitido a usar o acceder al sistema, se puede configurar para que actúe como un servidor SSH, se puede configurar para que almacene quien se ha conectado al servidor y desde dónde, se pueden configurar las credenciales y desde este menú se puede crear, modificar o borrar usuarios que pueden iniciar sesión en Jenkins.

- Información de estado de Jenkins. En esta opción se muestra información sobre las propiedades del sistema, los plugins, las variables de entorno, uso de memoria, logs, estadísticas de recursos en uso para las compilaciones e información sobre Jenkins (versiones, licencias, información de dependencias, etc.).

- Resolución de problemas. Esta opción sirve para limpiar el sistema borrando los archivos de configuración y los restos de complementos antiguos y versiones anteriores. Además, es posible volver a un estado o configuración anterior, una vez eliminada esa configuración, para ello cuando hay cambios en la forma en que se almacena los datos en el disco, Jenkins usa la siguiente estrategia: los datos se migran a la nueva estructura cuando se cargan, pero los elementos o registros no se vuelven a guardar en el nuevo formato. Con esto, es posible volver a restaurar una configuración anterior.

- Herramientas y acciones. En este menú se encuentran opciones para cargar en Jenkins la configuración almacenada en un disco, muy útil en caso de que se haya modificado archivos de configuración y se quiera volver a una configuración especifica, opciones para acceder o gestionar el acceso a Jenkins desde la Shell o desde un script, consola de scripts (Groovy) para la administración, diagnostico o resolución de problemas, así como opciones para detener las ejecuciones de la compilaciones, de modo que el sistema pueda apagarse, eventualmente, de forma segura.

4.5 Despliegue continuo

Jenkins proporciona un buen soporte para proporcionar implementación y entrega continuas. Si se observa el flujo de cualquier desarrollo de software a través de la implementación, será como se muestra a continuación [26]:

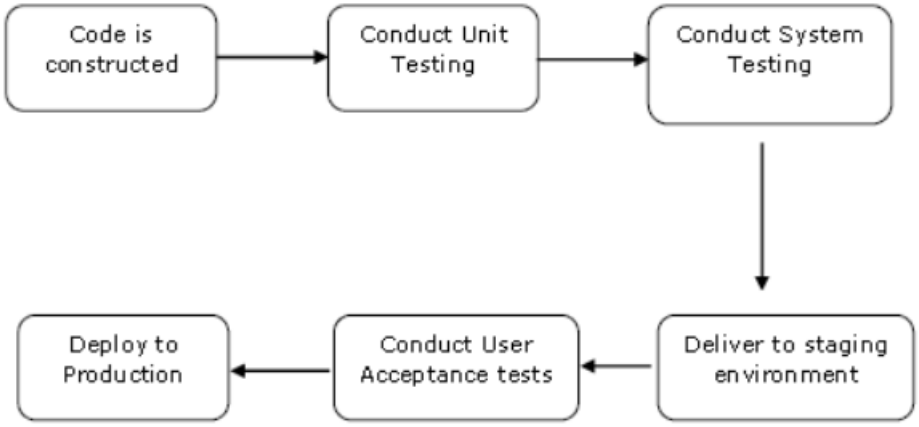


Ilustración 14. Fases integrantes del Despliegue continuo

La parte principal del despliegue continuo es garantizar que todo el proceso que se muestra en la imagen anterior esté automatizado. Jenkins logra todo esto a través de varios complementos, uno de ellos es el "Complemento de compilación en contenedor" o el “Complemento de compilación en tuberías”, entre otros complementos disponibles que pueden brindar una representación gráfica del proceso de despliegue continuo.

Llegados a este punto, se ejemplificará el despliegue continuo con un simple ejemplo que simula una prueba de control sobre una aplicación. Para ello primero se creará un job *myApplication* y otro *testApplication.*

1. Creación de la tarea *myApplication:*

a) Seleccionar la opción “New item”, introducir el nombre del job y a continuación, la opción “Freestyle project”, como se muestra en la siguiente imagen:

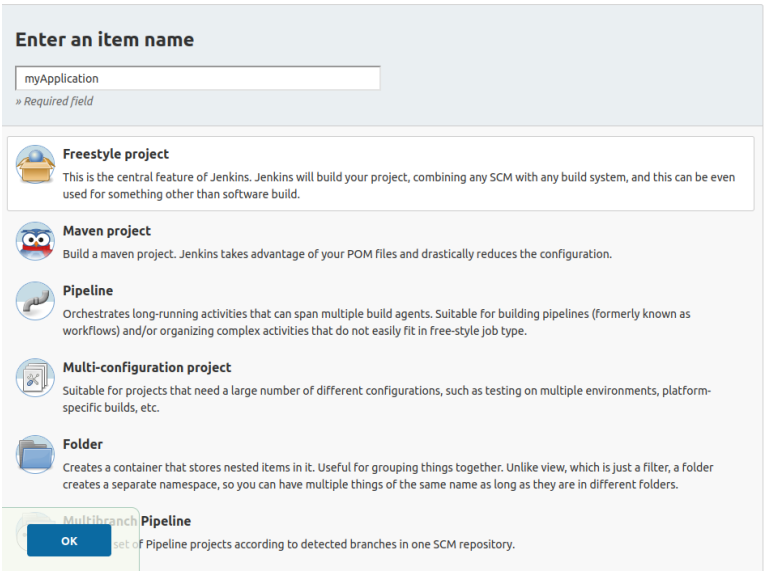


Ilustración 15. Jenkins. Creación de tareas: Pantalla inicial

b) La siguiente pantalla, indica todas las opciones configurables que la tarea ejecutaría, antes durante y después de la compilación de la aplicación. En este primer ejemplo se dejará las opciones por defecto, excepto la descripción:

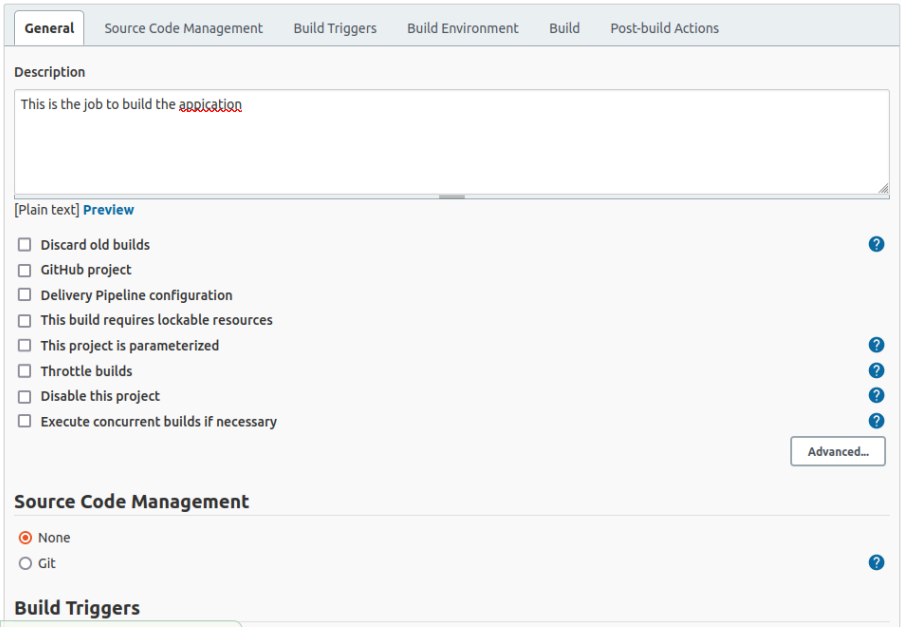


Ilustración 16. Jenkins. Creación de tareas: Configuración general

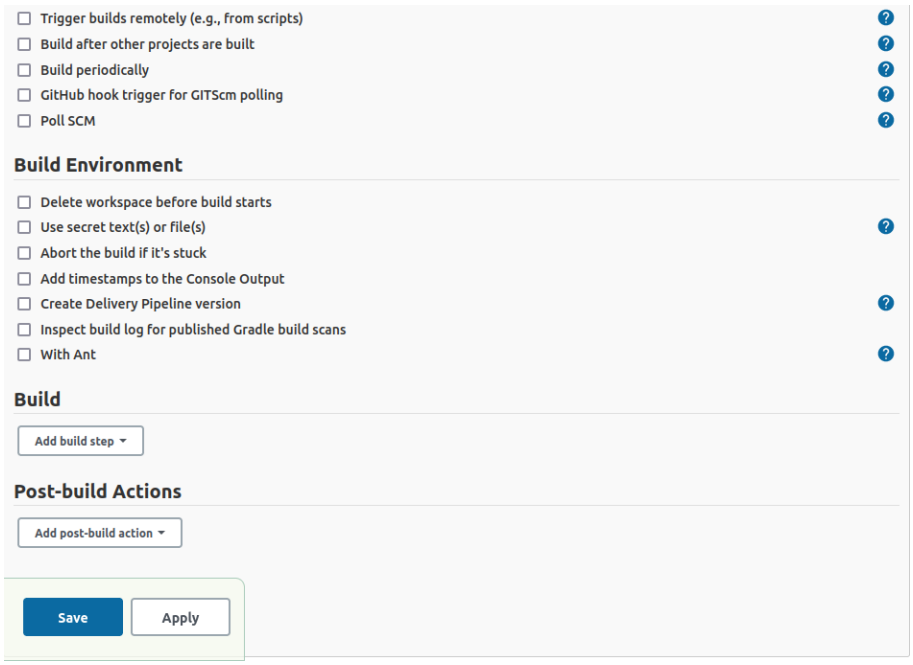


Ilustración 17. Jenkins. Creación de tareas: Configuración general 2

c) Se realiza la compilación para verificar que todo es correcto, con la opción "Build now" y en la opción "Console Output" se verifica el resultado:

Ilustración 18. Jenkins. Creación de tareas: Resultado del job

2. De la misma forma que el paso anterior, se crea el proyecto *testApplication* y se ejecuta.

3. En el proyecto "*myApplication" seleccionar* la opción "Configure" y a continuación la opción "Add post-build action", eligir "Build other projects" e introducir el nombre del proyecto bajo prueba, como se indica a continuación:

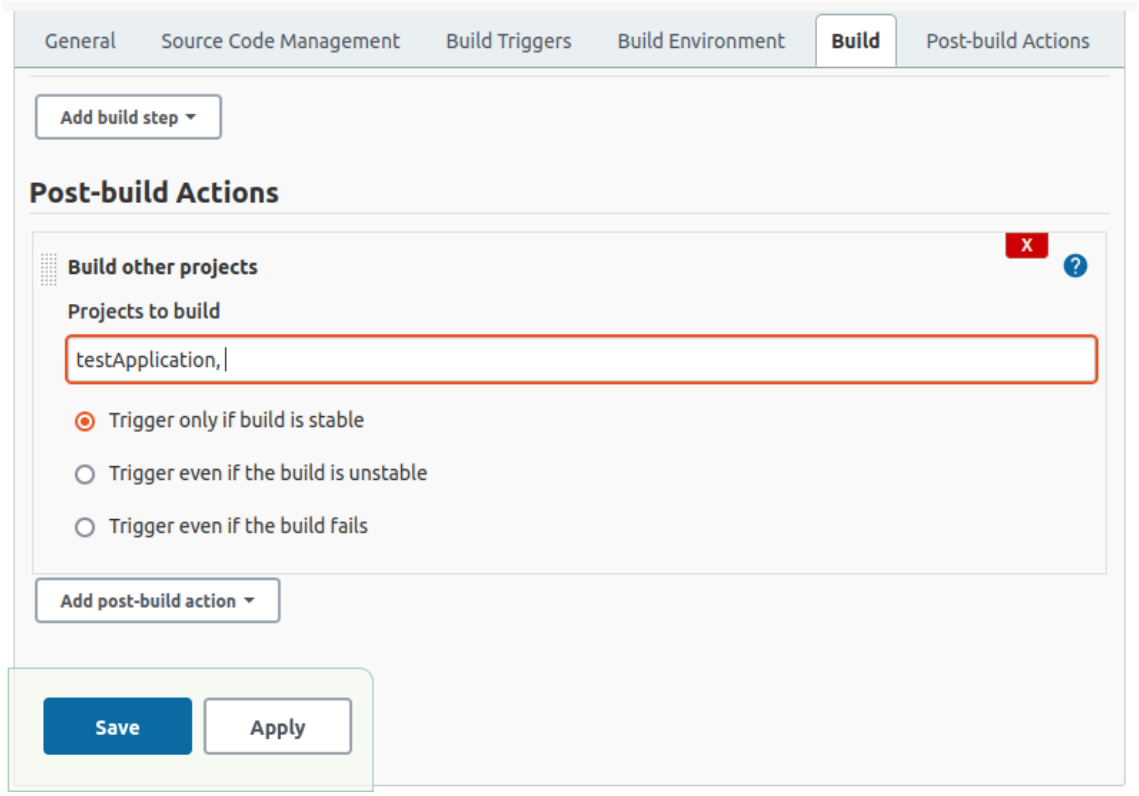


Ilustración 19. Jenkins. Ejemplo de despliegue de una aplicación

Tras aplicar y guardar los cambios, al volver a compilar el proyecto *myApplication* en la consola de salida muestra:

Ilustración 20. Jenkins. Resultados del despliegue

4. Para verificar el resultado anterior de manera gráfica, es necesario instalar el plugin comentado anteriormente: “Delivery pipeline plugin”, para ello, es necesario ir a: Manage → Manage Plugin’s. Y en la pestaña “Available” seleccionar e instar “Delivery Pipeline Plugin”:

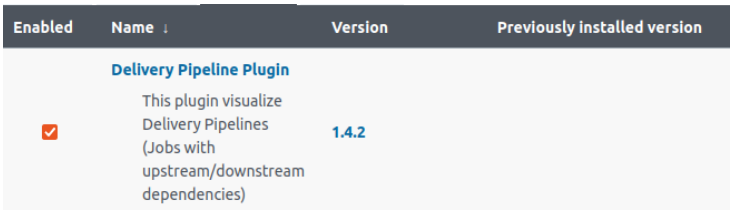


Ilustración 21. Jenkins. Plugin: Delivery Pipeline Plugin

Una vez instalado, en el Dashboard, seleccionar el símbolo “+”, y seleccionar la opción “Delivery Pipeline View”. Para este ejemplo, se puede dejar el resto de las opciones por defecto, excepto la de “Initial job”, en la que hay que introducir el proyecto en pruebas:

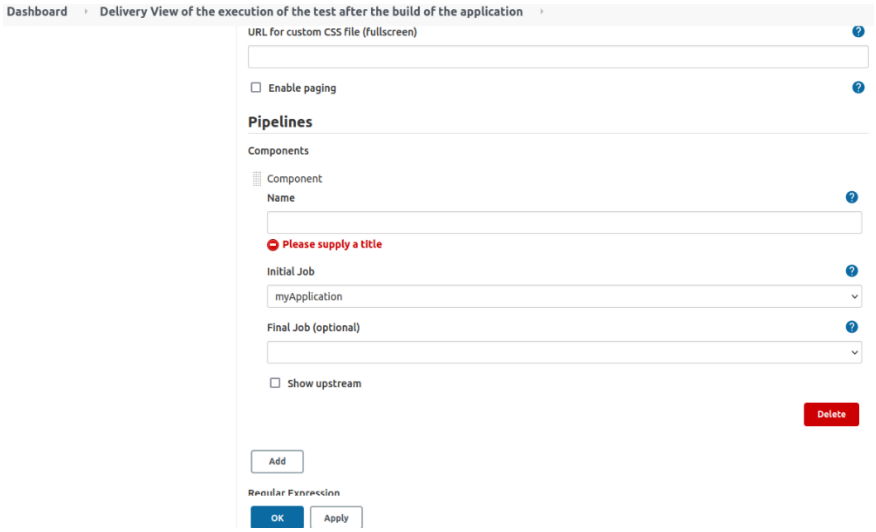


Ilustración 22. Jenkins. Integrando el plugin al job

Tras realizarse estos pasos, se tendrá una vista de todo el proceso de entrega y se podrá ver el estado de cada proyecto en todo el proceso, como muestra la siguiente imagen:

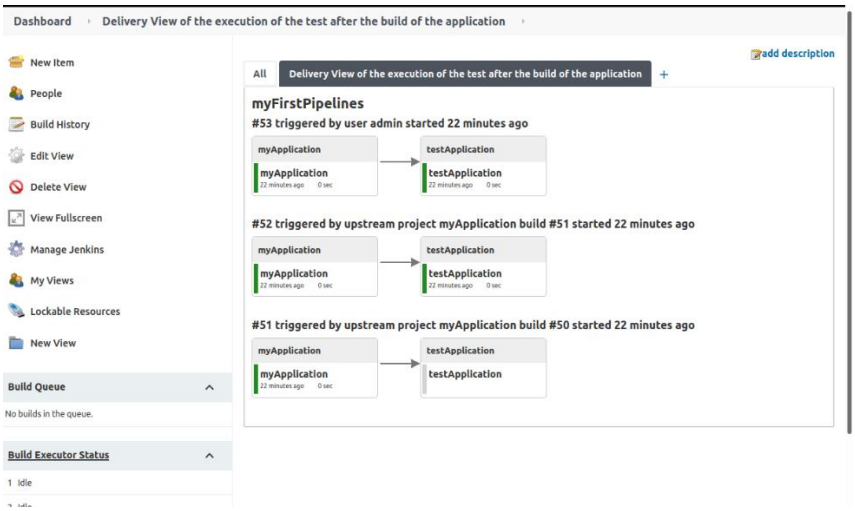


Ilustración 23. Jenkins. Resultado gráfico del despliegue

4.6 Pipelines

Jenkins Pipeline (o simplemente "Pipeline") es un conjunto de complementos que admite la implementación e integración de tuberías de entrega continua en Jenkins. Un pipeline de entrega continua (CD) es una expresión automatizada de su proceso para obtener software desde el control de versiones hasta sus usuarios y clientes. Cada cambio en su software (comprometido en el control de código fuente) pasa por un proceso complejo en su camino hacia su lanzamiento. Este proceso implica la construcción del software de una manera confiable y repetible, así como el progreso del software construido (llamado "compilación") a través de múltiples etapas de prueba e implementación.

Pipeline proporciona un conjunto extensible de herramientas para modelar canales de entrega simples a complejos "como código" a través de la sintaxis del lenguaje específico de dominio (DSL) de Pipeline.

La definición de Jenkins Pipeline se escribe en un archivo de texto (llamado Jenkinsfile) que, a su vez, se puede enviar al repositorio de control de código fuente de un proyecto. Esta es la base de "Pipeline-as-code"; tratar la canalización de CD como una parte de la aplicación para ser versionada y revisada como cualquier otro código.

Crear un Jenkinsfile y comprometerlo con el control de código fuente proporciona una serie de beneficios inmediatos:

- Crea automáticamente un proceso de construcción de *Pipeline* para todas las ramas y solicitudes de extracción.
- Revisión / iteración de código en *Pipeline* (junto con el código fuente restante).
- Auditoría del *Pipeline*.
- Fuente única para el *Pipeline*, que puede ser vista y editada por varios miembros del proyecto.

Un archivo Jenkins se puede escribir utilizando dos tipos de sintaxis: declarativa o con script. Pipeline declarativo es una característica más reciente de Jenkins Pipeline que proporciona características sintácticas más ricas que la sintaxis de basada en *scrips*, y está diseñado para facilitar la escritura y la lectura de código Pipeline.

Jenkins es, fundamentalmente, un motor de automatización que admite varios patrones de automatización. Pipeline agrega un poderoso conjunto de herramientas de automatización a Jenkins, lo que da soporte a casos de uso que abarcan desde una simple integración continua hasta una completa canalización de CD. Al modelar una serie de tareas relacionadas, los usuarios pueden aprovechar muchas características de Pipeline:

- Código: las canalizaciones se implementan en código y, por lo general, se registran en el control de código fuente, lo que brinda a los equipos la capacidad de editar, revisar e iterar sobre su canalización de entrega.
- Durable: las canalizaciones pueden sobrevivir a reinicios planificados y no planificados del controlador Jenkins.
- Pausable: las Pipilines pueden, opcionalmente, detenerse y esperar la intervención humana o la aprobación antes de continuar con la ejecución de la Pipeline.

- Versátil: las Pipeline admiten requisitos de CD complejos del mundo real, incluida la capacidad de bifurcar / unir, hacer bucles y realizar trabajos en paralelo.
- Extensible: el complemento Pipeline admite extensiones personalizadas para su DSL y múltiples opciones para la integración con otros complementos.

El siguiente diagrama de flujo es un ejemplo de un escenario de CD fácilmente modelado en Jenkins Pipeline:

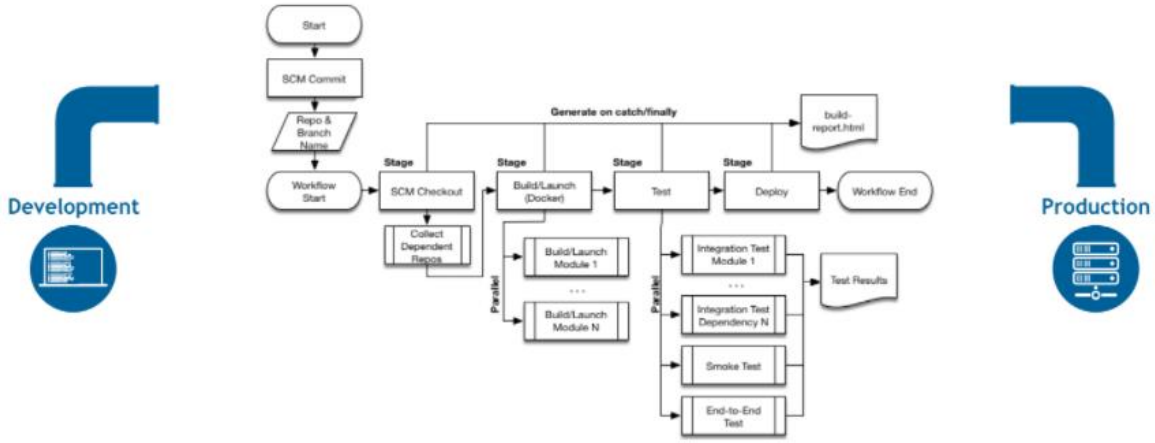


Ilustración 24. Modelado de un Pipeline

Los siguientes conceptos son aspectos clave de Jenkins Pipeline, que se relacionan estrechamente con la sintaxis de Pipeline:

- *Pipeline*. Un Pipeline es un modelo definido por el usuario de un pipeline de despliegue continuo. El código de una canalización define todo el proceso de compilación, que normalmente incluye etapas para compilar una aplicación, probarla y luego entregarla.
- *Nodo*. Un nodo es una máquina que forma parte del entorno de Jenkins y es capaz de ejecutar un Pipeline.
- Stage. Un bloque de etapa define un subconjunto conceptualmente distinto de tareas realizadas a lo largo de todo el Pipeline (por ejemplo, las etapas "Construir", "Probar" e "Implementar"), que es utilizado por muchos complementos para visualizar o presentar el estado / progreso de Jenkins Pipeline.
- *Step.* Define una sola tarea. Básicamente, un *step* le dice a Jenkins qué hacer en un momento particular en el tiempo [27].

Ejemplo de un Pipeline declarativo:

```
Jenkinsfile (Declarative Pipeline)
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                //
            }
        }
        stage('Test') {
            steps {
                //
            }
        }
        stage('Deploy') {
            steps {
                //
            }
        }
    }
}
```

Ejemplo de un Pipeline con *scripts:*

```
Jenkinsfile (Scripted Pipeline)
node {
    stage('Build') {
        //
    }
    stage('Test') {
        //
    }
    stage('Deploy') {
        //
    }
}
```

DESARROLLO DE UN CASO PRÁCTICO

5.1 Planteamiento del problema

En este apartado se realiza un estudio de un caso práctico en el que se hace uso de integración y despliegue continuo. Se instalará los componentes y se configuraran adecuadamente para que se lleve a cabo el ciclo de despliegue esperado. Toda la instalación y configuración de los componentes se realizará sobre un servidor local. Se utilizará un programa sencillo ya que solamente tiene como objeto mostrar el resultado del ciclo de integración continua, aplicando automáticamente los cambios en el servidor con la aplicación ejecutándose. En Jenkins se establecerán diversas tareas para que en primer lugar compruebe el repositorio de control de versiones cada cierto tiempo, realice test unitarios y funcionales tras la compilación.

5.2 Tecnologías y herramientas

Para implementar el caso práctico se hará uso de las siguientes herramientas/plataformas:

- GIT

Para el desarrollo del caso práctico se utilizará como sistema de control de versiones GIT, si bien no es el único sistema de control de versiones, es el más popular siendo usado por la mayoría de los desarrolladores, ya que permite acciones como guardar el trabajo avanzado, coordinar el trabajo para múltiples desarrolladores en un solo proyecto, manejar diferentes versiones de aplicación en desarrollo, conciliar conflictos en los cambios de un archivo.

Los comandos básicos de Git son los siguientes:

- *git init*. Crea un nuevo repositorio local GIT. Usando *git init* [nombre del proyecto] también se puede crear un repositorio dentro de un directorio especificando el nombre del proyecto.
- *git clone.* Se utiliza para clonar un repositorio.
- *git add.* Se utiliza para agregar archivos al área de preparación.
- *git commit*. Se usa para crear un cambio que se guardará en el directorio git.
- *git config.* Se usa para establecer una configuración específica de usuario, podría ser el email, usuario o tipo de formato.
- *git status.* Se utiliza para que se muestre la lista de archivos que se ha cambiado, junto con archivos que serán preparados y confirmados.
- *git push.* Se sa para enviar confirmaciones a la rama maestra/principal del repositorio remoto.
- *git remote.* Permite ver todos los repositorios remotos.

- *git checkout*. Permite crear ramas y navegar entre ellas.
- *git pull.* Se utiliza para fusionar todos los cambios que se ha hecho en el repositorio local con el directorio de trabajo local.

GIT es el sistema de control de versiones más usado, como se muestra en la siguiente imagen [28]:

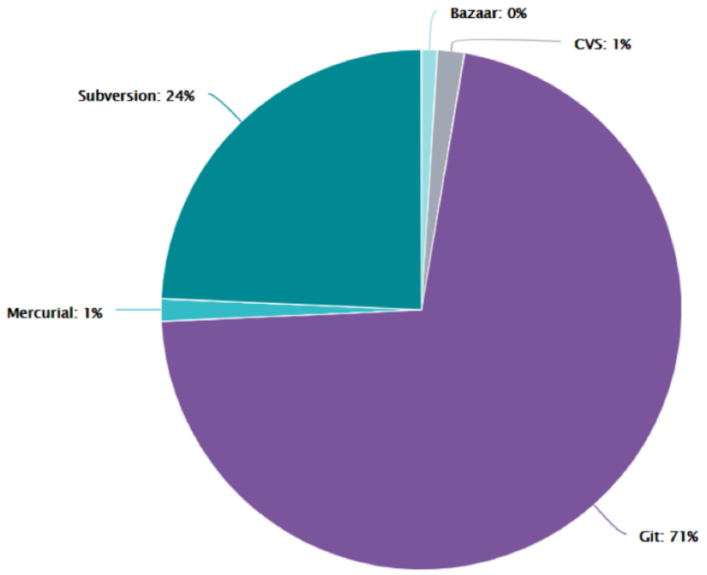


Ilustración 25. Sistemas de control de versiones más populares a nivel global

En Git son importantes las ramas. Todo repositorio va a contener una línea de tiempo sobre la cual se marcarán los cambios que se hagan sobre los archivos, ésta sería una Rama. Una Rama son los registros donde se marcan los cambios que haga uno o varios programadores en un conjunto de archivos en una línea de tiempo (estas características se pueden configurar).

- Gitflow

En el caso de estudio, para que se asemeje lo más posible a un entorno real y actual, se establecerá Gitflow, que consiste en un manejo adecuado de las ramas dentro de los repositorios de Git. Existen diferentes flujos de trabajo a seguir para generar menos conflictos entre cambios según las mejores prácticas y la lógica de trabajo del equipo del proyecto o iniciativa. Gitflow es uno de esos flujos de trabajo y se basa en cinco palabras clave durante la construcción de una aplicación y sus respectivas entregas de valor:

- Máster: Es la rama principal de proyecto, donde va el código para generar la aplicación en entornos de producción. Sobre ella no se debería desarrollar.
- Hotfix: Este tipo de rama se crea para la solución de un incidente. Debe ser generada a partir de Máster. Hotfix se utiliza en ambientes en producción para resolver incidencias.
- Develop: En esta rama se tienen las nuevas funcionalidades de la aplicación. No es aconsejable hacer commit (guardar cambios) directamente sobre ella,

excepto si son cambios pequeños y que no afecten la lógica, por ejemplo, cambiar un texto.

- Feature: Rama generada a partir de la rama develop. Sobre ella se harán los desarrollos de nuevas funcionalidades. Una vez esté desarrollada la funcionalidad la rama feature podrá ser mezclada con Develop para que los cambios queden allí sincronizados, y finalmente se procede a borrar la rama feature. Es aconsejable tener esta rama sincronizada con los cambios que se agreguen a la rama develop para evitar conflictos.
- Release: Esta rama utiliza para las entregas a producción o ambiente real. Su propósito es habilitar que en ella se hagan las pruebas del cliente y propender por la calidad de la entrega. Una vez se termine con las pruebas y sean exitosas se mezcla con la rama master. Y se sale a producción.

Un ejemplo del flujo se puede observar en la siguiente imagen:

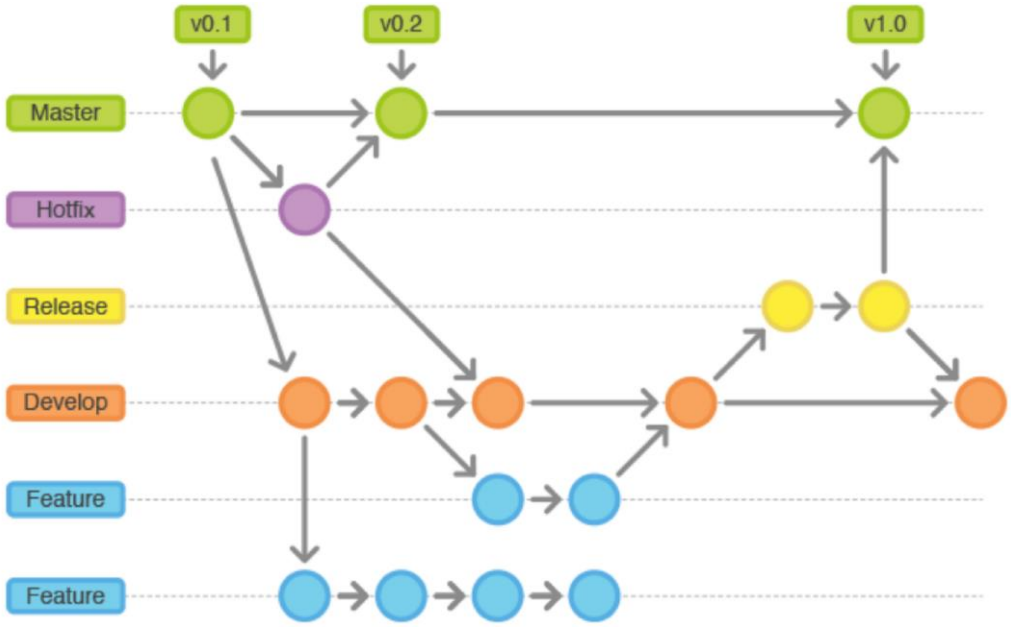


Ilustración 26. Flujo de git

- GitHub.

Como gestor de repositorio se utilizará GitHub, ya que cuenta con un servicio web y un sistema de control de versiones también basado en Git. Se apoya sobre el sistema de control de versiones Git. Así, se puede operar sobre el código fuente de los programas y llevar un desarrollo ordenado. Se ideó para el alojamiento de proyectos de código abierto, es una plataforma de colaboración, ya que ayuda a los desarrolladores a comunicarse de manera efectiva en su código. GitHub es tan fácil de usar, que incluso algunas personas usan GitHub para administrar otro tipo de proyectos – como escribir libros. Además de esto, cualquier persona puede inscribirse y ser hospedar un repositorio de código público completamente gratuito, el cual hace que GitHub sea especialmente popular con proyectos de fuente abierta.

Se opta por esta solución, ya que se trata de una plataforma con:

- Servicio gratuito, aunque también tiene servicios de pago.
- Búsqueda muy rápida en la estructura de los repos.
- Amplia comunidad y fácil encontrar ayuda.
- Ofrece prácticas herramientas de cooperación y buena integración con Git.
- Fácil integrar con otros servicios de terceros.
- Trabaja también con TFS, HG y SVN.

En cuanto a popularidad, la última encuesta de desarrolladores de Stack Overflow de 2020, el 82,8% de los encuestados dicen que usan GitHub, y solo el 37% dicen que usan GitLab (https://insights.stackoverflow.com/survey/2020#technology-collaboration-tools-all-respondents) :

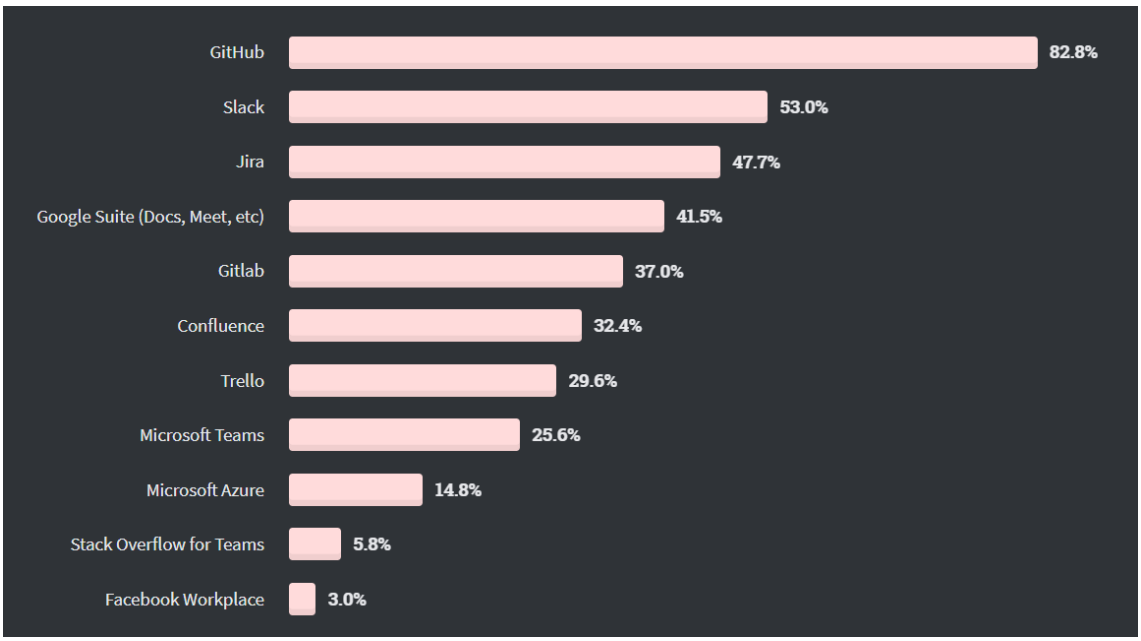


Ilustración 27. Gestor de repositorios más populares

- IntelliJ IDEA

IntelliJ IDEA es un IDE (entorno de desarrollo integrado) de código abierto para desarrollar aplicaciones. Es desarrollado y mantenido por Jetbrains, una compañía también popular para desarrollar PyCharm IDE. Se estima que más del 70% de los desarrolladores de Java prefieren IntelliJ IDE sobre otros IDE como Eclipse.

- Gradle

La herramienta elegida para la construcción del código será Gradle, ya que como se ha comentado en apartados anteriores mejora, con respecto a Maven, las siguientes funcionalidades: el lenguaje (no emplea el lenguaje XML, sino que se basa en DSL), gestión del ciclo de vida (añade la capacidad de soportar todo el proceso de vida del software) y es más personalizable.

- GitKraken

Esta herramienta es una potente y elegante interfaz gráfica multiplataforma para git. Permite que de una forma muy sencilla se pueda llevar el completo seguimiento de los repositorios, ver ramas, tags, crear nuevos, todo el historial de trabajo, commits, etcétera.

5.3 Instalación y configuración del entorno

- Instalación de Git[2]

Como en este caso de estudio, el sistema operativo utilizado es una distribución Linux (Ubuntu 20.04) Git ya está instalado por defecto. Se puede comprobar mediante el siguiente comando:

```
ubuntu@ubuntu:~$ git --version
git version 2.25.1
ubuntu@ubuntu:~$
```

Todo se encuentra disponible en los repositorios predeterminados, de modo que se puede actualizar el índice local de paquetes y luego instalar los paquetes pertinentes mediante:

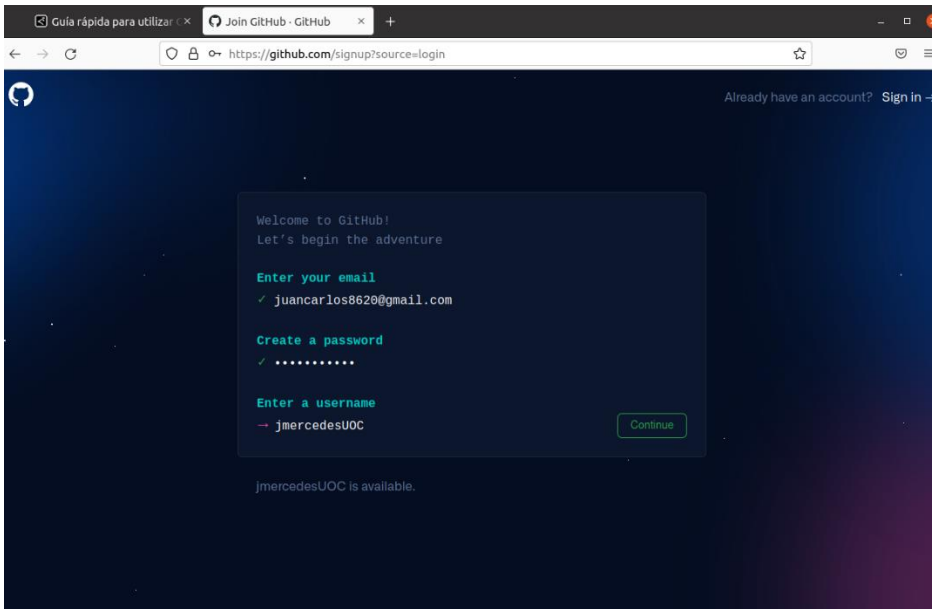
sudo apt update

sudo apt install libz-dev libssl-dev libcurl4-gnutls-dev libexpat1-dev gettext cmake gcc

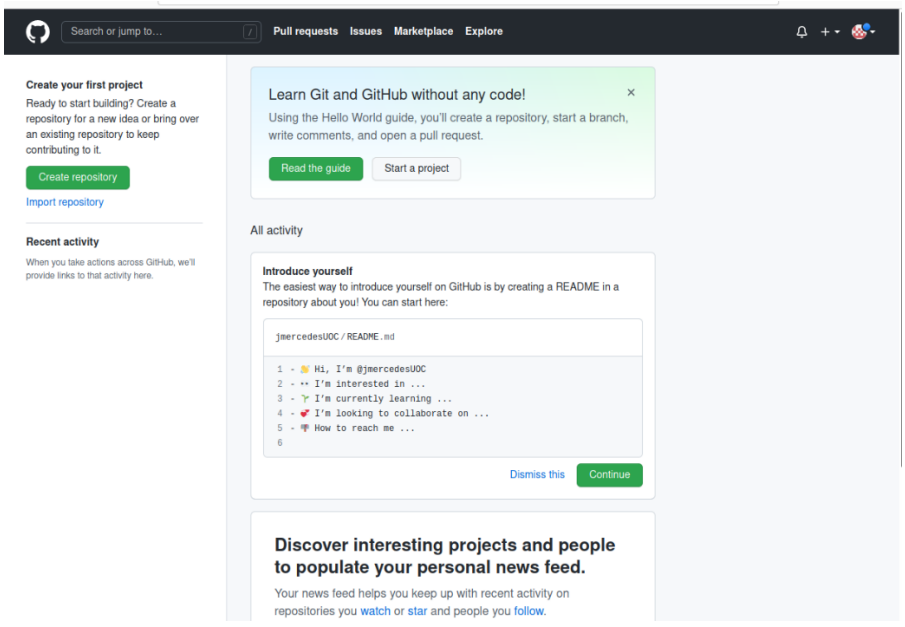
[2] Anexo B

- GitHub

Para utilizarlo, es necesario registrarse desde su propia web (https://github.com/), introduciendo los datos requeridos, como aparece a continuación:

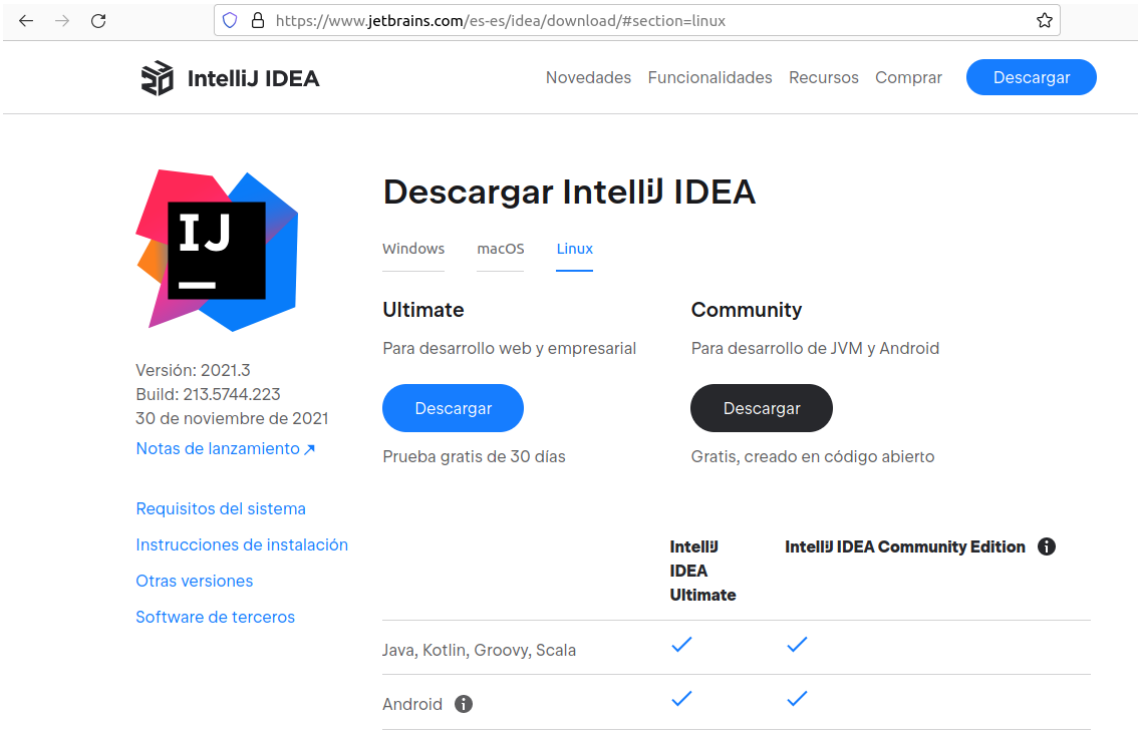


Una vez registrados la primera pantalla que se visualiza:

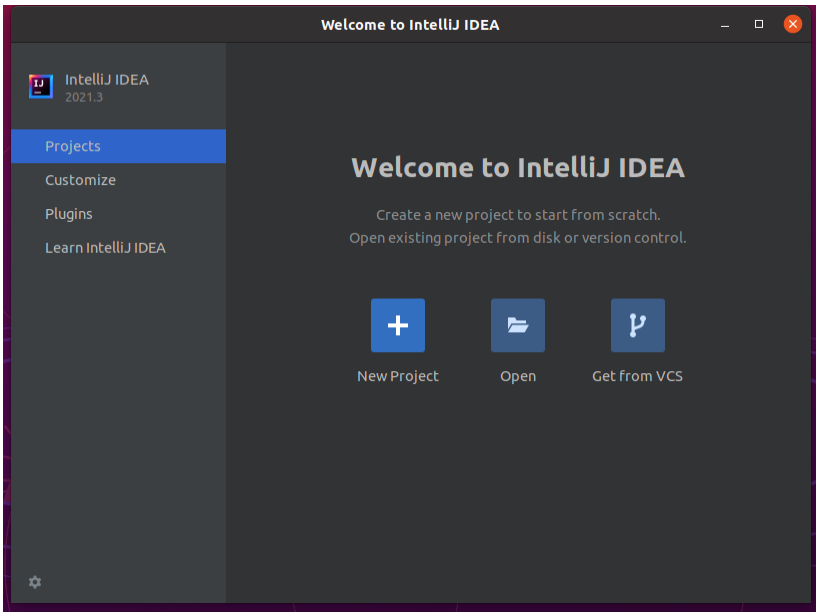


- Intellij

Existen varias formas de instalar Intellij. En este caso, se optará por descargar directamente desde la web (https://www.jetbrains.com/es-es/idea/download/#section=linux), la versión gratuita (Community):

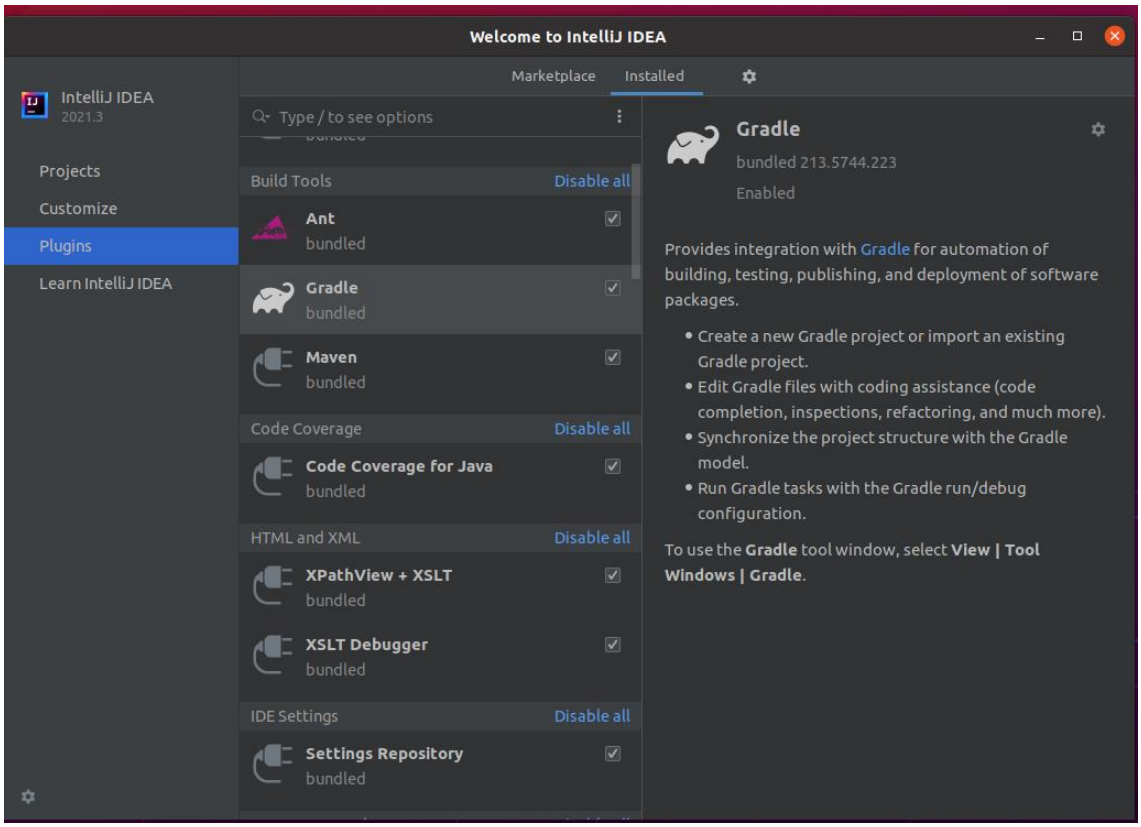


Para instarlo es necesario descomprimir el fichero descargado, situarse en la carpeta bin y ejecutar el script de instalación (./idea.sh). A continuación, ya se puede visualizar la primera pantalla de intellij:



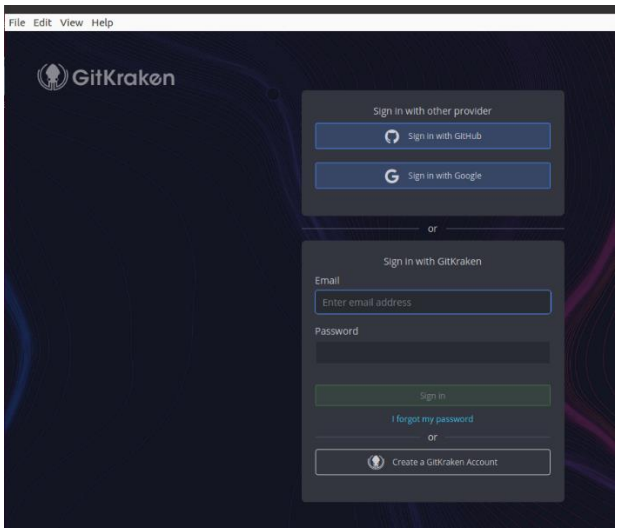
- Gradle

No es necesario realizar ninguna acción para instalar Gradle ya que viene por defecto en intellij, como se muestra a continuación (en caso de que no esté preinstalado, desde la opción plugins se puede instalar):



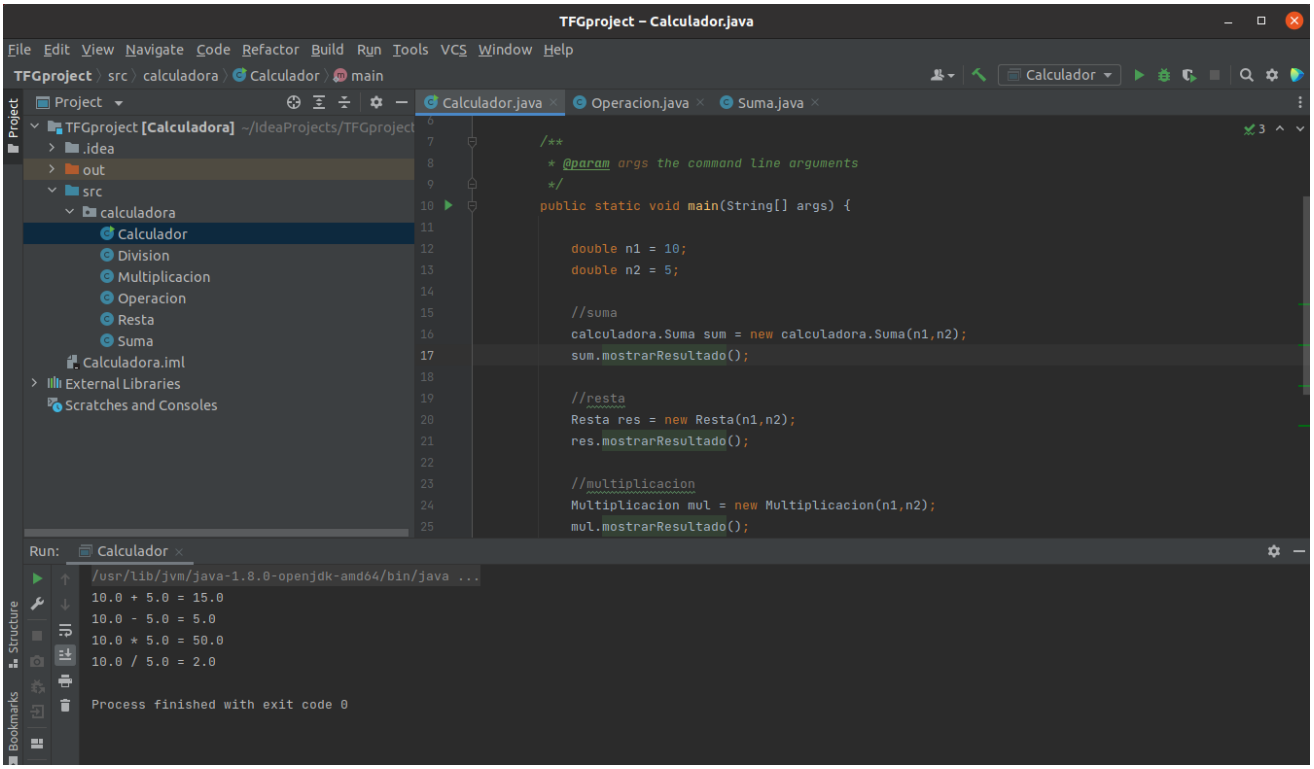
- GitKraken

Para descargar GitKraken, se irá a su web oficial (https://www.gitkraken.com/) y, en este caso, descargar la versión disponible para Ubuntu. Para instalar, basta con hacer doble click, una vez finalizado, se muestra la primera pantalla de gitlab:



- Programa

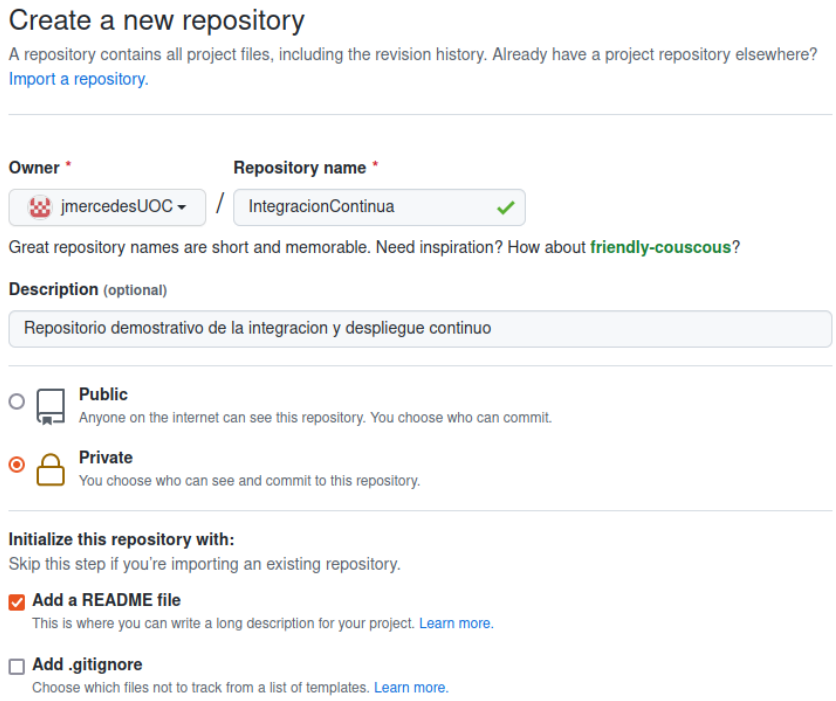
Se utilizará un simple programa escrito en Java. Se trata de un programa “Calculadora”. En la captura adjunta se muestra la correcta ejecución del programa, pasándole argumentos válidos:



5.4 Desarrollo e integración continua con Jenkins

- Crear repositorio

En este caso se creará el repositorio desde GitHub, seleccionando la opción “Create repository” como se muestra a continuación (este mismo paso se podría realizar desde local con el comando *init*):



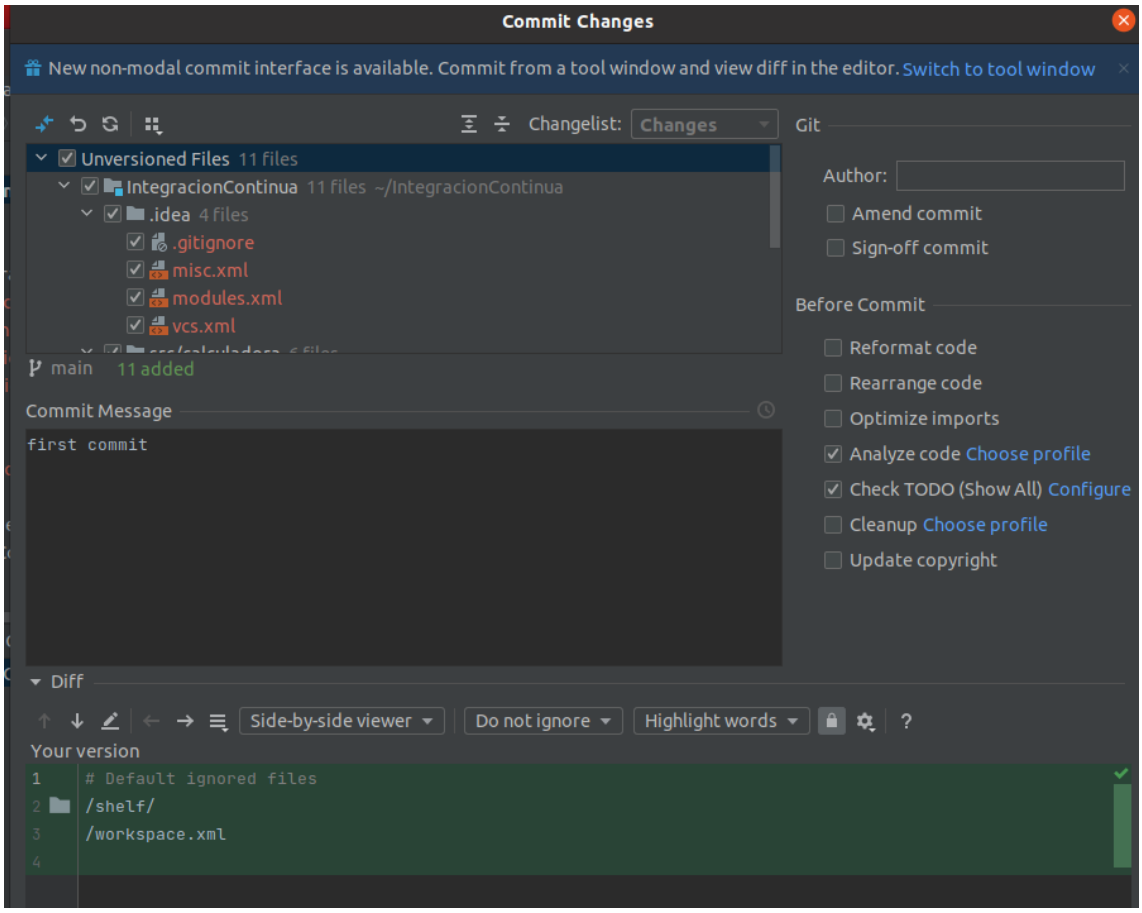
- Clonar repositorio

El siguiente paso es clonar el repositorio a local. Una vez clonado el repositorio, se añadirá a éste el programa que se utilizará para el estudio. Para poder ejecutar correctamente los comandos *git* es necesario usar SSH. Para clonar el repositorio vía SSH es necesario utilizar una clave pública. Los detalles en cuanto a criptografía de clave pública (también llamada simétrica), exceden del alcance del TFG, sin embargo, se puede generar una clave simétrica utilizando el comando: *ssh-keygen*.

```
usuario@ubuntu-20:~$ git clone git@github.com:jmercedesUOC/IntegracionContinua.git
```

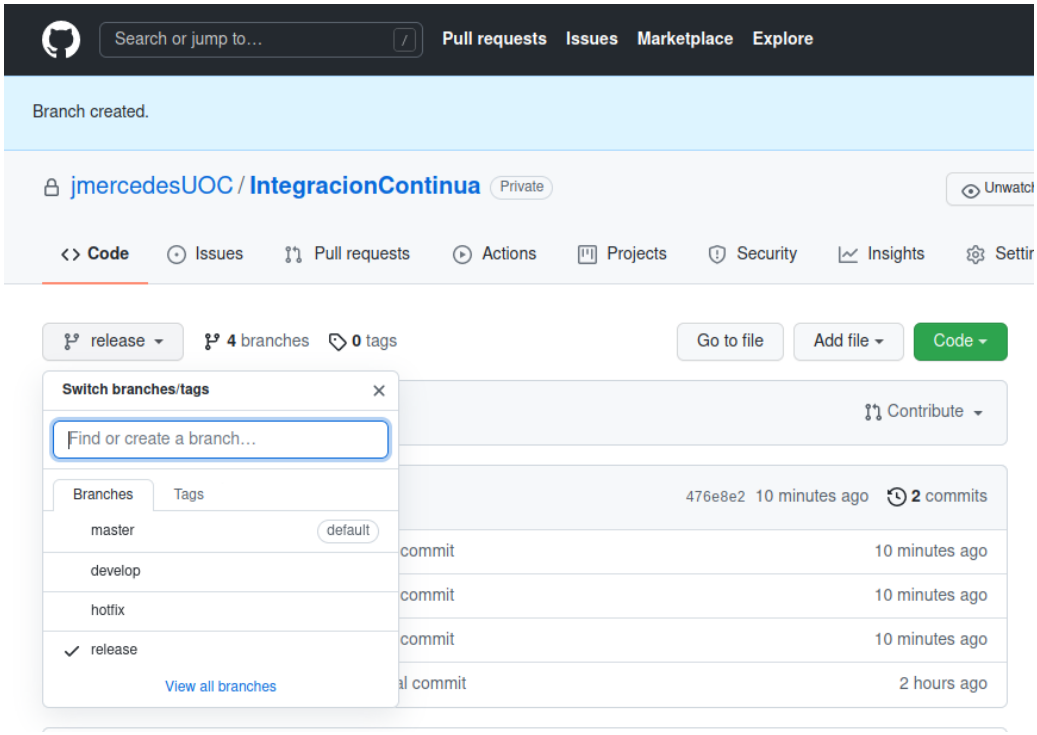
- Cargar el proyecto en Intellij, comitear y subir a GitHub

Desde Intellij, abrir el proyecto clonado y desde las opciones *Git* en el mismo Intellij seleccionar la opción *commit*, los ficheros que se quieran subir al repositorio remoto y finalmente la opción *push:*



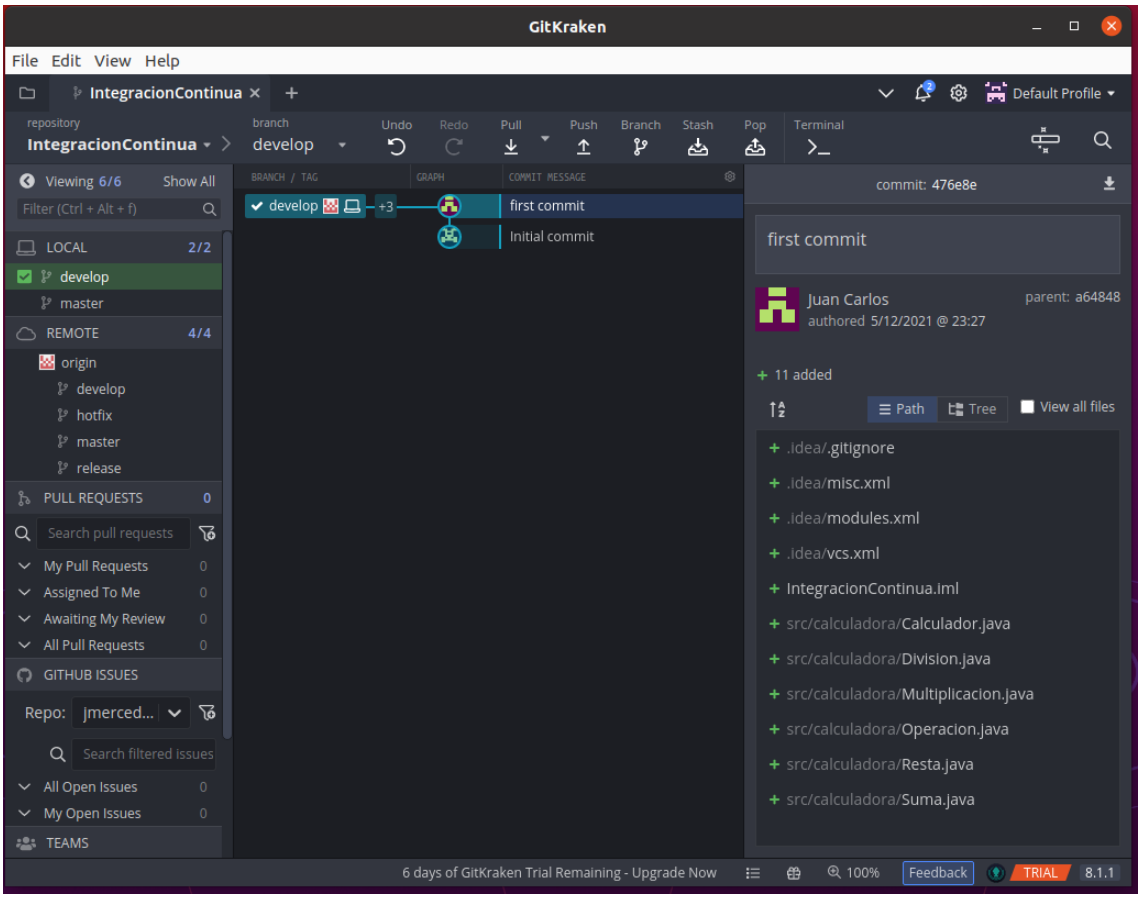
- Crear ramas en GitHub

Para llevar a cabo el GitFlow descrito en el apartado 5.2 se crearán las diferentes ramas en GitHub:



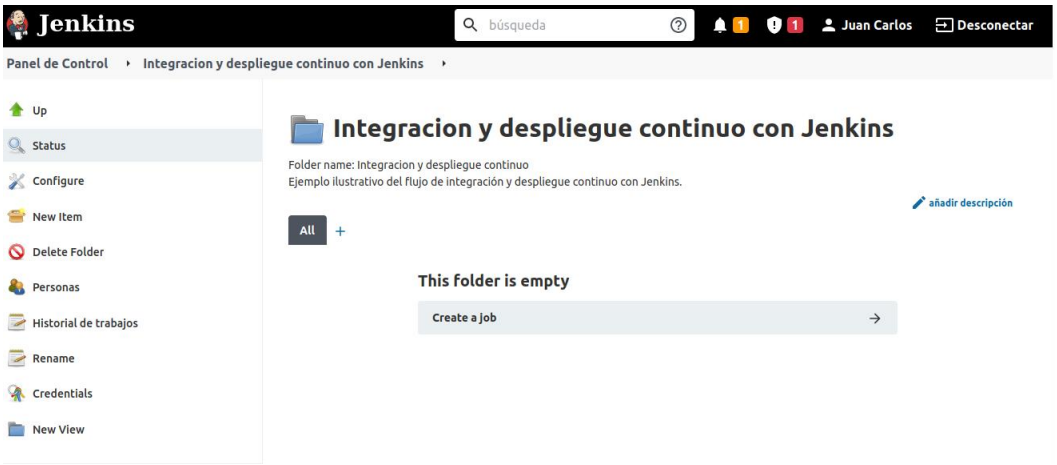
- Clonar el proyecto en GitKraken

Todos los pasos descritos se pueden realizar de diversas maneras para llegar al mismo fin. En este caso, se ha realizado un *pull* desde la misma terminal para actualizar el repositorio remoto creado previamente en GitHub. A continuación, se abre el proyecto desde GitKraken a través de su opción *Open repo* del menú *File***.** Para poder hacer actualizaciones en el repositorio local y subirlo al repositorio remoto, al igual de como se hizo en GitHub, es necesario añadir en GitKraken la clave pública (*File > Preferences > SSH*). En la siguiente imagen se muestra como se ha abierto el repositorio.

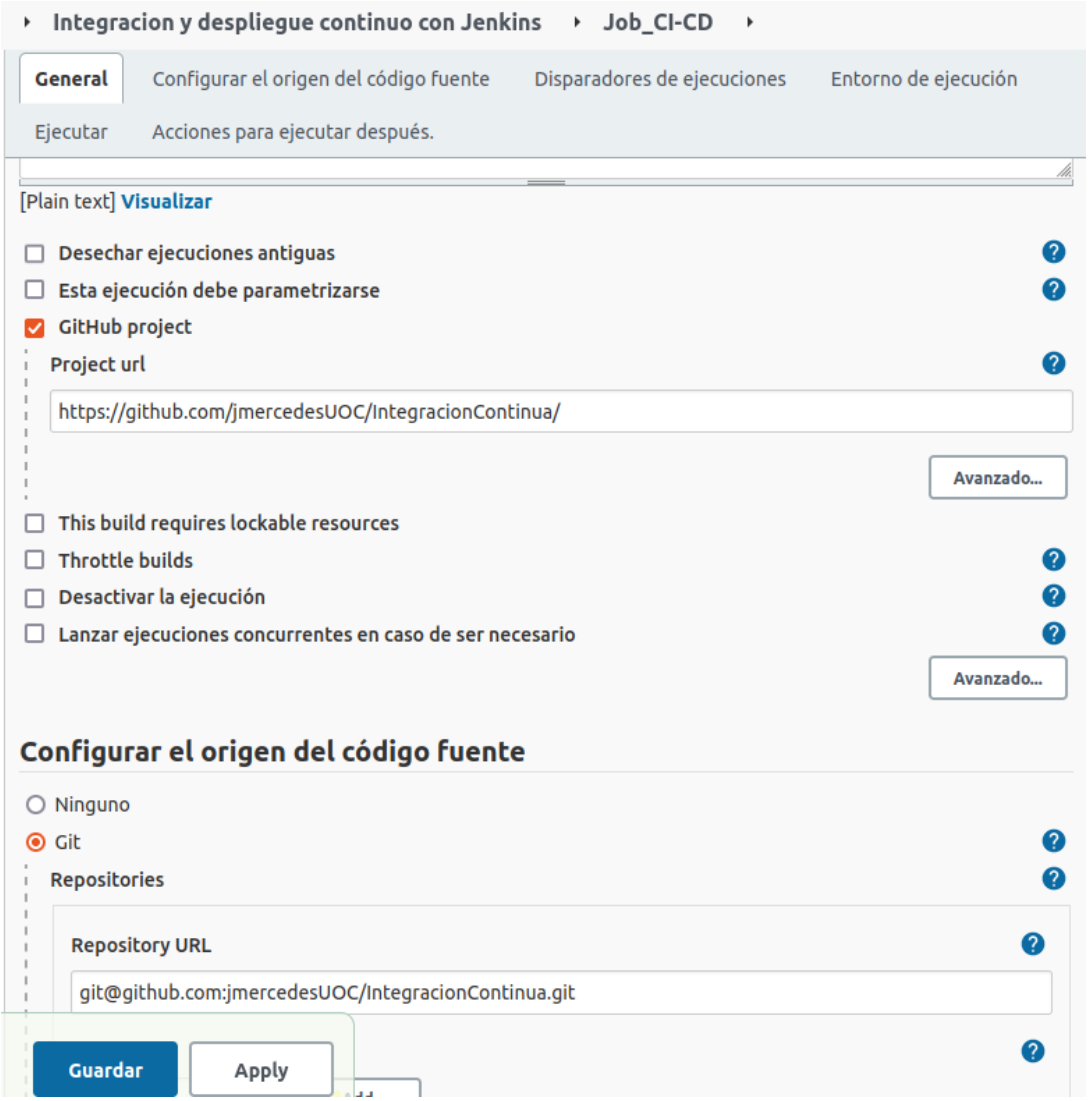


- Integrar Jenkins con GitHub

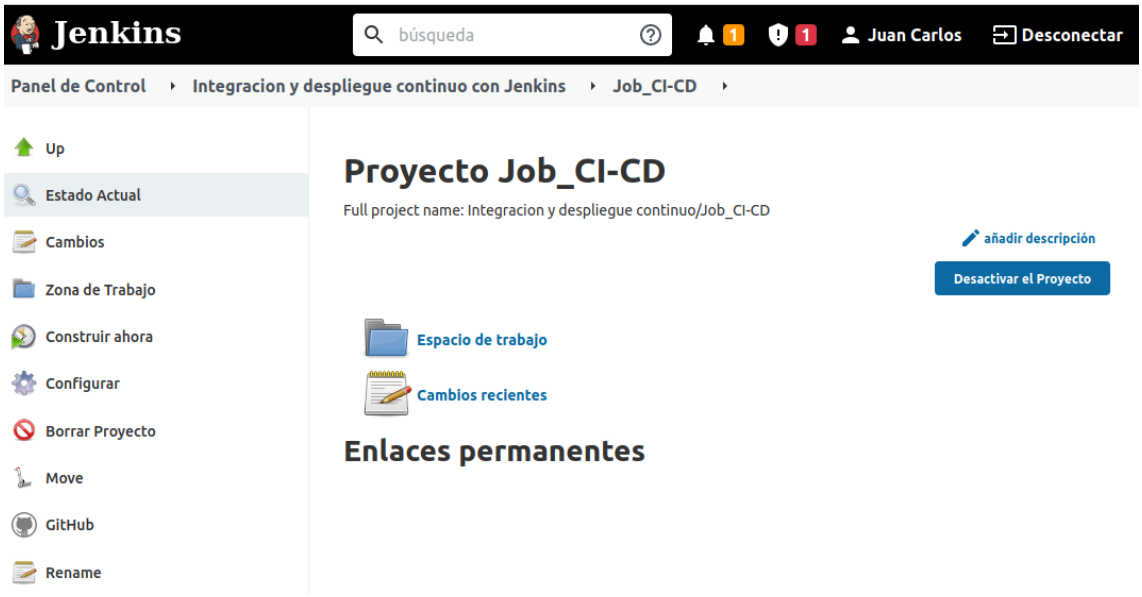
Una vez instaladas las diferentes herramientas y creado el repositorio, es hora de integrar el repositorio remoto con Jenkins. Para ello se creará un nuevo job de forma similar a como se ha realizado previamente, pero en este caso seleccionando la opción de Git y GitHub como se muestra a continuación:



Se crea el nuevo job y las opciones generales se introduce la ruta al repositorio remoto y además la url del proyecto como se muestra a continuación (para poder configurar el origen del código fuente, nuevamente es necesario hacer uso de la clave publica y añadirla en la ruta: */var/lib/Jenkins/.ssh/*):

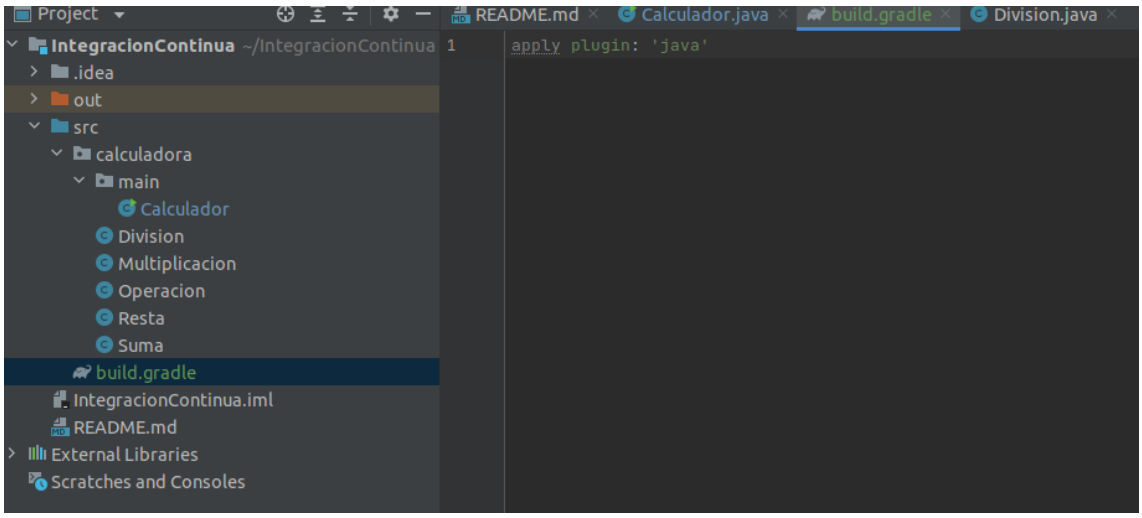


Tras guardar los cambios, en la vista del job se observa como este queda integrado con GitHub:

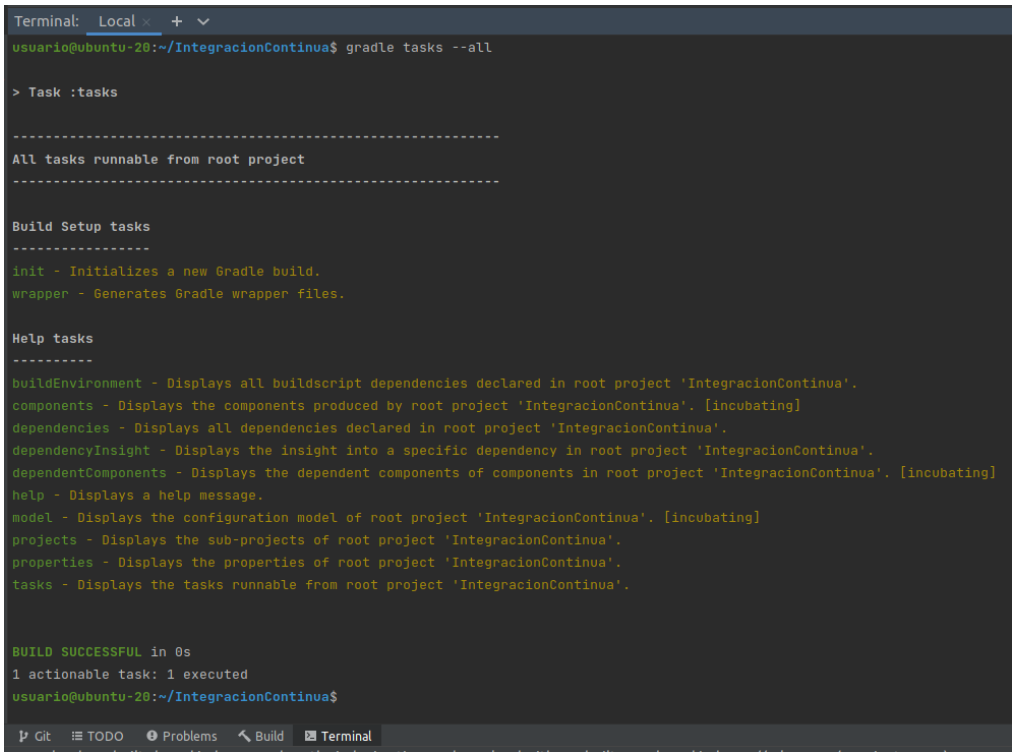


- Compilar el proyecto usando gradle

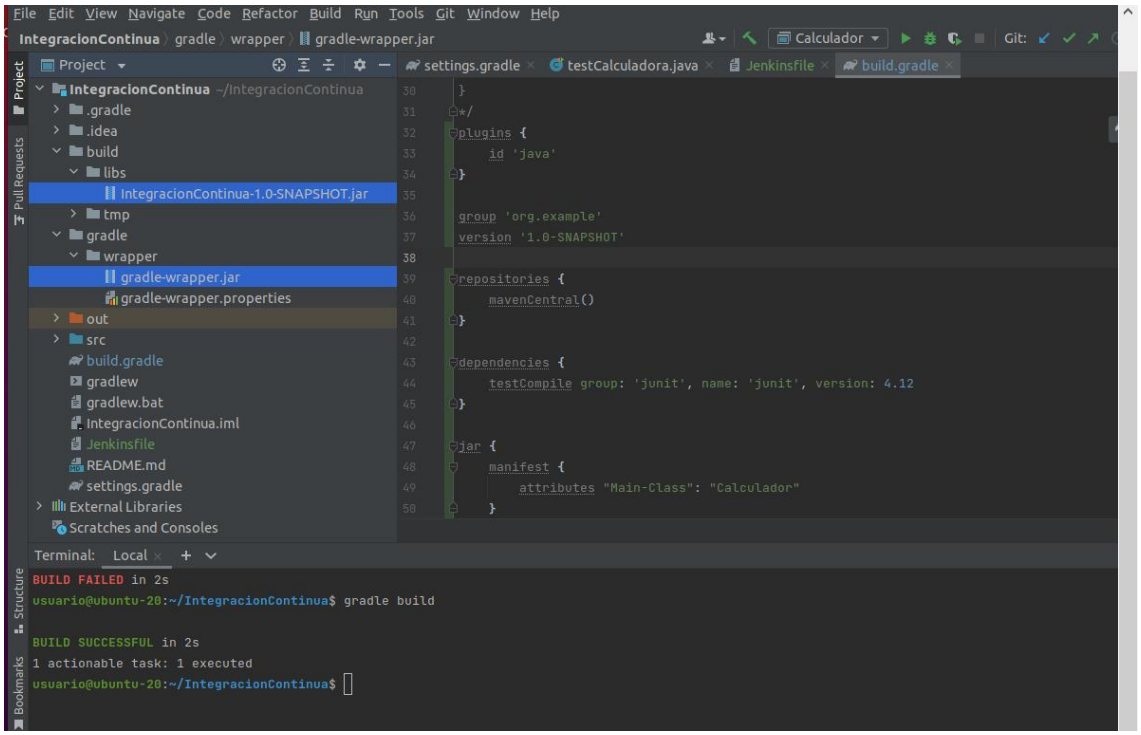
En este punto se muestra como gradle automatiza la compilación del código. Para ello lo primero es crear el fichero *build.gradle* que contendrá las sentencias necesarias para lleva a cabo la compilación, u otras tareas que se deseen automatizar. Para indicarle a gradle que utilice el plugin, de java en este caso, se utiliza la siguiente sentencia:



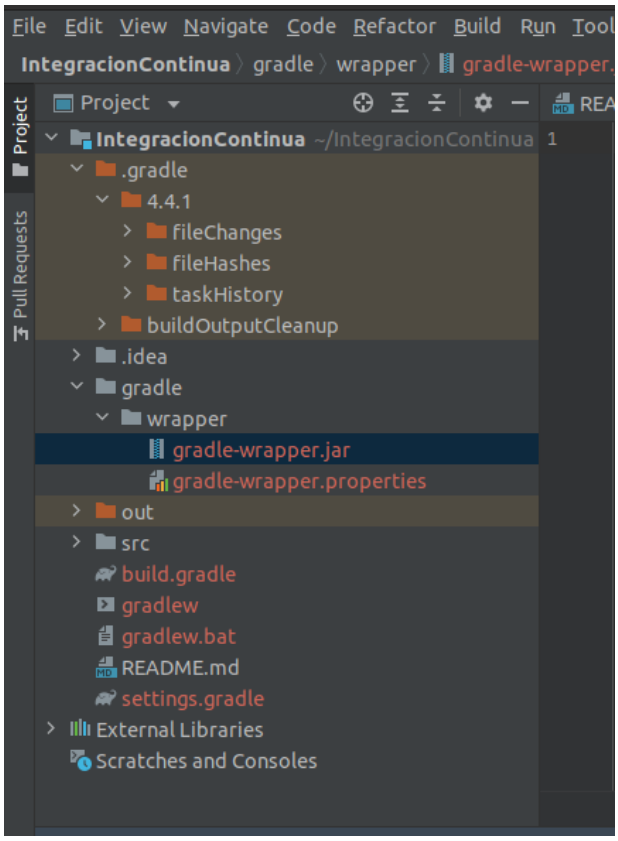
Con el comando *gradle tasks --all* se puede visualizar todas las tareas que el plugin puede realizar para trabajar con java:



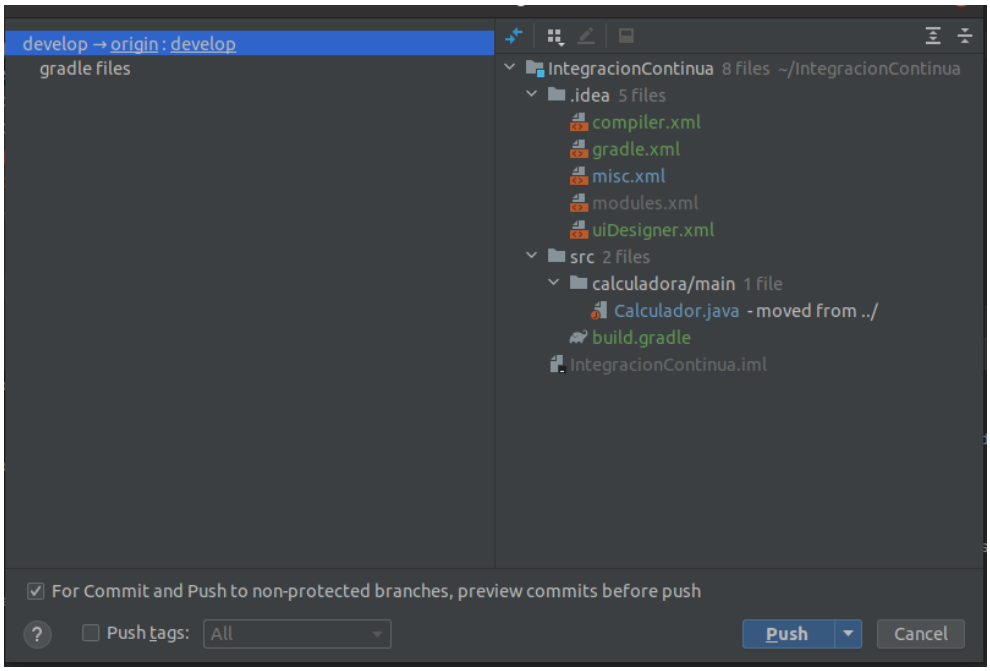
Contenido del fichero build.gradle que se ha creado, con el fin de generar un .jar. Se utilizará el comando *gradle -q build* para compilar proyectos de java:



Una vez ejecutado el comando se puede visualizar como compila el proyecto y genera el .jar:

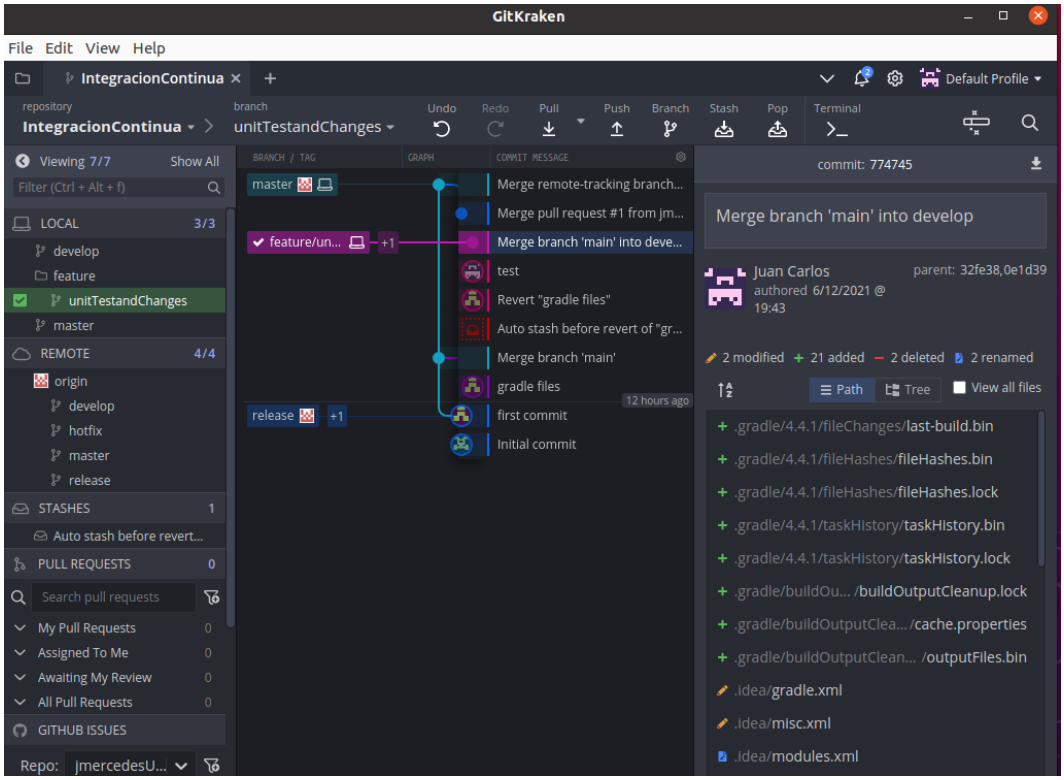


Para finalizar, se hace un *commit*, y un *push* de los cambios para subirlos al repositorio remoto*:*

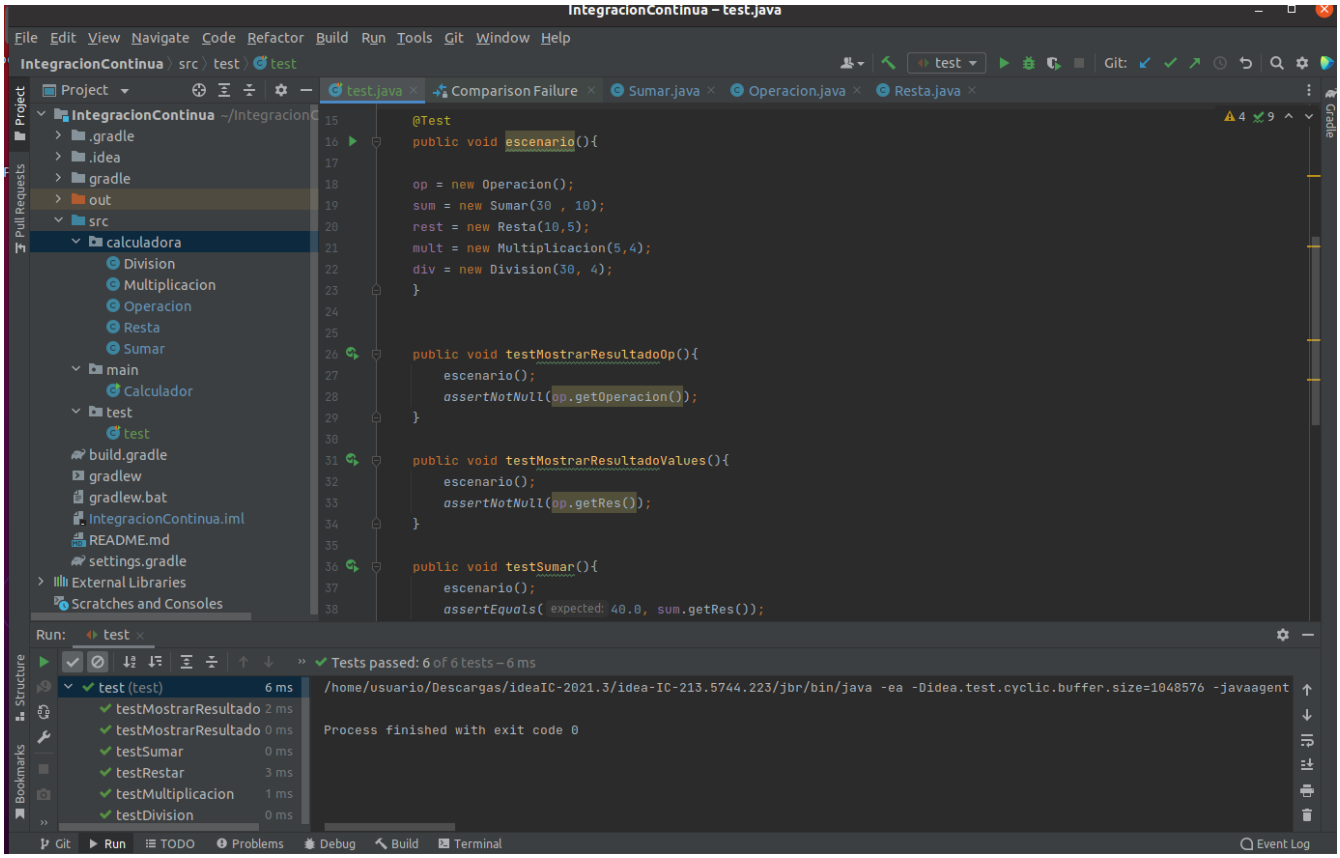


- Pruebas unitarias

A modo de ejemplarizar el flujo git, en GitKraken se ha creado una nueva rama *feature.* En esta se hará algún cambio en el código original y además se crearán pruebas unitarias que posteriormente se ejecutará como un step de Jenkins. A continuación, se muestra la rama:

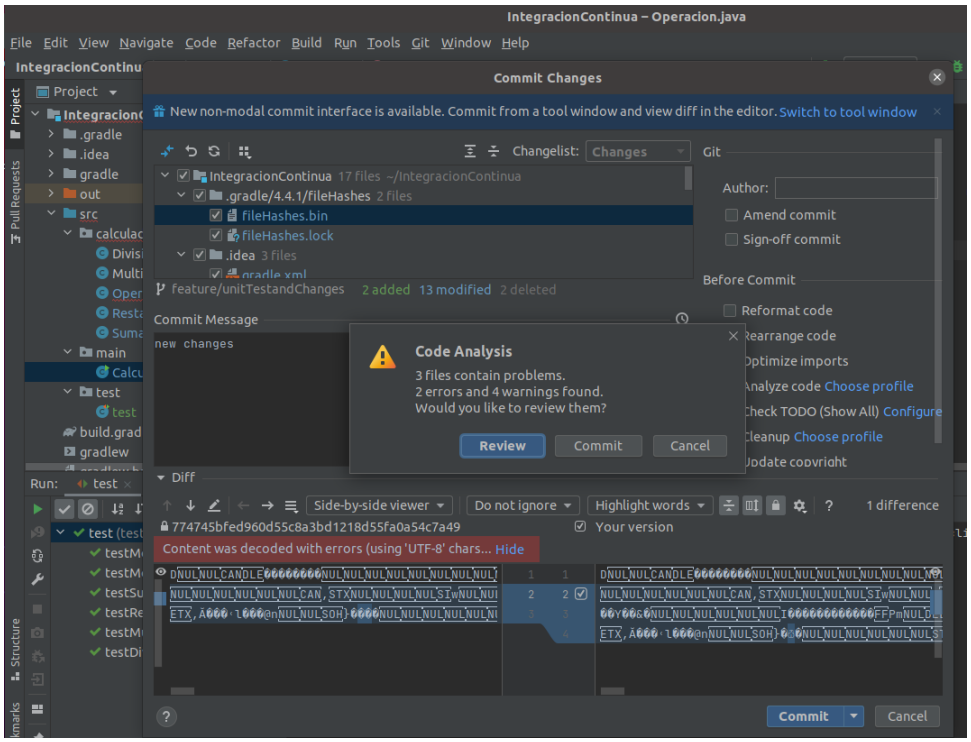


Se procede a crear un test unitario muy simple, que verifica que se devuelve el tipo de operación correcto y que además se devuelve un valor correcto por cada operación pasándole como argumentos diferentes valores:



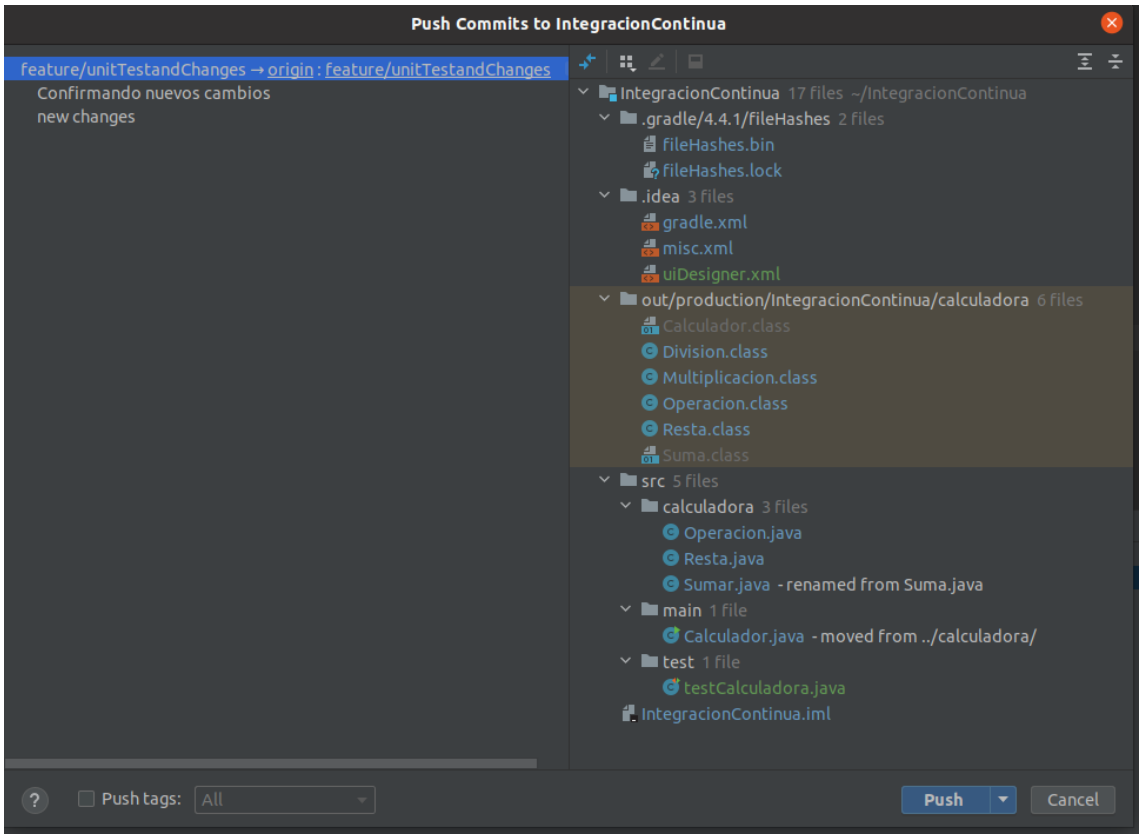
- Commit, push y pull request.

Llegados a este punto, revisadas y configuradas los componentes básicos necesarios para un equipo de desarrollo real, una vez se han realizado los cambios en la *feature*, al igual que como se ha hecho en apartados anteriores, se procederá a confirmar los cambios en el repositorio realizando un *commit.* Para visualizar el gran valor que aporta en cuanto a calidad y optimización del tiempo, primero se simula un código con un error, como se indica a continuación:



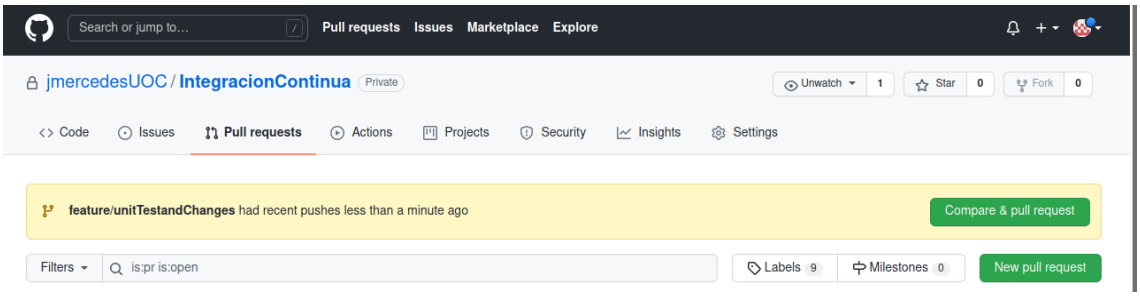
En la imagen anterior, además del mismo IDE, al intentar hacer un *commit* Git indica que existen errores y warnings, y la ubicación exacta de los mismos para poder solucionarlos.

Tras solucionar los errores, se procede a realizar el *commit* y a continuación el *push* hacia el sitio remoto:

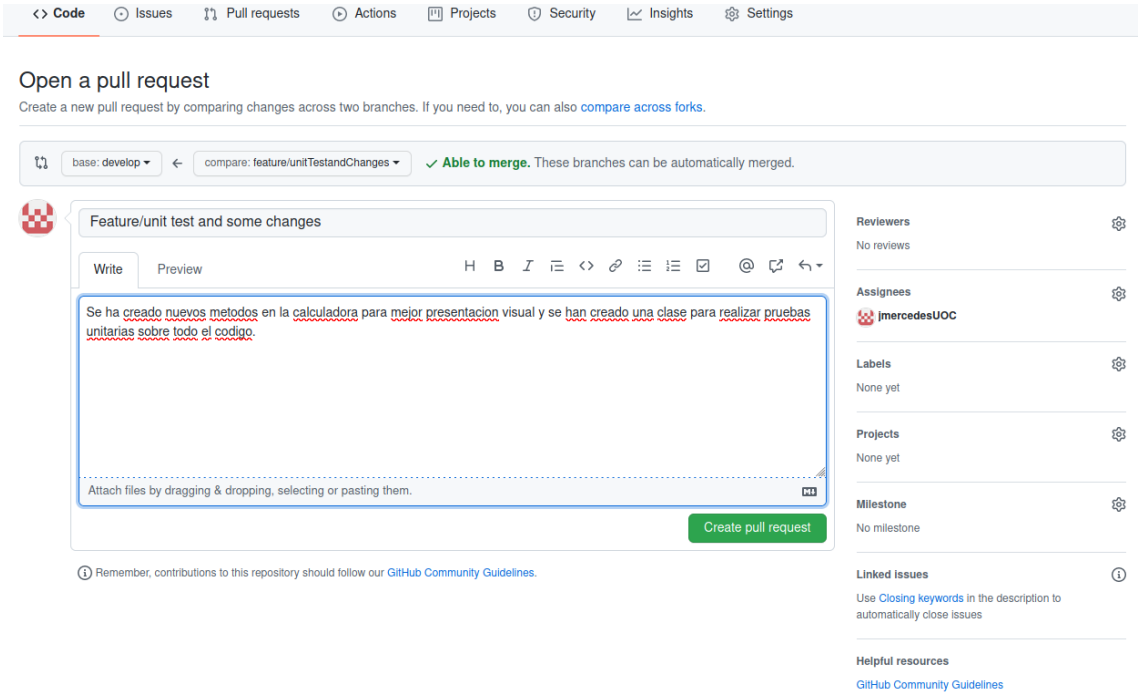


Hasta este momento la rama *feature* ya se encuentra en el repositorio remoto (GitHub) y ésta puede ser descarga por otros desarrolladores, equipos de pruebas o involucrados en el proyecto. Sin embargo, estos cambios o nuevas funcionalidades desarrolladas no están integradas con el resto del código. Un paso no comentado hasta ahora y recomendable (casi obligatoriamente) es realizar una *pull request,* que es la acción de validar un código que se va a *mergear* de una rama a otra. Usualmente, se suele asignar a un responsable técnico que realice esta acción, aunque, adicionalmente, puede ir acompañada de pruebas automáticas. Aunque se puede solicitar desde el mismo IDE o desde GitKraken (en ese caso), la *pull request* se realizará en el mismo GitHub de la siguiente manera:

- Al acceder al repositorio en GitHub, en la opción "*Pull request*" aparece la *feature pusheada*:



- Siguiendo con el GitFlow, explicados en capítulos anteriores, se requiere que los cambios, una vez revisados y aprobados, se mezclen (*merge*) a la rama *develop:*



Cabe mencionar, que tanto los designados como los revisores a la revisión del código, serán notificados mediante email por GitHub. En la siguiente imagen se muestra como GitHub muestra el código que se ha añadido, editado o borrado:

12 src/calculadora/Calculador.java → src/main/Calculador.java

Viewed

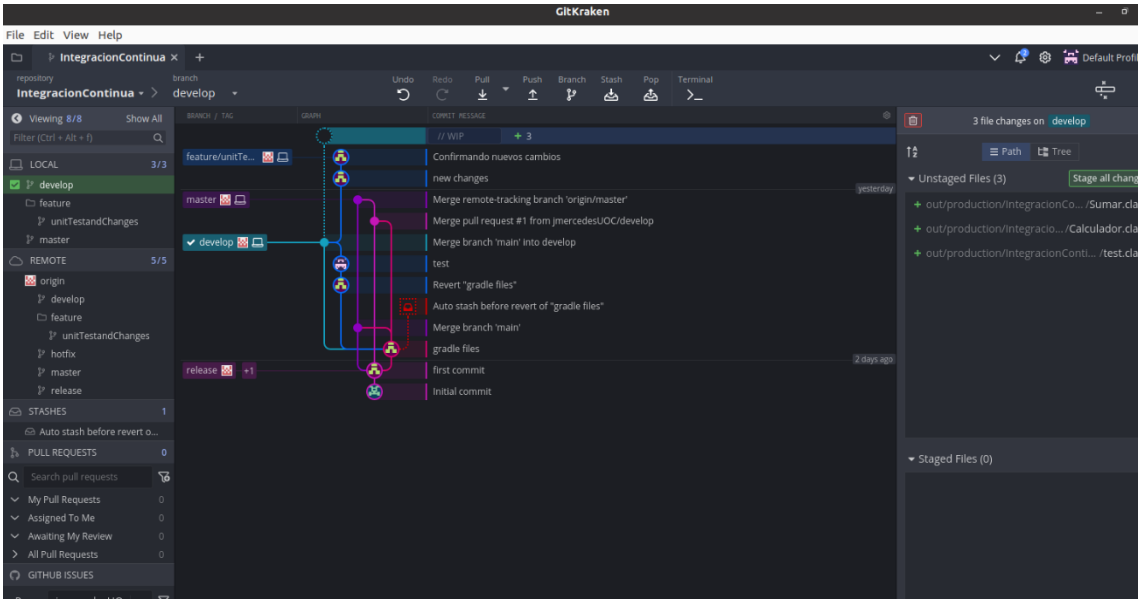
```
21 +            Sumar sum = new Sumar(n1,n2);
17 22           sum.mostrarResultado();
18 23
   24 +
19 25           //resta
20 26           Resta res = new Resta(n1,n2);
21 27           res.mostrarResultado();
```

57 src/test/testCalculadora.java

Viewed

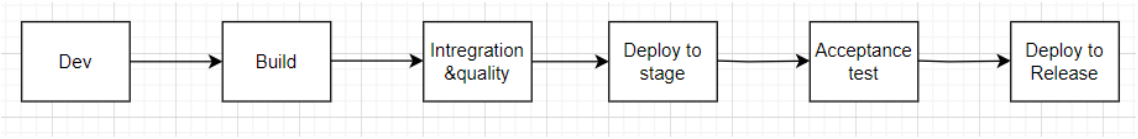
```
... ...  @@ -0,0 +1,57 @@
1  + package test;
2  +
3  + import calculadora.*;
4  + import junit.framework.TestCase;
5  + import org.junit.Test;
6  +
7  + public class testCalculadora extends TestCase {
8  +     private Operacion op;
9  +     private Operacion sum;
10 +     private Operacion rest;
11 +     private Operacion mult;
12 +     private Operacion div;
13 +
14 +
15 +     @Test
16 +     public void escenario(){
17 +
18 +     op = new Operacion();
19 +     sum = new Sumar(30 , 10);
20 +     rest = new Resta(10,5);
21 +     mult = new Multiplicacion(5,4);
22 +     div = new Division(30, 4);
23 +     }
24 +
25 +
```

Si todo es correcto, mediante la opción *Merge pull request*, el asignado (s) puede finalmente mezclar el nuevo código con el que se encuentra en *develop*. Tras realizar esto en GitKraken se puede visualizar de forma gráfica como la rama *develop* está por encima:



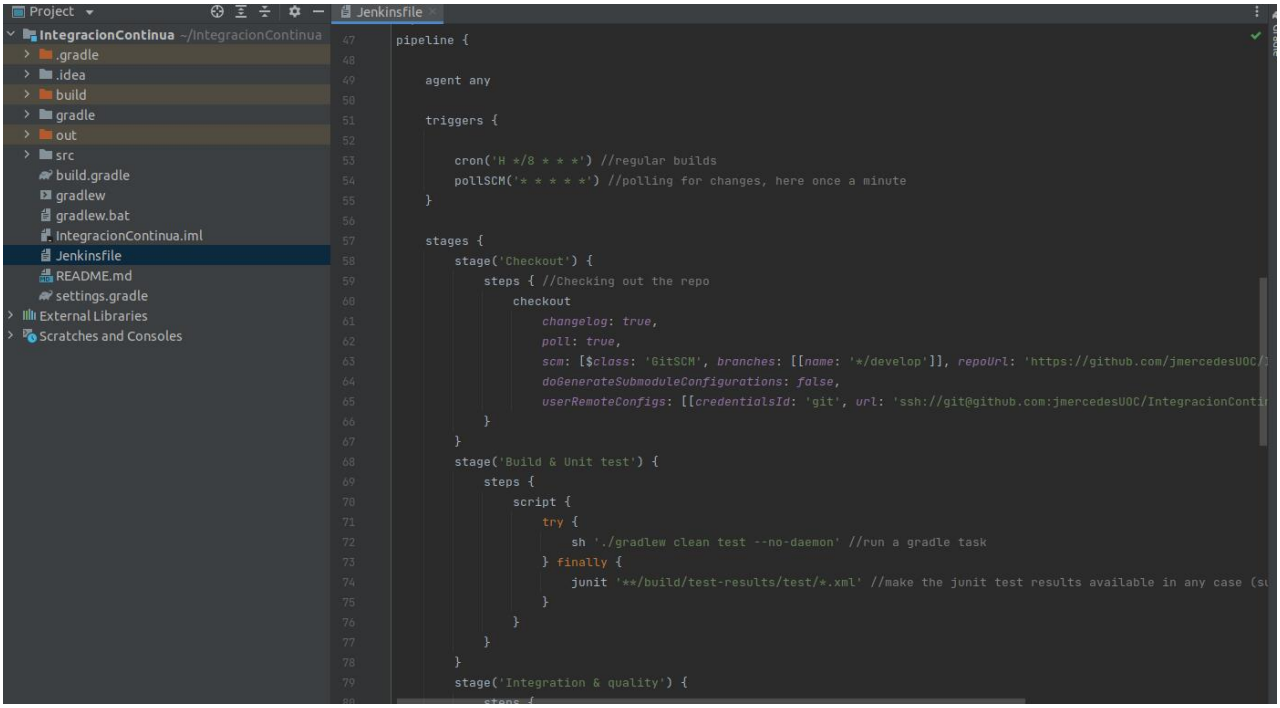
- Creación de pipeline declarativo

Los siguientes pasos para continuar con la integración y despliegue continuo es utilizar la herramienta Jenkins como se ha indicado anteriormente. Se va a seguir el siguiente diagrama:



En los pasos vistos hasta ahora, se ha concluido con la primera fase, la de desarrollo y con la inclusión de herramientas que facilitan el desarrollo de código, su gestión y calidad. Ahora entrará en juego Jenkins que, mediante un pipeline, revisará que se ha mezclado nuevo código a la rama *develop* y cuando detecte que así es, continuará con las siguientes fases mostradas en el diagrama de flujo que se va a seguir.

En este caso, se creará un nuevo fichero (Jenkinsfile) en el repositorio, que será el encargado de realizar las gestiones/tareas necesarias para llevar a cabo la integración y despliegue continuo de forma automatizada, como se ha explicado en el apartado 4.6. Este fichero se puede crear desde Jenkins, pero en este ejemplo, se creará directamente en el proyecto y posteriormente se utilizará en la definición de pipeline de Jenkins. El contenido del pipeline se muestra a continuación:



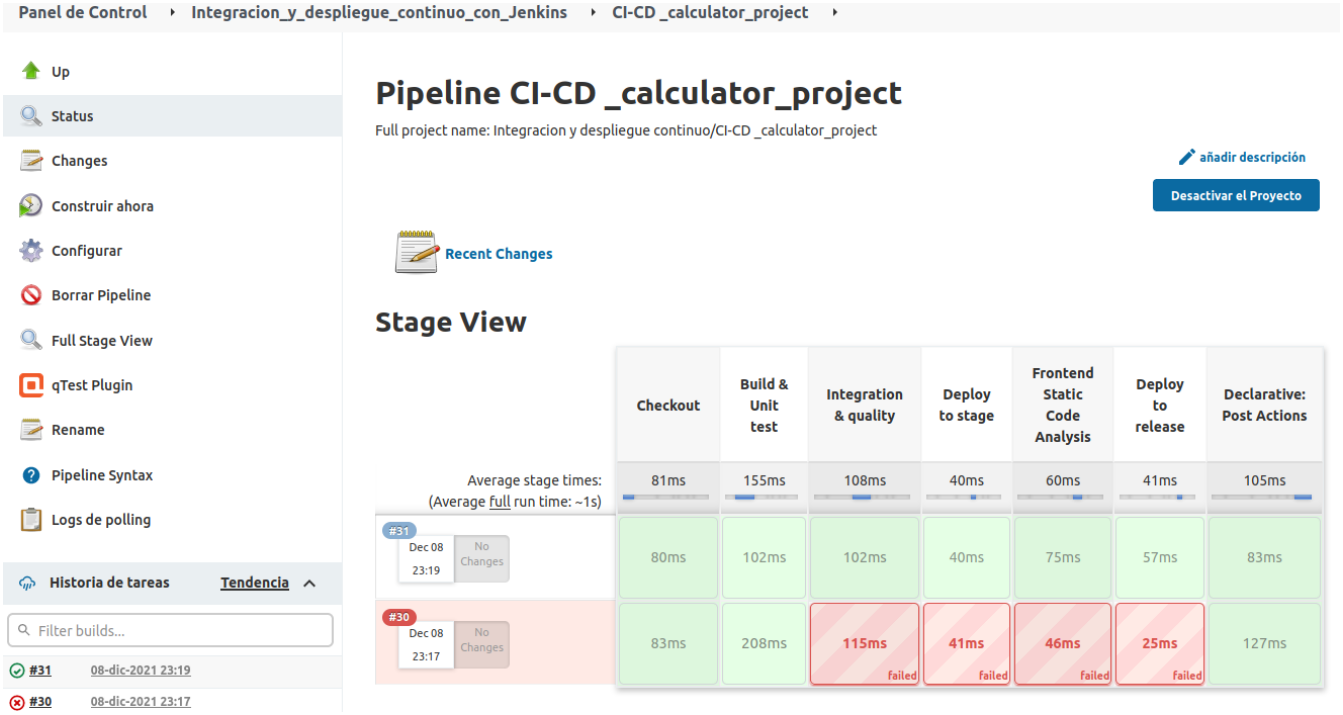
```
        steps {
            sshagent(['git']) {
                script {
                    try {
                        sh './gradlew cleanFrontendTest --no-daemon'
                        sh './gradlew frontendUnitTest --no-daemon'
                    } finally {
                        junit 'publicapi/frontend/test/karma-result.xml'
                    }
                }
            }
        }
    }
      stage('Deploy to stage') {
                steps {
                    sh './gradlew publish --no-daemon'
                    echo 'Build pre-ready to release'
                }
            }

    stage('Frontend Static Code Analysis') {
        steps {
            script {
                try {
                    sh './gradlew tslint --no-daemon'
                } finally { //Make checkstyle results available
                    checkstyle canComputeNew: false, defaultEncoding: '', healthy: '', pattern: 'publicapi/frontend/ts
                }
            }
        }
    }
```

```
    stage('Deploy to release') {
        steps {
            sh './gradlew publish --no-daemon'
        }
    }
}
post {
    always { //Send an email to the person that broke the build
        step([$class                  : 'Mailer',
              notifyEveryUnstableBuild: true,
              recipients              : [emailextrecipients([[$class: 'CulpritsRecipientProvider'], [$class: 'Requeste
    }
}
}
```

Una vez subido al repositorio remoto, se crea el nuevo job de tipo Pipeline en Jenkins y se selecciona el fichero anterior, como se indica en la siguiente imagen:

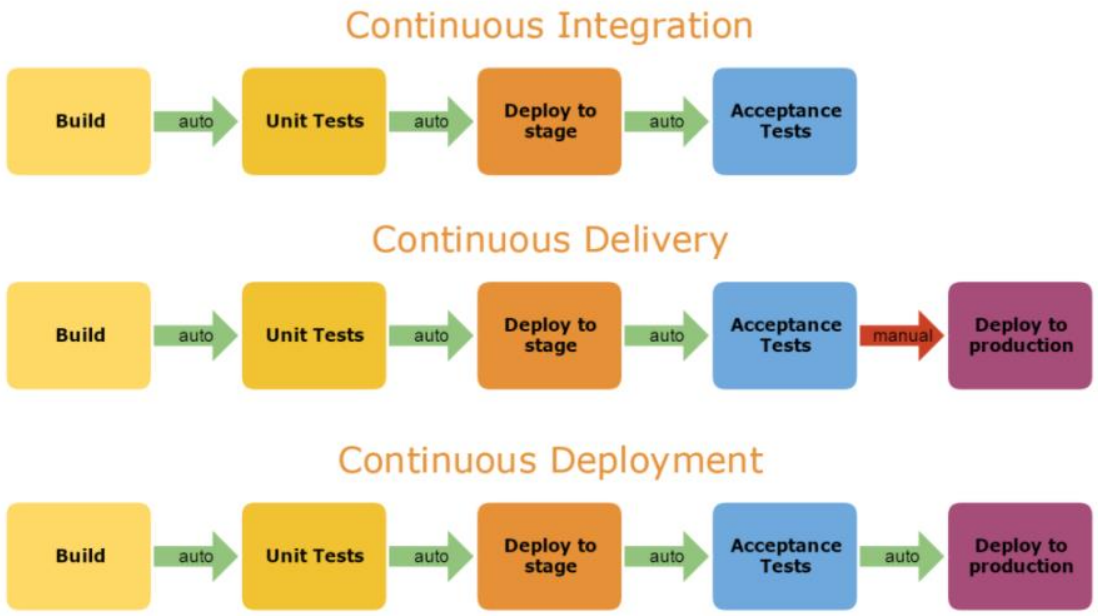
Tras realizar esto seleccionar la opción “Construir ahora” en el job. Finalmente, la vista de estados del job de Jenkins produce la siguiente salida:



Una vez finalizado todo el proceso de integración y despliegue, y para continuar el GitFlow propuesto, como se ha comprobado de forma automatizada, se estaría listo para hacer el *merge* de *develop* a la rama *reléase,* consiguiendo de esta manera convertirse en el código candidato a producción (rama *master*).

CONCLUSIONES Y LÍNEAS FUTURAS

A modo de resumen, y tal como muestra la imagen siguiente, la entrega y el despliegue continuo añaden a la integración continua un paso más, al agregar el paso de despliegue a producción a todo el proceso de desarrollo hasta llegar a producción. La diferencia entre la entrega continua y el despliegue es que para la entrega este paso se realiza manualmente y para la implementación es automático.



La integración continua hace referencia a la combinación de código a menudo, para garantizar que las ramas (código de cada desarrollador) no se desvíen demasiado de su rama principal (al combinarse en la rama principal), y se puede notificar a cada desarrollador, si durante la mezcla se ha producido algún error.

En realidad, no vale la pena profundizar en diferenciarlos. Solo es necesario recordar que la integración y el despliegue continuo son un proceso que suele percibirse como una canalización e implica incorporar un alto nivel de automatización permanente y supervisión constante al desarrollo de las aplicaciones. El significado de los términos varía en cada caso y depende de la cantidad de automatización que se haya incorporado a la canalización de integración y distribución continuas.

Hoy en día, muy pocas organizaciones continúan empleando metodologías clásicas de desarrollo de software porque las siguen considerando válidas para su proyecto, sin embargo, para un desarrollar software seguro de alta calidad será cada vez mas importante impulsar metodologías ágiles para adecuarse a las necesidades actuales del negocio, estando siempre enfocado a mejorar los resultados. En la práctica, en cuando a desarrollo de software se refiere, las metodologías ágiles se han impuesto sobre las clásicas como se indica en el ultimo estudio de Project Manager Institute (PMI), que indica que el 71% de las empresas ingeniería de software utiliza estas metodologías [29]. Por lo tanto, si se quiere dedica profesionalmente al desarrollo de software, se deberá aprender cómo funcionan las metodologías ágiles. La entrega y el despliegue continuos son fundamentales para entregar productos de software de alta calidad más rápido que nunca.

En cuanto al caso práctico se ha demostrado como implementando algunas herramientas y técnicas en el ciclo de desarrollo de software, se puede agilizar, optimizar y, por tanto, conseguir desarrollar un código más robusto y en mucho menos tiempo. Como se ha demostrado, estas herramientas ayudan a compartir información, automatizar procesos, reducir el tiempo de implementación, en implementación continua, etc. Jenkins es la herramienta más utilizada para gestionar construcciones de integración continua y canalizaciones de entrega. Ayuda a los desarrolladores a construir y probar software continuamente. Aumenta la escala de automatización. Tras la integración y despliegue continue se concluye que se ha conseguido el objetivo de facilitar, mediante la automatización de procesos, el ciclo completo de desarrollo de una aplicación desde su fase inicial de desarrollo hasta su puesta en (una posible *release*) producción.

El desarrollo de software se ha convertido en una de las actividades más importantes de la sociedad moderna. En un mundo donde la inteligencia artificial y la tecnología en general constituyen una parte fundamental de nuestras vidas, desarrollar software seguro de alta calidad será cada vez más importante, por lo que la integración y despliegue continua sin ninguna duda seguirá evolucionando.

A modo de mejora, en cuanto a Jenkins se refiere, al ser prácticamente todas las integraciones externas (plugins), requiere mucho tiempo, además de consumo de los recursos de las máquinas, algo que se podría mejorar en Jenkins.

DevOps.

Hasta este momento, aunque se ha mencionado para hacer referencia a otros conceptos, es hora de definir DevOps. DevOps es uno de los términos más mencionados en el actual entorno de IT. Normalmente se asocia a estrategias de transformación digital, y a metodologías como Continuous Delivery o desarrollo ágil. DevOps es un acrónimo inglés de development (desarrollo) y operations (operaciones), que se refiere a una metodología de desarrollo de software que se centra en la comunicación, colaboración e integración entre desarrolladores de software y los profesionales de sistemas en las tecnologías de la información (IT)". DevOps es una respuesta a la interdependencia del desarrollo de software y las operaciones IT. Su objetivo es ayudar a una organización a producir productos y servicios software más rápidamente, de mejor calidad y a un coste menor. Las empresas con entregas (releases) muy frecuentes podrían requerir conocimientos de DevOps.

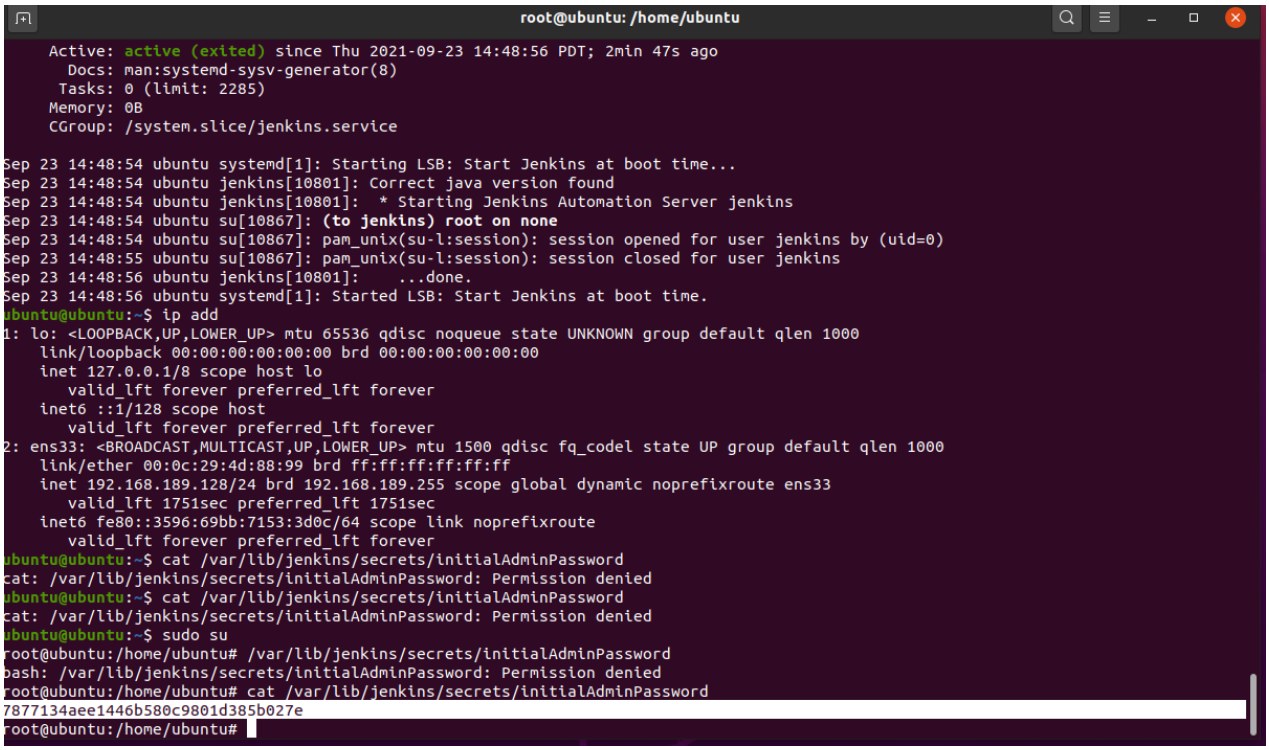
DevOps es una metodología de desarrollo software, y un cambio de cultura no es en sí mismo una forma de desarrollar software.

ANEXOS

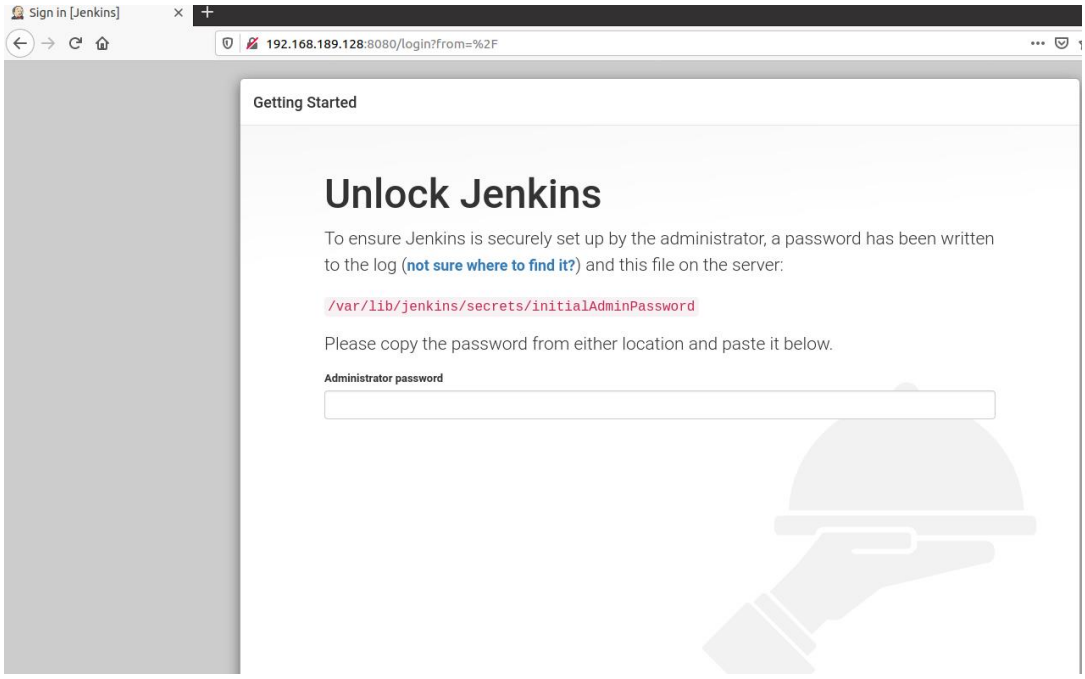
Anexo A. Descarga e instalación de Jenkins.

En este caso, se enumeran los pasos para descargar e instalar Jenkins en un sistema operativo Ubuntu:

- Lo primero, es necesario instalar Java (en este caso Java 8) en el SO:
  sudo apt install openjdk-8-jre
- Descargar e instalar la clave del repositorio Jenkins:
  sudo wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add
- Agregar los binarios a los recursos del sistema:
  sudo sh -c 'echo deb http://pkg.jenkins.iodebian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
- Instalar Jenkins:
  sudo apt-get install Jenkins
- Arrancar Jenkins:
  systemctl start Jenkins
- Verificar el estado y la dirección IP de Jenkins:
  systemctl status Jenkins
  ip add



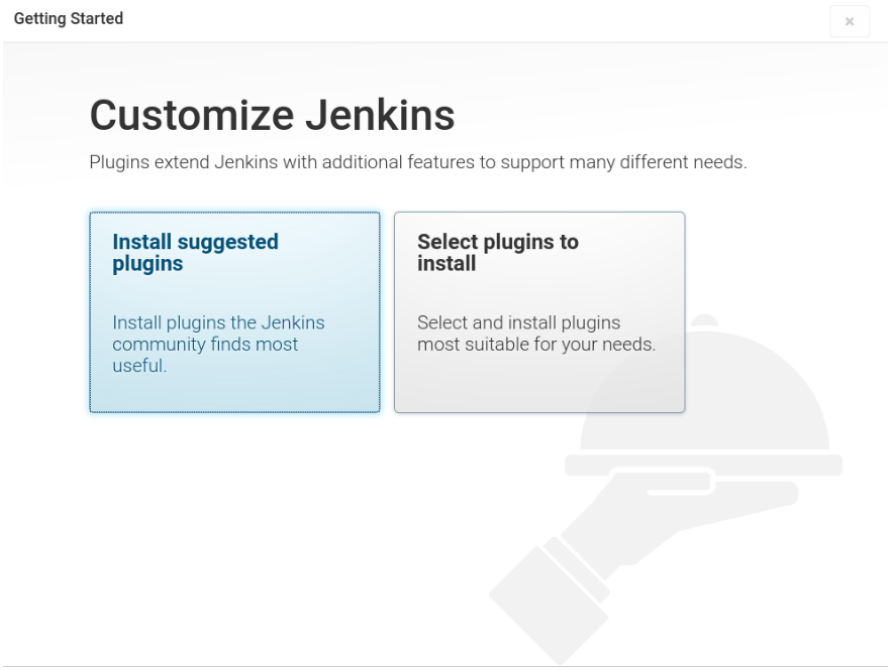
Continuando con el caso de la instalación en Ubuntu, tras la correcta instalación es necesario abrir el navegador e introducir la dirección IP y puerto del servidor Jenkins para acceder a su interfaz gráfica. La IP que observamos en la captura anterior es: 192.168.189.128 y puerto por defecto: 8080. Se muestra la siguiente interfaz gráfica:

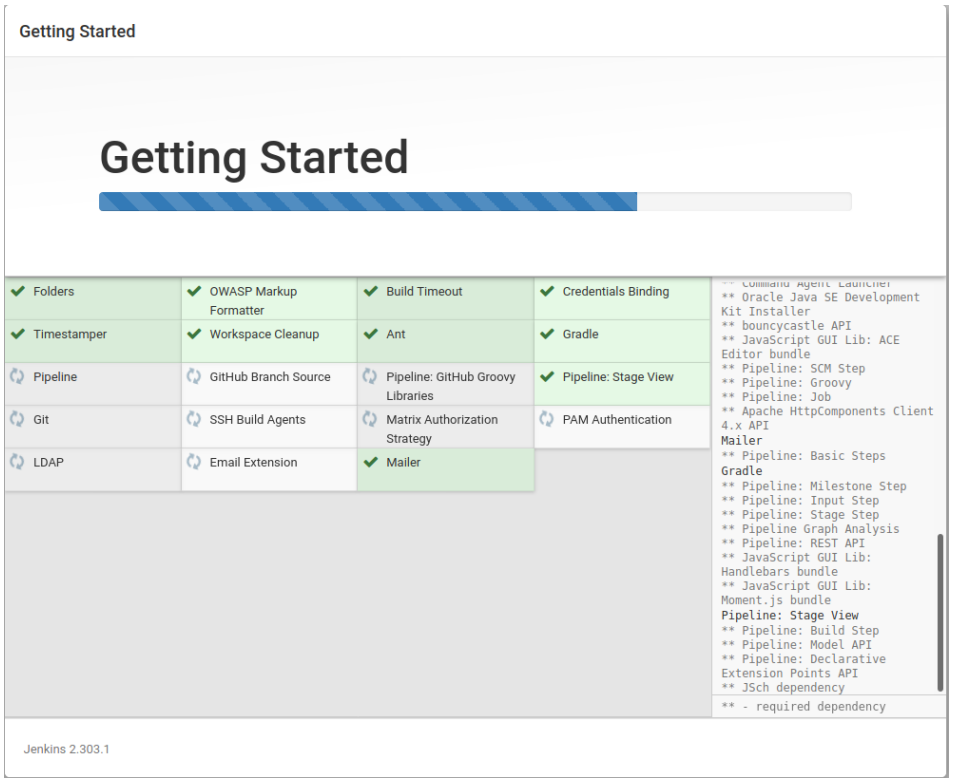


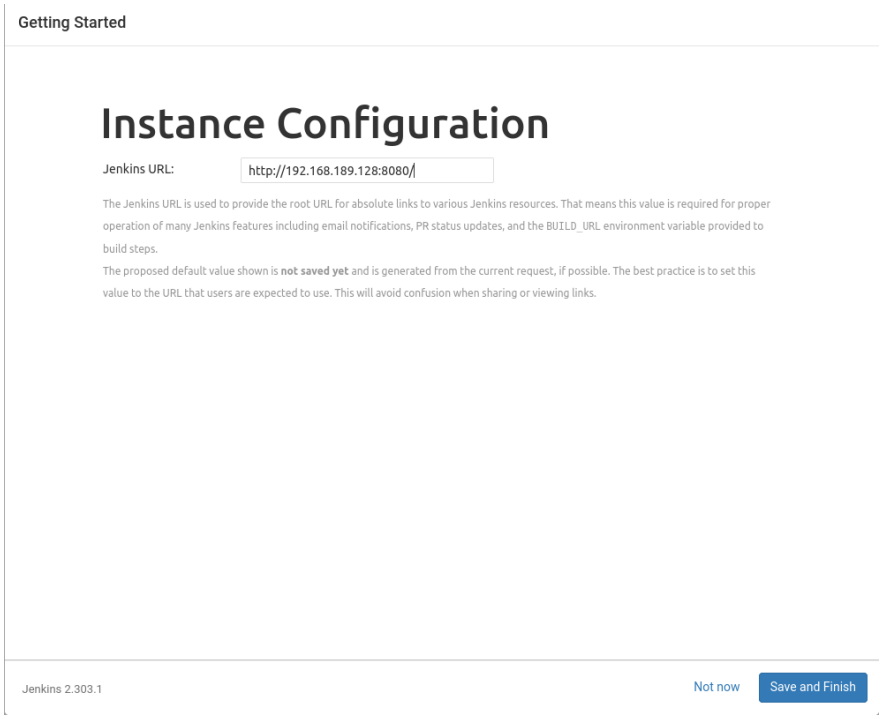
En el apartado de instalación, se muestra cómo se copia la contra contraseña de admin con el siguiente comando:

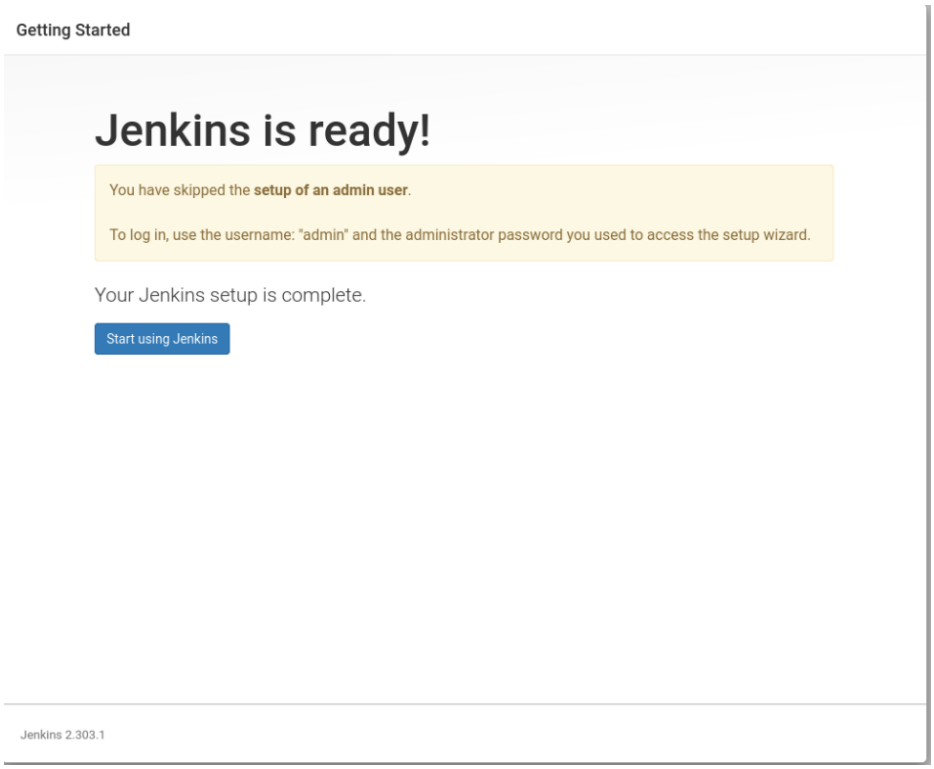
cat /var/lib/jenkins/secrets/initialAdminPassword

Tras introducir la contraseña, se muestran las pantallas de plugins (en este caso se instalarán los sugeridos), a continuación, se muestra la pantalla de creación del usuario administrador y finalmente muestra la pantalla de configuración de instancia, en la que se introduce la url y puerto predeterminado para la instancia de jenkins, como se muestra en las siguientes imágenes:

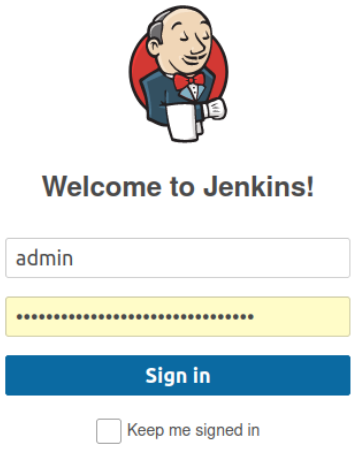



Getting Started
Getting Started
Folders
OWASP Markup Formatter
Build Timeout
Credentials Binding
Timestamper
Workspace Cleanup
Ant
Gradle
Pipeline
GitHub Branch Source
Pipeline: GitHub Groovy Libraries
Pipeline: Stage View
Git
SSH Build Agents
Matrix Authorization Strategy
PAM Authentication
LDAP
Email Extension
Mailer
** Oracle Java SE Development Kit Installer
** bouncycastle API
** JavaScript GUI Lib: ACE Editor bundle
** Pipeline: SCM Step
** Pipeline: Groovy
** Pipeline: Job
** Apache HttpComponents Client 4.x API
Mailer
** Pipeline: Basic Steps
Gradle
** Pipeline: Milestone Step
** Pipeline: Input Step
** Pipeline: Stage Step
** Pipeline Graph Analysis
** Pipeline: REST API
** JavaScript GUI Lib: Handlebars bundle
** JavaScript GUI Lib: Moment.js bundle
Pipeline: Stage View
** Pipeline: Build Step
** Pipeline: Model API
** Pipeline: Declarative Extension Points API
** JSch dependency
** - required dependency
Jenkins 2.303.1

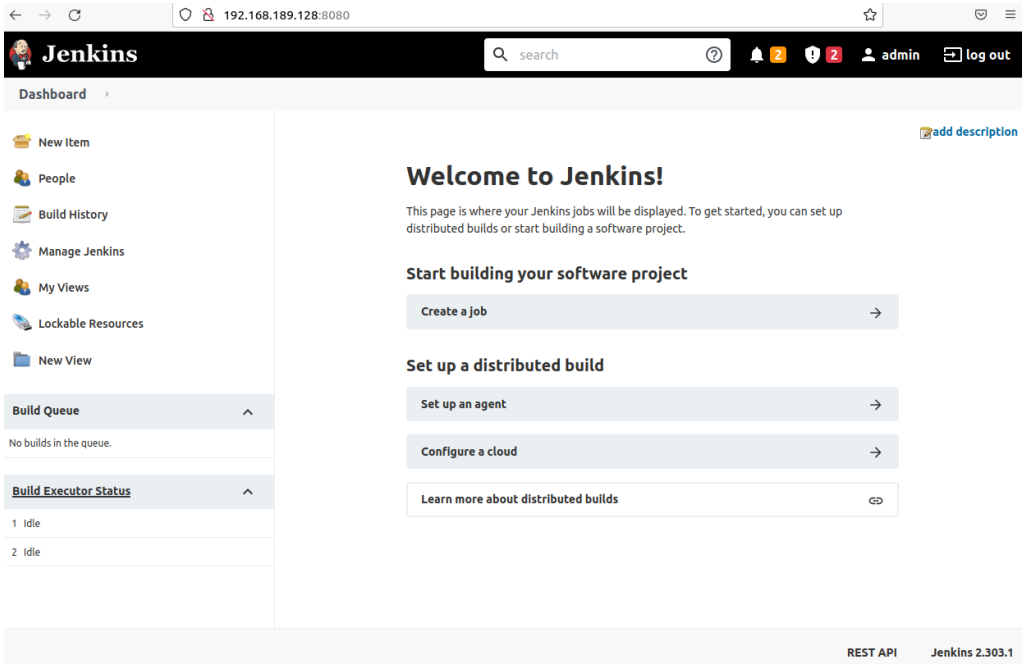

Getting Started
Instance Configuration
Jenkins URL:
http://192.168.189.128:8080/
The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.
The proposed default value shown is not saved yet and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.
Jenkins 2.303.1
Not now
Save and Finish



A partir de este momento, para acceder a Jenkins, será necesario introducir en el navegador la url y puerto configurado. La pantalla inicial es la mostrada a continuación:



Accediendo a Jenkins por primera vez:



Anexo B. Configuración de Git

Es necesario configurar Git de modo que los mensajes de confirmación que genere contengan la información correcta y lo respalden a medida que se realice cambios en su repositorio. Esta configuración es posible mediante el comando *git config*. Específicamente, se debe proporcionar un nombre y una dirección de correo electrónico debido a que Git inserta esa información en cada confirmación (commit) que se realiza:

```
ubuntu@ubuntu:~$ git config --global user.name "Juan Carlos"
ubuntu@ubuntu:~$ git config --global user.email "jmercedes@uoc.edu"
ubuntu@ubuntu:~$
```

La información se guarda en el archivo de configuración de Git. Se puede verificar y editar la información mediante cualquier editor de texto como se muestra a continuación:

```
ubuntu@ubuntu:~$ nano ~/.gitconfig
```

```
  GNU nano 4.8                          /home/ubuntu/.gitconfig
[user]
        name = Juan Carlos
        email = jmercedes@uoc.edu
```

REFERENCIAS

[1] Disponible en: https://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software [Consulta: 03 de octubre 2021]

[2] StudentPlace (septiembre 2018). Metodología de Desarrollo de Software. Disponible en: https://studentplace98.blogspot.com/2018/09/metodologia-de-desarrollo-de-software.html [Consulta: 03 de octubre 2021]

[3] Paradigmas de la Ingenieria de Software (enero 2020). Disponible en: https://web.archive.org/web/20130420132709/http://www.sites.upiicsa.ipn.mx/polilibros/portal/Polilibros/P_proceso/ANALISIS_Y_DISEnO_DE_SISTEMAS/IngenieriaDeSoftware/CIS/UNIDAD%20I/1.5.htm [Consulta: 5 octubre 2021]

[4] PMPP Software. Qué es el método Agile. Definición y características. Disponible en: https://future.inese.es/que-es-el-metodo-agile-definicion-y-caracteristicas/ [Consulta: 7 octubre 2021]

[5] Disponible en: https://www.nextu.com/blog/que-es-scrum/ [Consulta: 8 octubre 2021]

[6] Disponible en https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema [Consulta: 11 octubre 2021].

[7] Javier Garzas (04 septiembre 2012). La metodología ágil FDD. Una metodología ágil para equipos / empresas / proyectos grandes (04 septiembre 2012). Disponible en: https://www.javiergarzas.com/2012/09/metodologia-gil-fdd-1.html [Consulta: 13 octubre 2021].

[8] Disponible en: https://en.wikipedia.org/wiki/Continuous_integration [Consulta: 03 de octubre 2021]

[9] M. Fowler (01 mayo 2006). Integración Continua. Disponible en: https://martinfowler.com/articles/continuousIntegration.html [Consulta: 07 de octubre 2021]

[10] Ilimit (19 mayo 2020). DevOps. Disponible en: https://www.ilimit.com/blog/automatizacion-despliegue-continuo/ [Consulta: 13 de octubre 2021]

[11] IBM Cloud Education (02/10/2019). Despliegue continuo. Disponible en: https://www.ibm.com/es-es/cloud/learn/continuous-deployment [Consulta: 09 de octubre 2021]

[12] RedHat. ¿Qué son la integración/distribuciones continuas (CI/CD)? Disponible en: https://www.redhat.com/es/topics/devops/what-is-ci-cd [Consulta: 11 de octubre 2021]

[13] AWS. ¿Qué es la integración continúa? Disponible en: https://aws.amazon.com/es/devops/continuous-integration/ [Consulta: 15 de octubre 2021]

[14] MAX REHKOPF. Herramientas de integración continua. Disponible en: https://www.atlassian.com/es/continuous-delivery/continuous-integration/tools [Consulta: 15 de octubre 2021]

[15] Disponible en: https://es.wikipedia.org/wiki/Git [Consulta: 15 de octubre 2021]

[16] Disponible en: https://about.gitlab.com/stages-devops-lifecycle/ [Consulta: 17 de octubre 2021]

[17] Disponible en: https://github.com/features [Consulta: 17 de octubre]

[18] The apache Ant Project. Disponible en: https://ant.apache.org/ [Consulta: 17 de octubre 2021]

[19] Gustavo Vázquez. Gradle vs Maven: Definiciones y diferencias principales. Disponible en: https://www.chakray.com/es/gradle-vs-maven-definiciones-diferencias/ [Consulta: 20 de octubre 2021].

[20] Disponible en: https://es.wikipedia.org/wiki/Gradle [Consulta: 20 de octubre 2021]

[21] IONOS. Desarrollo web (19/03/2020). Disponible en: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/herramientas-de-integracion-continua/ [Consulta: 22 de octubre 2021]

[22] Disponible en: https://ciberninjas.com/jenkins/ [Consulta: 27 de octubre 2021

[23] Disponible en: https://www.jenkins.io/doc/ [Consulta: 27 de octubre 2021]

[24] Disponible en: https://en.wikipedia.org/wiki/Jenkins_(software) [Consulta: 30 de octubre 2021]

[25] Disponible en: https://learntutorials.net/es/jenkins/topic/919/empezando-con-jenkins [Consulta: 3 noviembre 2021]

[26] Disponible en: https://www.tutorialspoint.com/jenkins/index.htm [Consulta: 7 noviembre 2021]

[27] Disponible en: https://www.jenkins.io/doc/book/pipeline/ [Consulta: 10 noviembre 2021]

[28] Álvaro García. Integración continua de Softtware: GIT y GIT FLOW. Disponible en: https://castor.com.co/integracion-continua-de-software-git-y-git-flow/ [Consulta: 15 noviembre 2021]

[29] Transforming the high cost of low performance. Disponible en: https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2017.pdf [Consulta: 09 diciembre 2021]