DEFINICIÓN DE INTEGRACIÓN Y ENTREGA CONTINUA PARA EL DESARROLLO DE APLICACIONES MÓVILES MULTIPLATAFORMA

MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN INGENIERÍA DEL SOFTWARE Y SISTEMAS INFORMÁTICOS

Ismael Calvo Villalvilla

TRABAJO FINAL DE MASTER

ITINERARIO: Ingeniería del Software

CODIGO ASIGNATURA: 31105151

TIPOLOGIA: Tipo B

TITULO: DEFINICIÓN DE INTEGRACIÓN Y ENTREGA CONTINUA PARA EL DESARROLLO DE APLICACIONES MÓVILES MULTIPLATAFORMA

NOMBRE: Ismael Calvo Villalvilla

DIRECTOR: Ismael Abad Cardiel

CURSO ACADÉMICO: 2019/2020

CONVOCATORIA: Septiembre

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE MASTER

Fecha: 14/09/2020

Quién suscribe:

Autor: Ismael Calvo Villavilla

D.N.I: 51689141E

Hace constar que es el autor del trabajo:

TRABAJO FINAL DE MÁSTER:

DEFINICIÓN DE INTEGRACIÓN Y ENTREGA CONTINUA PARA EL DESARROLLO DE APLICACIONES MÓVILES MULTIPLATAFORMA

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Firmado:

Autorización de publicación y difusión del TFM para fines académicos

Autorización

Autorizo a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del Autor

1 Resumen

Una de las tareas más importantes en el ciclo de vida de un producto software es realizar el correcto proceso de integración de código fuente entre los miembros de un equipo de trabajo y disponer de un repositorio centralizado y único, además también debemos asegurar que nuestro producto se construye correctamente y cumple con los estándares de calidad, una vez integrado el código. Es importante que estas tareas de integración y validación del producto se apliquen desde el comienzo del desarrollo y se realicen lo más frecuentemente posible. Siguiendo esta línea, en los últimos tiempos se están imponiendo metodologías agiles, basadas en la realización de entregables funcionales en cortos periodos de tiempo, este paradigma se asocia perfectamente con la automatización de las tareas de integración de código, la realización de pruebas y la construcción de entregables.

En este contexto aparecen conceptos como Integración Continua, Entrega Continua y Despliegue Continuo, en el presente documento intentare definir brevemente en que consiste y describir un modelo de arquitectura de Entrega Continua para el desarrollo de aplicaciones móviles.

2 Palabras Clave

Integración Continua (CI), Entrega Continua (CD), Despliegue Continuo (CD), Repositorio de versiones, Aseguramiento de la calidad (QA), Test unitarios, test funciones, pruebas de regresión, pruebas de carga, generación de versiones debug, beta y release, Metodologías Agiles, Aplicaciones móviles híbridas, Apache Cordova, IOS, Android, Ionic framework. Jenkins, Jobs, pipeline, JenkinsFile, build.

Contenido

Índice de Figuras

3 Introducción

El presente documento pretende definir un marco de trabajo eficaz para el desarrollo de software en entornos colaborativos, el modelo va a facilitar las tareas de integración del código fuente y la definición y realización de acciones de manera automática que permiten asegurar la calidad del producto. El modelo definido está basado en una arquitectura de Integración y Entrega Continua (CI/CD por sus siglas en ingles), como ejemplo de aplicación se ha seleccionado el desarrollo de aplicaciones móviles hibridas.

El motivo de implementar este tipo de modelo sobre el desarrollo de aplicaciones móviles es porque, a mi parecer, se trata de un entorno donde puede facilitar en gran medida las tareas que conlleva, cabe recordar que el desarrollo de este tipo de soluciones esta intrínsecamente relacionado con la plataforma destino (IOS, Android) teniendo cada una sus propias herramientas de desarrollo, lo que a veces crea la necesidad de disponer de equipos multidisciplinares. Además, sobre todo con Apple, las licencias de desarrollador deben estar instaladas en el equipo donde se generan los entregables por lo que es de gran importancia disponer de un entorno centralizado donde se puedan realizar de forma automática la construcción de dichos entregables release[1].

A pesar de que hemos centrado el modelo al desarrollo de aplicaciones híbridas, cabe decir que este tipo de modelos es aplicable a cualquier ciclo de vida del software.

A la hora de abordar el trabajo nos encontramos con dos arquitecturas con fines bien distintos, por una parte, los conceptos y herramientas de CI/CD que nos sirven para introducir mejoras en nuestros procesos de desarrollo software, y por otra nos encontramos con el software en sí que hemos desarrollado, en este caso, una aplicación para dispositivos móviles multiplataforma.

[1] Versión estable de un producto liberada para el uso de clientes finales

4 Objetivos

El objetivo principal de este trabajo es definir un modelo de Integración y Entrega Continúa enfocado al desarrollo de aplicaciones móviles. Con este modelo CI/CD se pretenden automatizar en todo lo posible las tareas dentro del ciclo de vida del desarrollo y mantenimiento de un producto software con el fin de minimizar los posibles errores en la integración de nuestro código que pueden acarrear problemas graves, tanto a la hora de generar entregables como en el uso funcional final de nuestros productos. Al hilo de este objetivo, se ha evaluado un modelo básico de trabajo sobre un repositorio de versiones de código, punto fundamental para el éxito de nuestro desarrollo software.

Dentro de nuestro modelo CI/CD, se plantea también el objetivo de realizar tareas que mejoren la calidad de nuestros productos, como son la ejecución de test (unitarios y de integración) y la revisión de código de estático, de esta manera impedimos entregar nuestros productos con bugs de programación o problemas de seguridad.

Como objetivo último, también se ha tratado de evaluar las ventajas de los modelos CI/CD en el desarrollo de aplicaciones móviles automatizando tareas mecánicas y sacándolas del día a día del desarrollo.

5 Alcance y restricciones

Se desarrolla un modelo de CI/CD para proyectos de desarrollo de aplicaciones móviles híbridas, basado en herramientas Open Source[2]. Las acciones que debe realizar son:

1. Detección de cambios en el repositorio de versiones que iniciaran el ciclo de CI/CD. Cuando un programador sube código a una rama determinada de nuestro repositorio, el sistema debe ser capaz de detectar los cambios y comenzar el ciclo de CI/CD.
2. Gestión de dependencias del proyecto. Por las características de las aplicaciones móviles, es de suma importancia disponer de las dependencias necesarias para la construcción de nuestro producto con sus versiones correspondientes.
3. Control de calidad de producto, ejecución de test y revisión de código, con el fin de controlar la calidad de nuestros programas. El modelo debe controlar que cuando no se cumpla con los criterios de calidad establecidos, no se realizará la creación de entregables o artefactos[3], marcando como errónea la integración. Es importante comunicar al equipo de desarrollo estos incumplimientos con el fin de solucionar los problemas lo antes posible.
4. Construcción de los entregables para la instalación de las aplicaciones móviles tanto en IOS como en Android y webApp[4].
5. Reporte de resultado. En caso de problemas en el ciclo, creación de ticket para solucionar dichos problemas.

Ha quedado fuera del alance de este trabajo la realización de pruebas funcionales sobre móviles, si bien los de test con Jasmine[5] podrían definirse como pruebas funcionales, estas solo se realizan sobre navegador.

[2] Licencia de código abierto. Mas información en https://opensource.org/
[3] Empaquetado o compilado de una aplicación software para la instalación o su uso directo
[4] Página web adaptada a dispositivos móviles
[5] Framework de pruebas unitarias para JavaScript. Mas información https://jasmine.github.io/

6 Estado del arte

6.1 ¿Qué entendemos por CI/CD?

6.1.1 Integración Continua

Martin Fowler[6] define la Integración Continua con "*una práctica de desarrollo de software en la que los miembros de un equipo integran su trabajo con frecuencia, por lo general cada persona se integra al menos diariamente, lo que da lugar a múltiples integraciones por día. Cada integración se verifica mediante una construcción automatizada (incluida la prueba) para detectar los errores de integración lo más rápidamente posible.*"

La mayoría de los autores asocian las prácticas de CI con metodologías agiles, aunque nada impide que se generalicen dichas prácticas a otras formas de desarrollar software.

Las acciones principales que incluye la práctica CI son:

- Repositorio único y centralizado de código.
- Construcción automática de la aplicación
- Ejecución Automática de pruebas.
- Revisión de calidad y seguridad de código, también automática.
- Publicación de Resultados

A continuación, se detalla cada una de las acciones desde un marco teórico

6.1.1.1 Repositorio único y centralizado de código

Todo modelo de CI tiene como punto de partida disponer de un único repositorio de código centralizado, sobre el que trabaja el equipo de desarrolladores y donde se encuentra el código actualizado. Este punto es importante, no hay CI si los miembros

[6] Definición del blog de Martin Fowler https://martinfowler.com/articles/continuousIntegration.html

del equipo no suben con frecuencia sus cambios y no los integra con los del resto del equipo. Es necesario, por tanto, definir una estrategia de gestión de versiones que nos permita integrar el código con frecuencia (creación de ramas, branch, tag) y con la cual el equipo se sienta cómodo trabajando. Gran parte del éxito en la implantación de CI en nuestros procesos de desarrollo de software es la definición de una estrategia de trabajo sobre el repositorio de código, normalizar como se integran los sucesivos desarrollos y correctivos del equipo y cuando y como se generan las release de entrega. Debe existir una figura de "responsable o gestor" que defina la forma de trabajar y decida como y cuando se realizan las integraciones o "merges" de código y promoción entre versiones.

Al ser el proceso donde más está implicada la acción humana es, a mi modo de ver, el que más riesgos corre de no realizarse correctamente.

6.1.1.2 Construcción Automática

Además de definir la estrategia de integración de nuestro código debemos realizar las acciones necesarias que nos aseguren que nuestro código está bien integrado y que, además, cumple con los requisitos de usuario, de sistema y de seguridad. La implantación de CI en nuestros sistemas nos facilita las acciones de compilación, revisión y ejecución de pruebas que se realizaban de forma manual por el equipo y que ahora serán automáticas y sobre un entorno estable.

La primera tarea que debemos realizar es generar los entregables de nuestro proyecto, comprobando que dicha generación es correcta. Es recomendable realizar esta tarea de forma automática e incorporar herramientas que nos faciliten la gestión de dependencias de nuestro proyecto.

6.1.1.3 Ejecución Automática de pruebas

Una parte importante del proceso de CI es realizar pruebas que nos indique si nuestro desarrollo cumple con los requisitos funciones y de sistema establecidos previamente. Cada vez es más frecuente encontrarnos framework que nos permiten automatizar tanto los tests unitarios como las pruebas funcionales, de sistema y de aceptación. La

ejecución de pruebas en los procesos de CI nos permite aflorar en fases tempranas del desarrollo bugs inesperados en nuestro código y cualquier desajuste entre los requisitos y la implementación del programa. Algunos autores, como Kent Beck[7], defienden la metodología de "Diseño dirigido por pruebas, TDD por sus siglas en inglés (Test-Driven Development), en líneas generales plantean que, a partir de los requisitos, se diseñan e implementan las pruebas que certifiquen dichos requisitos, el siguiente paso es implementar el código, el código no estará correcto hasta que no pase las pruebas anteriormente diseñadas.

6.1.1.4 Revisión de calidad y seguridad de código

La revisión automática de código nos permite conocer el estado de la calidad de nuestro desarrollo, descubrir y anticipar errores latentes de ejecución y problemas de seguridad de nuestro código. Las características principales sobre las que se mide la calidad del software son:

- Confiabilidad: Que nuestro sistema hace lo que tiene que hacer y lo hace siempre de la misma forma. No se producen comportamientos inesperados y presenta una gran resiliencia sobre fallos no esperados.
- Rendimiento**:** El sistema hace un uso eficiente de los recursos disponibles,
- Seguridad**:** El sistema debe asegurar la privacidad e integridad de la información, también debe asegurar el buen funcionamiento ante ataques maliciosos.
- Mantenibilidad**:** Mide el grado de dificultad y coste que conlleva acometer cambios en nuestro sistema. A mayor complejidad de nuestro código mayor esfuerzo se debe realizar al abordar cambios en los requisitos o correcciones de errores.

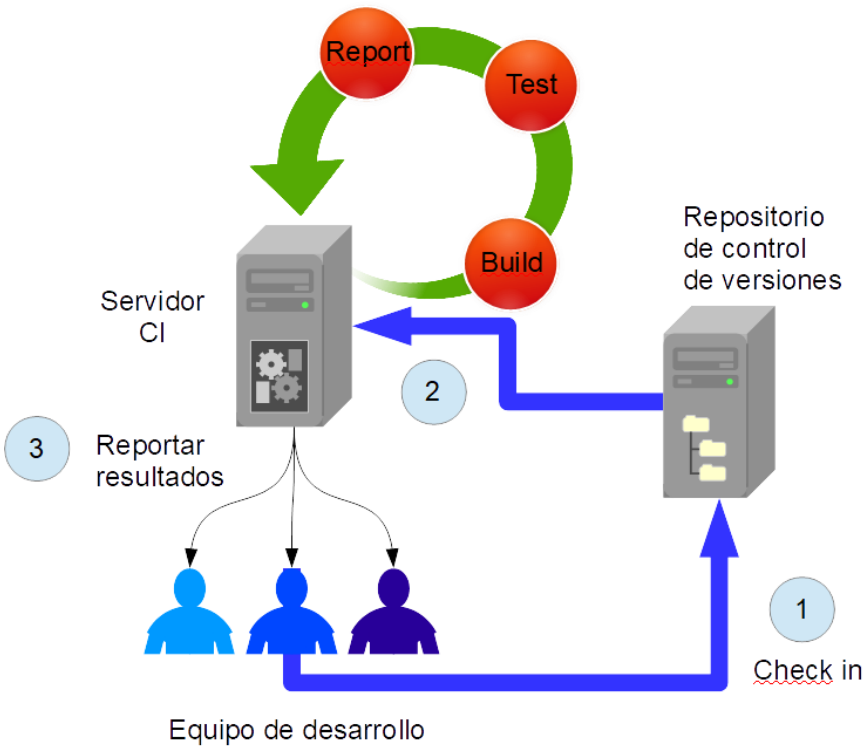
[7] Ken Beck - Test Driven Development: By Example Ed. Addison-Wesley 2002

6.1.1.5 Publicación de Resultados.

En cada ejecución de CI se debe reportar al equipo de desarrollo el resultado de los diferentes procesos anteriormente descritos, es de suma importancia ser conscientes del estado de nuestro desarrollo y acometer lo antes posible los bug o problemas detectados en el proceso. El informe de cada ejecución de CI debe contener:

- El resultado de la construcción de entregable.
- Los resultados de las pruebas ejecutadas.
- El resultado de la revisión de código estático.

Es recomendable que, en caso de fallos, se integre el proceso con algún sistema de seguimientos de incidencias que cree tickets de trabajo para el equipo, donde se registren dichos problemas.



1 Integración Continua

6.1.1.6 Ventajas e inconvenientes de realizar CI.

A lo largo de los anteriores apartados hemos ido desgranado las ventajas de incluir CI en nuestros procesos de desarrollo software, a modo de resumen cabe destacar:

- Reducción de tareas manuales. Muchas tareas repetitivas, que se realizan de forma manual y aportan poco valor al equipo, pasan a automatizarse. Automatizar las tareas minimiza errores porque se realizan de la misma forma y en las mismas condiciones.
- Al realizarse comprobaciones periódicas continuamente hay una detección temprana de problemas y errores en nuestros desarrollos.
- Las integraciones de código continuas agilizan el proceso de generar entregas.
- Tenemos una visión más real del estado de nuestro proyecto durante el desarrollo, evitando sorpresas en el momento de la entrega final.

Pero antes de plantearnos la implantación de CI deberíamos sopesar los inconvenientes habituales que nos podemos encontrar y si, dependiendo de las características de nuestro sistema, realmente nos va a aportar mejoras en nuestros procesos.

Los principales inconvenientes son:

- Puede ser un cambio cultural importante, de difícil superación en equipos donde la integración de código se realiza en fases muy avanzadas del proyecto y se trabaje de forma atómica por sus componentes. También hay un choque cultural con equipos donde no se prioriza la realización y ejecución del plan pruebas.
- La propia CI pasa a ser un proyecto en sí, conlleva esfuerzo y coste asociado y hay que dedicar recursos específicos, no solo para la definición e implantación sino también de hardware dedicado. También hay que disponer de recursos que se encarguen del aseguramiento de la calidad. Por

tanto, es posible que, para pequeños proyectos donde el equipo de trabajo es muy reducido, nos aporte más inconvenientes que ventajas.

- Uno de los factores de éxito de la CI es la realización exhaustiva de pruebas, por ellos, es importante la creación de un plan de pruebas que se ajuste a los requisitos y una ejecución correcta del mismo. Cualquier cambio en el proyecto implica cambios en el plan de pruebas y la ejecución de este, el problema es que determinados equipos no ven este esfuerzo tan necesario y no se aplican con el rigor requerido. Si el plan de pruebas y su ejecución durante la CI es incompleto o no se ajusta a los requisitos estaremos dando un resultado de "falsa visión del estado" de muestro proyecto.

6.1.2 Entrega Continua

Enlazando con los procesos de CI y a continuación de estos, los procesos de Entrega Continua (CD) consisten en realizar las acciones necesarias para dejar nuestro software listo para su instalación en los entornos productivos. Jez Humble y David Farley en su libro[8] describen la CD "*como un complemento o extensión de la Integración Continua, donde en cada ejecución se generan los artefactos necesarios para la implantación en producción de nuestro sistema*", es decir, con este proceso nos aseguramos de que en cuanto nuestro sistema esté preparado funcionalmente para su despliegue e implantación tengamos los artefactos necesarios integrados y generados correctamente. Las implantaciones se realizarán fuera de los procesos de CD.

CD nos permite ser más agiles en las puestas en producción, manteniendo un alto grado de eficacia y reduciendo riesgos inesperados al tener controlado y automatizado el proceso de generación de entregas y realizarse en periodos de tiempo relativamente cortos. La clave está en el continuo trabajo de mejora e integración de nuestro código, asociado a la ejecución continua de pruebas. Todos estos trabajos facilitan que el proceso de generar las entregas finales sea menos traumático.

[8] Continuous Delivery - Reliable Software Releases Through Build, Test And Deployment Automation Jez Humble and David Farley Upper Ed. Addison-Wesley. 2010

En ocasiones se define CI como un todo que incluye tanto los procesos de Integración como de CD, aunque lo más común es referirse a dichos procesos por sus siglas en ingles CI/CD, *Continuous Integration/Continuous Delivery*.

6.1.3 Despliegue Continuo

Con Despliegue Continuo vamos un paso más allá en los procesos de CI/CD al incluir el despliegue automático de los productos en los entornos productivos, en cada ciclo de los procesos CI/CD.

Muchos de los procesos de despliegue incluyen tareas ajenas al software, configuración de entornos, definición sistemas y operación sobre los mismos, …. Implementar un proceso de despliegue dentro de CI/CD conllevan la participación de equipos multidisciplinares, lo que en metodológicas agiles[9] se ha denominado DEVOPS[10]. Además, hay que añadir tareas sobre los despliegues de monitorización y operación sobre los sistemas.

2 Despliegue continuo

El ciclo CI/CD comienza con la planificación, el equipo junto con el cliente decide que mejoras serán incluidas en nuestro sistema en la iteración, el resultado de esta tarea

[9] Manifiesto por el Desarrollo Ágil del Software https://agilemanifesto.org/iso/es/manifesto.html
[10] Acrónimo del inglés development y operations, desarrollo y operación, metología basada en un entorno colaborativo entre el equipo de desarrollo y el de sistemas.

servirá como documento de trabajo para la fase de desarrollo. Una vez concluido el desarrollo, se procederá a las tareas automatizadas de construcción, ejecución de pruebas (unitarias, integración y funcionales) y revisiones de calidad, solo si el resultado de lo anterior se ajusta a los umbrales de calidad exigidos se procede a genera la release de nuestro producto y pasar a realizar el despliegue en los entornos productivos. CI/CD deberá realizar tareas de monitorización, generando los informes precisos que se hayan planificado en la primera fase. Con la información recolectada durante las fases de CI/CD junto con la experiencia de ejecución del usuario se volverá a la fase de planificación para comenzar un nuevo ciclo.

Independientemente de la aplicación de Despliegue Continuo, creo que es muy importante que en el proceso de definición de un sistema se involucre también al equipo de operación y que acompañe en el proceso de desarrollo, de esta manera evitamos los habituales contratiempos a la hora de la implantación de que no hayamos definido una arquitectura de Hardware y comunicaciones coherente y óptima para nuestro sistema.

Hay que tener en cuenta tanto las características de nuestro sistema, si es posible automatizar los procesos de despliegue, como de nuestra organización, si es conveniente que todos los cambios o mejoras incluidas en cada ciclo de CI/CD deban incluirse en un producto final. Además, hay que tener en cuenta que los procesos de despliegue continuo dependen en gran medida del equipo de desarrollo software, que decide que mejoras y cuando se despliegan, algunas organizaciones, sobre todo con alto grado de externalización de los desarrollos, prefieren dar ese rol a equipos específicos de despliegue e implantación ajenos al desarrollo del software.

6.2 Movilidad

En las últimas décadas, con la aparición de los smartphones y la democratización del acceso a internet, se ha producido una revolución tecnológica con un nuevo paradigma de comunicación e interrelación entre personas y Organizaciones. Con nuestros dispositivos móviles podemos realizar operaciones bancarias, pagos en comercios, navegar por la web, compras online, leer prensa, consumir contenido audiovisual, controlar el estado físico, cámara digital y un sinfín de funcionalidades. En los últimos tiempos se ha incorporado la tecnología móvil a otros dispositivos como relojes, tarjetas de crédito, televisiones.

La compañía Apple revolucionó el mercado de la telefonía móvil cuando en 2007 lanzó al mercado su teléfono inteligente, en la presentación Steve Jobs[11], CEO de Apple, ponía el énfasis de que con "un iPhone disponíamos de tres dispositivos en uno: reproductor de música de pantalla táctil, revolucionario teléfono móvil y dispositivo de conexión a internet, además de añadir una cámara fotográfica".

El éxito del iPhone obligó a los grandes fabricantes de la época (Nokia, BlackBerry, Ericsson) a cambiar de estrategia, pasando a pantallas táctiles y modificando sus sistemas operativos. Algunos con más éxito que otros, como es el caso de Nokia que paso de ser el mayor vendedor de móviles a desaparecer prácticamente del mercado. Google, que ya estaba desarrollando Android llegó a un acuerdo con fabricantes de dispositivos, hardware y software junto con operadores de telefonía creando la Open Handset Alliance[12] (Alianza del Dispositivo Móvil Abierto) y liberando el código de Android bajo licencia de código abierto. A partir de este momento, los fabricantes de dispositivos comenzaron a utilizar Android como base de su sistema operativo.

El mercado actual este practicante copado por dos familias de dispositivos según su sistema operativo, por una parte, están los fabricantes que adoptaron el sistema operativo

[11] Apple kytenote, 9 de enero de 2007
[12] https://www.openhandsetalliance.com/

Android, liberado por Google, y por la otra Apple y sus dispositivos móviles sobre el sistema operativo IOS.

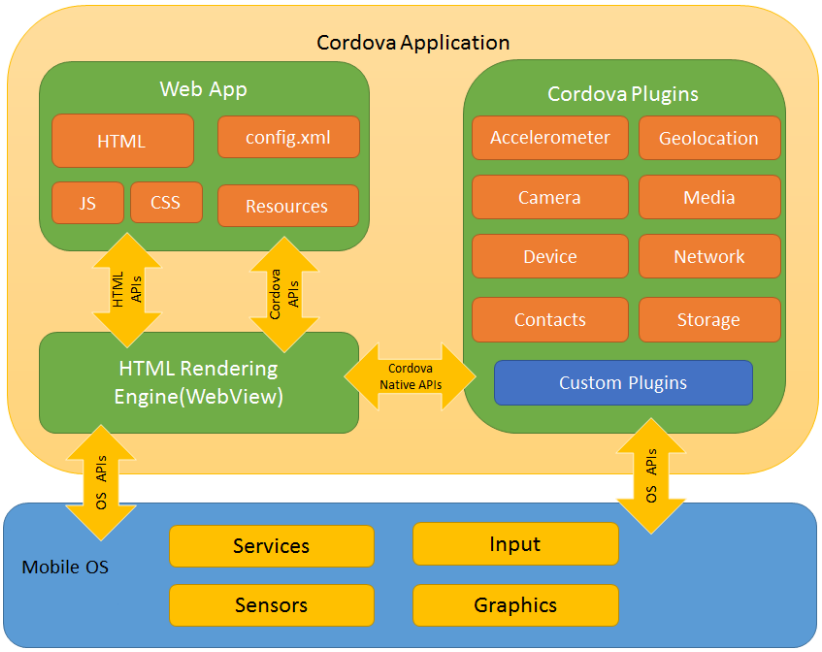
Por otra parte, tanto IOS como Android permitieron la publicación en sus tiendas o Stores de aplicaciones de terceros para que los usuarios pudieran instalarlas fácilmente en sus dispositivos, abriendo un abanico de posibilidades jamás imaginadas hasta ese momento.

Desde el punto de vista del desarrollo de aplicaciones, nos encontramos en la actualidad con dos Sistemas Operativos, por lo que si queremos llegar a un gran número de usuarios debemos desarrollar dos aplicaciones diferentes en dos entornos de desarrollo diferente con la misma funcionalidad, a este tipo de aplicaciones se las denomina **Nativas**.

Con el objetivo de compartir el código entre las diferentes plataformas surgieron proyectos como Phonegap[13] y Apache Cordova[14]. Se parte de una aplicación web desarrollada en JavaScript, HTML5 y CSS que se ejecuta en un componente WebView de la plataforma para la que se genere la aplicación. El sistema también dispone de "plugins[15]" o componentes que nos van a permitir interactuar con los dispositivos como si se tratase de una aplicación nativa. Las aplicaciones móviles que siguen esta arquitectura de construcción se las denomina hibrida.

Con esta arquitectura abstraemos al desarrollador del tipo de dispositivo y solo debe preocuparse de realizar la funcionalidad requerida. La aparición de Frameworks como Angular[16], Ionic[17] o ReactJS[18] hay logrado una gran operatividad y rentabilidad de este tipo de aplicaciones. El uso de tecnologías web, ampliamente generalizada en la comunidad de desarrolladores, hace que la construcción de este tipo de aplicaciones sea menos traumática que las nativas, obteniendo habitualmente mejores "time-to-market".

[13] https://phonegap.com/
[14] https://cordova.apache.org/
[15] Un plugin es un componente adicional de código que aporta nueva funcionalidad a un progama
[16] Angular, framework JavaScript de Google basado en "single on page" https://angular.io/
[17] Framework para desarrollo de aplicaciones híbridas que se apoya en JS, HTML5 sobre Apache Cordova.
[18] Framework JavaScript "single on page" liberado por Facebook https://es.reactjs.org/



3 Arquitectura Apache Cordova

Fuente: https://cordova.apache.org/docs/en/latest/guide/overview/index.html

A la hora de decidirnos por una arquitectura, nativa o híbrida, para construir una aplicación debemos valorar sobre todo la funcionalidad de esta, hay que tener en cuenta que una aplicación Nativa siempre tiene un mejor rendimiento, sobre todo si nuestra aplicación debe utilizar el hardware del dispositivo o partes de su core[19], como la cámara, la geolocalización, el acceso a archivos o fotos. El uso de la arquitectura de aplicaciones híbridas está más recomendado para funcionalidades orientadas a gestionar información, sobre todo el acceso a servicios remotos como prensa online, administraciones públicas o donde el acceso a los componentes del dispositivo es más secundario. Por tanto, no se puede decir que una arquitectura sea mejor que otra, sino que dependiendo de los requisitos debemos elegir la que mejor se amolde a ellos. Hay que mencionar que el desarrollo de aplicaciones hibridas, al basarse en aplicaciones web y por el uso intrínseco de Apache Cordova o Phonegap hace de orquestadores como npm[20], nos facilita la inclusión de su ciclo de vida en procesos de CI/CD.

[19] Core. Núcleo central o base de un programa software
[20] Node Package Manager, gestor de dependencias de NodeJs. Ver https://www.npmjs.com/

6.3 Repositorio de control de Versiones

Un repositorio de control de versiones (VCS, *Version Control System*) es un software que nos permite centralizar nuestro código fuente, de esta manera, todo el equipo trabaja sobre el mismo código. Los VCS nos permiten:

- Controlar colisiones de código: Dos miembros del equipo desarrollan sobre el mismo código fuente, un VCS nos permite gestionar los cambios y el merge entre ambos.
- Nos permite volver a versiones anteriores de nuestro código sin dificultad.
- Siempre disponemos de una copia del código, sin depender de los equipos locales de los desarrolladores.

Hay dos tipos diferenciados de software:

- Centralizado: se basan en un único servidor donde los desarrolladores se encargan de realizar el commit de su código. El sistema controla las subidas e impide que se produzcan conflictos entre los desarrolladores al trabajar sobre los mismos ficheros. Estos conflictos se deben solventar de forma manual, otro hándicap que nos podemos encontrar es la caída del servidor central que nos va a impedir trabajar con garantías en nuestro código. Apache Subversion[21] es un claro exponente de esta tecnología y durante muchos años fue el software más utilizado.
- Distribuido: Buscado mejorar los puntos débiles de los sistemas centralizados, la arquitectura distribuida realiza una réplica del repositorio central sobre la instalación local del desarrollador, de esta manera trabaja de manera independiente del repositorio central, pero se mantiene el mecanismo de control sobre los conflictos y su resolución. El mayor

[21] https://subversion.apache.org/

exponente es git[22] y los softwares comerciales que utilizan su core (Github, gitLab, bitBucket. etc.).

Actualmente el uso de git está ampliamente difundidos, quedando subversión y los sistemas basados en repositorios centralizados con un uso residual.

6.4 Orquestadores de tareas

El auge de las metodologías Agile en el desarrollo de software y la amplia aceptación de prácticas DEVOPS han dado un gran impulso a herramientas de CI/CD, que permiten poner en práctica aspectos importantes de dicha metodología.

La herramienta más extendida es Jenkins, aunque están tomando cierto relieve otras como Bamboo, TravisCi, Circleci, TeamCity o la incorporación en GitLab de un orquestador de CI/CD totalmente integrado con el repositorio. La mayoría de los productos se basan en la ejecución de "pipeline" o script que permiten personalizar las diferentes fases de Ci/CD.

Jenkins es un producto bajo licencia Open Source que nos permiten automatizar y centralizar tareas como la construcción de programas, despliegues en servidores, creación de artefactos y cualquier tarea dentro del ciclo de vida del software. Como principales características tenemos:

- Multiplataforma: Podemos instalarlo en Windows, Linux o Mac. Además, su instalación es sencilla.
- Dispone de una interfaz web amigable y con una curva de aprendizaje relativamente pequeña.
- Buena documentación y amplia comunidad de desarrolladores y usuarios que comparten muchas soluciones.
- Podemos personalizar y ampliar su funcionamiento con plugins desarrollados por la comunidad.

[22] https://git-scm.com/

- Posibilidad de disponer de "esclavos" en otras tecnologías lo que nos facilita centralizar tareas sin dependen de Sistemas Operativos.

Travis CI es una herramienta de CI on Cloud[23] para proyectos que utilicen como repositorio GitHub o bitBucket. Su uso es gratuito para proyectos públicos y de pago para los privados. La versión de pago dispone de la posibilidad de despliegues on premise[24]. Las tareas de CI se describen en YAML, y permite personalizar las fases, aunque de una forma más limitada a lo que podemos hacer con Jenkins. Siguiendo la filosofía de Travis CI, **CircleCi** es una alternativa de pago que dispone de versiones Cloud y "*on Premise*". Añade las características de alta disponibilidad y escalabilidad. La interfaz de usuario está más cuidada que en Jenkins y nos ofrece mejores informes. Al tener soporte del fabricante nos evitamos disponer de recursos propios que administren el sistema. Sus limitaciones principales es que solo trabaja con repositorios git y que la configuración del pipeline es más limitada que en Jenkins.

La compañía Atlassian, con sus productos muy dirigidos a metodologías Agile, ofrece **Bamboo**, tanto en la nube como "on premise" nos proporciona todas las funcionalidades para compilar, realizar test y desplegar nuestros artefactos en los lenguajes de programación más populares y en entornos multiplataforma. Dispone de una integración completa con git, además con el resto de los productos Atlassian como Jira o Bitbucket.

Un caso especial es **GitLab**, basado en git en la nube**,** su funcionalidad principal es la de repositorio de control de versiones, aunque añade un ecosistema de funcionalidades, como bug trucking, wiki, o CI, que es lo que nos ocupa, y nos permite realizar en gran parte del ciclo de vida del software en una sola herramienta. Dispone de una versión libre con funcionalidades limitadas y varios planes de pago, también podemos instalarlo "on premise".

En general, los productos de pago añaden funcionalidad, alta disponibilidad mejores integraciones, pero es posible que se pierda mucha de la versatilidad de Jenkins.

[23] Servicio accesible en la nube, accesible en internet.
[24] Que se instala en un servidor local de la organización.

	Plataforma	Licencia	personalización	integración	Documentación
Jenkins	Multiplataforma on Premise	OpenSource	Alta	Svn, git, mercurial, etc.	Amplia comunidad
Travis CI	OnCloud	Propietario	Media	GitHub, BitBucket	Documentación de producto y soporte en versión de pago
Circle CI	Oncloud/ on Premise	Propietario	Media	git	Documentación de producto y soporte en versión de pago
Bamboo	Oncloud/ on Premise	Propietario	Media	git	Documentación de producto y soporte en versión de pago
GitLab	Oncloud/ on Premise	Propietario	Media	gitLab	Documentación de producto y soporte en versión de pago

4 Comparativa de Herramientas CI/CD

6.5 Revisión de código estático

En el mercado existen varias suite o paquetes globales que realizan tanto las revisiones de código estático, pruebas de caja negra, auditorias de seguridad y pruebas de penetración dentro de un ciclo propio de CI/CD. A continuación, se describen brevemente algunas de estas herramientas.

Veracode[25] ofrece soluciones de aplicaciones y servicios de seguridad que el mundo actual requiere de software. El ecosistema Veracode nos permite realizar análisis de seguridad sobre nuestro software. Realiza análisis de código estático y revisión de librerías de terceros, revisión de seguridad dinámica sobre ejecución y pruebas de penetración.

Fortify Software Security Center (SSC), de Micro Focus[26]**,** ayuda a crear software seguro mediante la identificación de vulnerabilidades con menos esfuerzo, en menos tiempo y manteniendo la calidad del código. Se pueden realizar pruebas de seguridad estáticas (SAST), pruebas dinámicas (DAST) y pruebas de penetración (pen testing). Fortify SSC es un ecosistema de seguridad ampliamente reconocido en la industria como el principal framework de seguridad para el aseguramiento de la calidad y la seguridad de sus productos amenazas.

Checkmarx[27] también aporta un universo de herramientas que nos van a permitir realizar análisis SAST, DAST y pruebas de penetración, para la revisión de código estático

[25] https://www.veracode.com/
[26] https://www.microfocus.com/es-es/products/software-security-assurance-sdlc/overview
[27] https://www.checkmarx.com/

aporta una solución de análisis precisa y flexible, escanea automáticamente código no compilado e identifica cientos de vulnerabilidades de seguridad en los lenguajes de codificación más frecuentes. El precio del producto depende tanto de las herramientas que pretendamos utilizar como del servicio

Synopsys[28]: al igual que sus competidores, presenta una serie de productos que nos permiten realizar todo tipo de pruebas de seguridad. Su herramienta de revisión de código estático permite analizar rápidamente grandes proyectos que superan los 100 millones de líneas de código. Al ofrecer integraciones con herramientas de desarrollo, Synopsys habilita las pruebas de AppSec a velocidad de DevOps y ha ayudado a miles de organizaciones a comercializar más rápido con costos y riesgos reducidos. Al igual que en el caso anterior, son varias herramientas diferentes las que nos proporcionan los diferentes análisis y con licencias individuales cada una.

IBM® Security AppScan®[29] y Aplicación Security on Cloud mejoran la seguridad de las aplicaciones web y móviles, mejoran la gestión de los programas de seguridad de las aplicaciones y fortalecen el cumplimiento de las normativas. Probar aplicaciones web y móviles antes de la implementación puede ayudarlo a identificar riesgos de seguridad, generar informes y corregir recomendaciones.

	Análisis Estático	Análisis Dinámico	P. Penetración	CI/CD	Tipo de Licencia
FortiFy	✓	✓	✓	✓	Pago según uso del servicio a media
Veracode	✓	✓	✓	Difícil	≈ 4500 $ al año
Checkmarx	✓	✓	✗	RestApi	Pago según uso del servicio a media
Synopsys	✓	✓	✗	✓	Pago según uso del servicio a media
AppScan	✓	✓	✗	RestApi	Pago según uso del servicio a media
SonarQube	✓	✗	✗	✓	OpenSource versión estándar

5 Comparativa de Herramientas de seguridad

[28] https://www.synopsys.com/
[29] https://www.ibm.com/developerworks/library/se-scan/index.html

Para nuestro ejemplo de CI/CD me he decantado por una herramienta Open Source, **SonarQube**[30], que presenta una gran integración con Jenkins. La elección se basa en el elevado coste de las herramientas para el mantenimiento de la calidad y de la seguridad, aunque algunas disponen de versiones "trial", su funcionalidad en estos casos es limitada y no facilita procesos CI/CD. Por su elevado coste, este tipo de herramientas son más utilizadas por organizaciones de cierto tamaño o que disponen de un porfolio de aplicaciones donde la seguridad es un requisito crítico.

Una alternativa más asequible, teniendo en cuenta las diferencias con Fortity o Synopsys, es Sonarqube, que está abalado por **OWASP**[31], y nos permite analizar y medir continuamente la calidad técnica dentro de nuestro ciclo de CI/CD. Su funcionalidad es fácilmente ampliable mediante complementos que nos permite añadir nuevos lenguajes de análisis, tipos de análisis, funcionalidad extra, etc.

Sonarqube nos reporta información sobre el estado de nuestro código en relación con la calidad y seguridad, los principales indicadores son:

- Vulnerabilidades.
- Bugs o errores de programación.
- Incumplimiento de estándares de programación.
- Porcentaje de código duplicado.
- Mantenibilidad, como es de complejo nuestro código.
- Cobertura de pruebas e informe de test.

Analiza más de 20 lenguajes como JAVA, JavaScript, PHP, Cobol, PL, C#, …, para algunos lenguajes hay que obtener una versión de pago, como PL/SQL.

SonarQube tiene tres tipos de reglas:

- reglas de confiabilidad (bug).
- Reglas de seguridad o vulnerabilidad

[30] https://www.sonarqube.org/

[31] Open Web Application Security Project. Organización Internacional cuya misión principal es propiciar el software seguro, liberando recursos Open Source. https://owasp.org/

- Mantenimiento, llamadas de “code smell” o código que huele mal, es decir, sin ser un error no parece seguir reglas estándar o es en farragoso.

Otra ventaja que podemos indicar es la facilidad que proporciona para que podamos definirnos nuestras propias reglas, mediante XPath o codificándolas en JAVA

Una vez realizado el análisis, Sonarqube nos va a permitir gestionar las incidencias o marcarlas como falso positivo si es el caso. Por la propia filosofía del producto, donde la regla señala cualquier sospecha de incumplimiento, es muy común encontrarnos con falsos positivos.

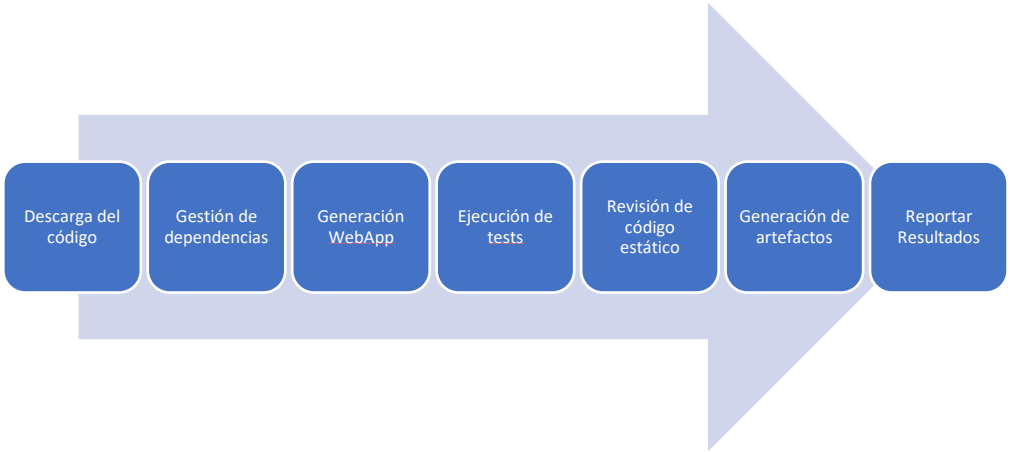
Por último, también nos va a permitir una gran flexibilidad a la hora de personalizar el Umbral de Calidad (Quality Gate), des definirla a nivel de global como por proyecto individualizado.

7 Definición de arquitectura

En los siguientes apartados se va a ir desgranado como se ha implementado cada fase de CD para aplicaciones Móviles, comenzando por la implementación y metodología de trabajo con el repositorio de versiones de nuestro código. Seguidamente, se muestra el diseño la orquestación de las fases siguientes, elección de software y arquitectura hardware propuesta.

Se implementa CI/CD sobre la generación de app móviles hibridas. Las fases que se han realizado son:

1. Descarga del código
2. Gestión de dependencias
3. Generación WebApp. Para realizar ejecución de pruebas y revisión de código.
4. Ejecución de pruebas
5. Revisión de código estático.
6. Generación de artefactos
    a. Aplicación Android
    b. Aplicación IOS
7. Comunicación de resultados al equipo. Generación de ticket de trabajo en caso de ser necesario.



6 Fases de la arquitectura

7.1 Repositorio de Versiones

Tras el estudio del punto 6.3 sobre de software específico para la gestión de repositorio de versiones, la elección para implementar nuestro repositorio pasa por utilizar git sobre otras herramientas como subversión (svn) o mercurial, esta elección está motivada por los siguientes aspectos:

- La gestión de versiones y ramas es mucho más sencilla en git, svn trabaja las ramas de forma menos imitativa, creando carpetas independientes. La solución de conflictos es mucho más engorrosa.
- La gestión de control de acceso, al trabajar git de forma distribuida, nos permite gestionar mejor quien y como se realizan los merges y los commits. Con svn se debe otorga a todo el equipo la posibilidad de realizar los commit y los merges.
- Que sea git un sistema distribuido nos da la ventaja de trabajar en entornos de mala conexión o sin conexión al repositorio central, esto no sucede con svn que dispone de un único repositorio centralizado.
- Git presenta una gran sinergia para la integración con CI, con git este proceso es a priori más sencilla que con otros productos,
- Gran difusión en la comunidad de desarrolladores.

Ahora bien, podemos utilizar git directamente o apoyarnos en alguno de los diversos productos que implementan git y que hacen el trabajo más amigable, con una interfaz web bastante intuitiva. En este caso, la elección es gitLab[32], el motivo principal es que nos permite tanto trabajar en la nube como disponer de la instalación para que nuestro repositorio lo tengamos "on premise" en nuestros servidores. Utilizar un producto u otro basado en git no influye en la implementación de CI/CD.

Una vez seleccionado el software hay que plantear un método de trabajo correcto que además nos permita avanzar en la línea adecuada en la implantación de CI/CD. En caso de Git, es lo que se denomina "gitflow", flujo de trabajo donde se describe el proceso y la

[32] https://gitlab.com/

creación de ramas de integración del código en el repositorio desde las versiones de desarrollo hasta la creación de las versiones “release”.

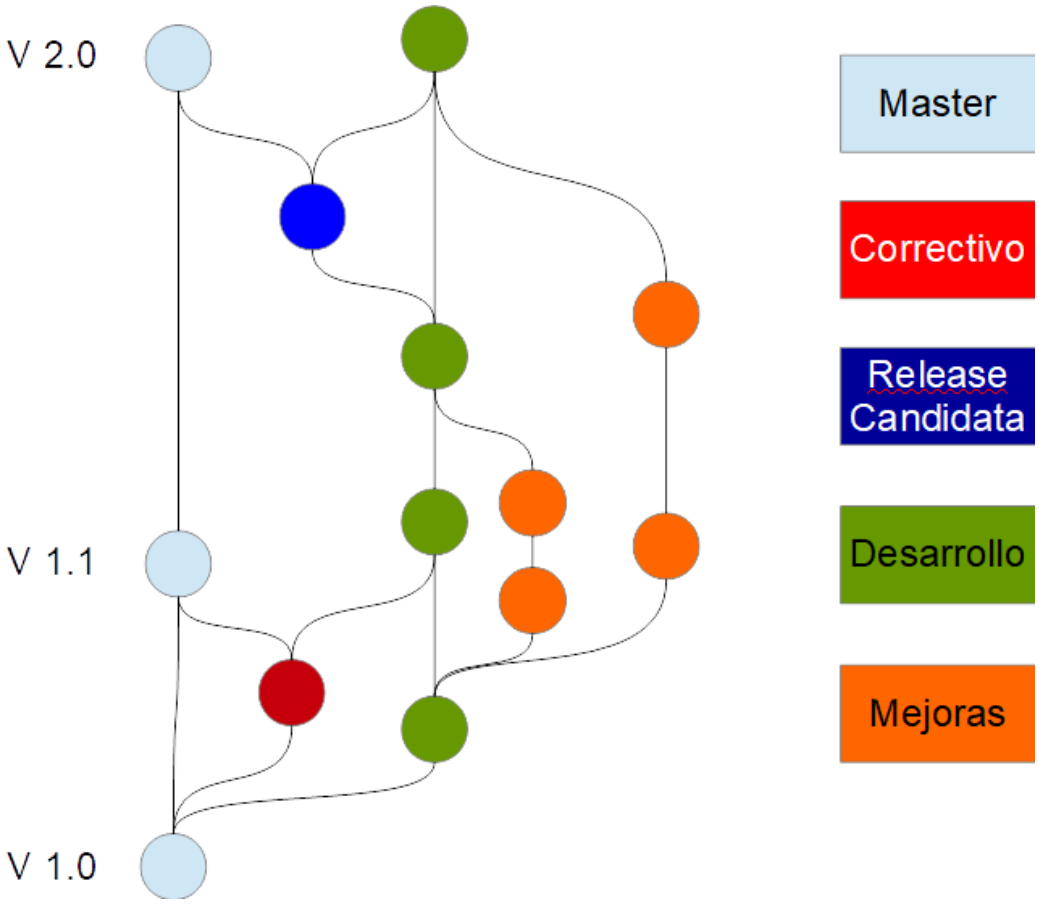
Partimos de una rama principal de nuestro código, denominada **master,** donde se encuentra las versiones estables que se va instalando en los entornos productivos.

La segunda rama en importancia es la rama de **desarrollo**, se genera de nuevo desde la versión release estable de la rama master. De esta rama surgen y se integran las mejoras planificadas para la generación de la siguiente versión estable.

Cada mejora planificada dispondrá de su propia rama, que surge a partir de la rama de desarrollo y que una vez se termina confluye en esta.

Cuando tengamos las mejoras preparadas en la rama de desarrollo prepararemos desde esta una rama de **versión o release candidata,** sobre este código se realizarán las pruebas de aceptación con el cliente, y en el caso de ser necesario, se arreglan los bugs de funcionamiento encontrados. Sobre esta rama no se pueden incluir mejoras. Una vez que la reléase candidata tenga el visto bueno, se realizara la integración del código a la rama master, etiquetando el número de versión creada, también se realiza la integración del código con la rama de desarrollo.

Una vez que nuestro sistema está en producción pueden surgir errores graves que necesiten un arreglo inmediato, fuera del flujo descrito más arriba. En este caso, desde la rama máster se creará una rama especifica de **correctivo** y una vez reparado el problema pasaremos a integrar con la rama master, etiquetando la versión y subversión correspondiente, y con la rama de desarrollo, con el fin de repercutir también el arreglo en el ciclo de mejoras.



7 Gitfow propuesto

7.2 Arquitectura CI/CD

La elección de orquestados de nuestra CI/CD es Jenkins, los motivos principales de dicha elección se basan en:

- La facilidad y flexibilidad de uso e instalación en cualquier sistema operativo y servidor
- Una curva de aprendiza relativamente pequeña
- Amplia difusión en la comunidad de desarrolladores.

Jenkins puede instalarse tanto en Linux, Windows como Mac OS X y arrancar como un despliegue war [33]JAVA en un contenedor de aplicaciones o sobre un puerto directamente. En este caso, la propuesta es desplegar Jenkins sobre un contenedor tomcat 9[34] en un sistema operativo Linux tipo CENTOS 7, en entornos empresariales es recomendable utilizar un Red Hut Enterprise Edition 7.

Se debe montar también un entorno de compilación y generación de app compatible con las versiones de sistemas operativos de los dispositivos móviles. Android Studio y SDK Android, la suite para la generación de app de Android está presente en Windows, Linux y Mac OS. Los dispositivos IOS necesitan un sistema operativo Mac OSX para la generación de sus aplicaciones. Por tanto, nos vemos en la necesidad de montar un nodo esclavo en Jenkins para la generación las aplicaciones IOS, por lo que veo también conveniente que todo el proceso se orqueste desde dicho servidor. El desarrollo de las aplicaciones móviles híbridas sobre el que queremos implantar CI/CD se sustenta sobre Apache Cordova con NodeJS como gestor de dependencias y automatizador de tareas, también compatible con OS X.

También hay que tener en cuenta el software que emplearemos para la revisión de código estático, la elección es SonarQube por la facilidad de integración con Jenkins, está avalado por OWAPS y posee una versión community que encajaba bien con lo que

[33] Acrónimo de Web Archive. Empaquetado Java de una aplicación web para su despliegue en servidores de aplicaciones
[34] Contenedor de aplicaciones web Java liberado por Apache

pretendíamos abordar. La recomendación para entornos empresariales es instalar SonarQube en un servidor dedicado, aunque en este caso lo instalaremos en el mismo que alberga Jenkins.

Por último, se ha realizado la integración con el producto Jira Software que nos va a permitir el seguimiento de nuestro desarrollo.

Barajé otras opciones Open Source como MantisBT, OpenProject o Bugzilla. Realizando varias pruebas de concepto sobre funcionalidad y, sobre todo, integración con CI/CD con resultado poco alentador.

MantisBT (Mantis Bug Traker) es una herramienta con Licencia GNU[35] que nos permite gestionar incidencias de nuestros productos y tareas en entornos colaborativos, basado en PHP y base de datos MySQL podemos instalarlo "on Premise" o en modo SAAS o Hosting, esta versión es de Pago. Permite una amplia personalización en su funcionalidad y características (estados, grupos de trabajo, características, roles), además presenta un interfaz de usuario es intuitiva y bastante amigable. A diferencia de Jira o Gitlab Inssues, no dispone de tableros para metodologías agiles, aunque sí que dispone de una vista muy intuitiva de agrupación por estados de las incidencias. Fue la primera opción que probé, en una instalación sobre CentOS7 y base de datos Maria DB. La instalación es sencilla. No elegí esta opción porque la integración con Jenkins está obsoleta, el plugin lleva más de 6 años sin actualizarse y lo marca como vulnerable por ataques CSRF[36], además encontré errores con la integración de la versión 2.24.2, posiblemente por la antigüedad del plugin. Otra forma de integrarse es mediante el API REST que proporciona la herramienta, es decir, programar un cliente REST en lenguaje groovy para ser utilizado en nuestro pipeline. Teniendo en cuenta la facilidad de integración de otras herramientas y que aportaban más funcionalidades, opte por desechar MantisBT.

Bugzilla se basa en una filosofía similar a MantisBT en lo que se refiere a la gestión de incidencias, fue creado por mozilla y es de libre distribución. Está programado en perl y su

[35] GNU – General Public License
[36] Cross Site Request Forgery

base datos puede ser MySQL, PostgreSQL, Oracle. Como MantisBT, también podemos instalarlo tanto en Windows, Linux y Mac OS. Es una herramienta muy potente y ampliamente extendida, pero a la hora de valorar la integración con Jenkins me encontré con el mismo problema, el plugin lleva 9 años sin actualizarse cuando la última versión estable del producto es de 2019, además. presenta alertas de seguridad. Dada la experiencia con MantisBT opte por descartar esta solución.

Existen otras herramientas soluciones on Cloud como OpenProject o Trello que ofrecen un entorno colaborativo de gestión de tareas orientado a metodologías agiles pero que en el seguimiento de bugs y su funcionalidad es algo más limitada, además no disponen de Integración con Jenkins mediante plugins por lo que la solución pasaba, al igual que en los casos anteriores, por programar la integración a medida mediante REST API.

Al final la elección de Jira Software, que es una herramienta de Atlassian para la gestión y planificación de proyectos que abarca desde la toma de requisitos a la gestión de Service Desk. No es una herramienta Open Source aunque nos permite disponer de versiones limitadas de libre uso. Presenta funcionalidades que facilitan un entorno colaborativo entre los miembros del equipo, hacer un seguimiento del flujo de trabajo, de los requisitos y de las incidencias. También presenta herramientas para realizar metodologías de desarrollo ágil como scrum y Kanban y gestionando fácilmente los sprints de entregas. Las principales características son:

- Backlog de proyectos (historias o requisitos de usuario)
- Gestión y seguimiento de incidencias y tareas
- Personalización de flujos de trabajo
- Tableros Scrum y/o Kanban
- Planificación de publicaciones y sprints
- Integraciones de CI/CD
- Informes y análisis
- Integración de Jira Service Desk.

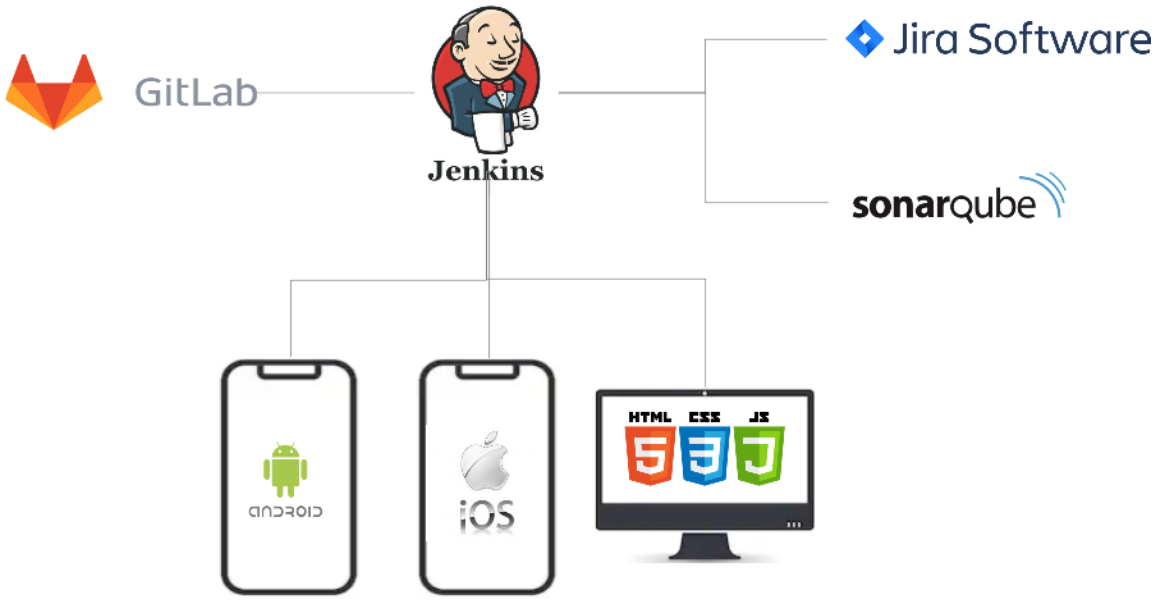
Jira no es un producto Open Source, pero dispone de una versión libre en la nube que nos permite tener proyectos públicos y una versión "On Premise" con una licencia de prueba de 30 días. Para el presente trabajo he realizado pruebas con ambas versiones, aunque la versión definitiva es sobre proyectos en la nube con Jira Cloud ya que nos permite rebasar en tiempo el periodo de 30 días.

Las razones principales de la selección de Jira son:

- Es un producto referente en el sector.
- Es una herramienta completa que permite realizar la gran mayoría de las actividades del ciclo de vida de un desarrollo en un entorno colaborativo.
- Es intuitiva y fácil de aprender
- La integración con CI/CD es sencilla

Para realizar un seguimiento de las ejecuciones de nuestro proceso de CI/CD este va a crear una serie de tickets de trabajo en nuestro proyecto Jira dependiendo del resultado de la ejecución:

- Ticket de error en caso de que la de ejecución de pipeline falle.
- Ticket de error en caso de test con errores o con incumplimientos de los umbrales de calidad.
- Se abrirá una tarea de revisión de resultados cuando la ejecución sea correcta. Indicando la rama de ejecución y los entornos donde debe ser desplegado. Si la rama de ejecución es master. Se abrirá tarea de despliegue de aplicación en entornos productivos.



8 Arquitectura Propuesta

7.3 Selección de aplicación para la prueba de Concepto.

Una vez definida la arquitectura pasaremos a describir el proyecto de aplicación híbrida sobre el que realizaremos el ciclo de CI. Como el objetivo principal del trabajo es la definición de un proceso de CI/CD he preferido apoyarme en un framework basado en Apache Cordova que nos va a permitir, mediante la utilización de "*templates*" o plantillas, obtener una aplicación de ejemplo operativa con las principales características de una aplicación móvil y que además incluya en el proyecto ejemplos de test unitarios, compilaciones por entorno, control de errores de TypeScript con TsLint, etc.

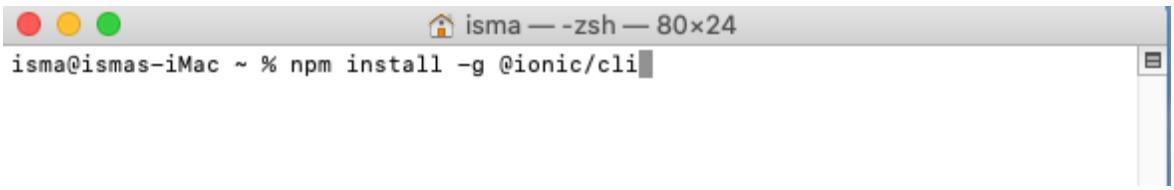
El framework seleccionado es **IONIC**, que incluye una serie de herramientas que nos facilita realizar aplicaciones móviles sobre tecnologías de desarrollo web como HTML5, JavaScript y CSS. Nos permite realizar nuestras webApp sobre los framework JavaScript ReactJS y Angular o si lo preferimos, sobre JavaScript directamente, su objetivo principal está centrado en simplificar la programación de la interfaz de usuario, facilitando la implementación de controles, eventos, animaciones, etc. Presenta una base de componentes, estilos y plantillas adaptados a todas las plataformas y optimizados en rendimiento, tanto para dispositivos móviles como para aplicaciones web, con el objetivo que los programadores solo se preocupen de la funcionalidad. Sus principales características son:

- Multiplataforma: A partir del mismo código obtenemos aplicaciones para dispositivos iOS y Android, aplicaciones de escritorio Windows y webs como aplicaciones web progresivas.
- Uso de estándares de desarrollo web (HTML, CSS y JavaScript).
- Interfaz de usuario amigable y funcional para todas las plataformas
- Curva de aprendizaje relativamente pequeña para desarrolladores web. Proporciona una base de plantillas que nos permite empezar nuestra app desde un arquetipo funcional.
- Permite el uso de frameworks JavaScript (Angular y ReactJS) que hacen facilitan la implantación de las funcionalidades como la conexión servicios, navegaciones y transacciones.

Ionic permite desarrollar tanto en Windows como Linux y Mac OS, las acciones se realizan sobre la consola de comandos, importante destacar que el desarrollo del código se puede realizar con cualquier IDE web. Ionic se apoya para realizar sus acciones sobre **NodeJS** y comandos **npm**. Además, requiere de un cliente **git** para la descarga de dependencias y plantillas.

NodeJS es un entorno de ejecución en JavaScript en servidor, habitualmente se utiliza para crear servicios web, pero también se pueden construir herramientas de desarrollo como son Apache Cordova o el propio Ionic. Por su parte, **npm**, es el gestor de paquetes que usa NodeJS y que nos va a permitir descargar, instalar y ejecutar paquetes, así como administrar las dependencias de nuestros aplicativos. Mediante npm instalaremos los paquetes necesarios para la ejecución de Ionic:

Instalación del CLI Ionic:



9 Instalación de Ionic

Una vez instalado Ionic hay que crear nuestra aplicación a partir de una plantilla concreta y decidir si utilizaremos AngularJS o React. Con el comando ionic start –list nos mostrará las diferentes posibilidades de plantillas para comenzar nuestra aplicación.

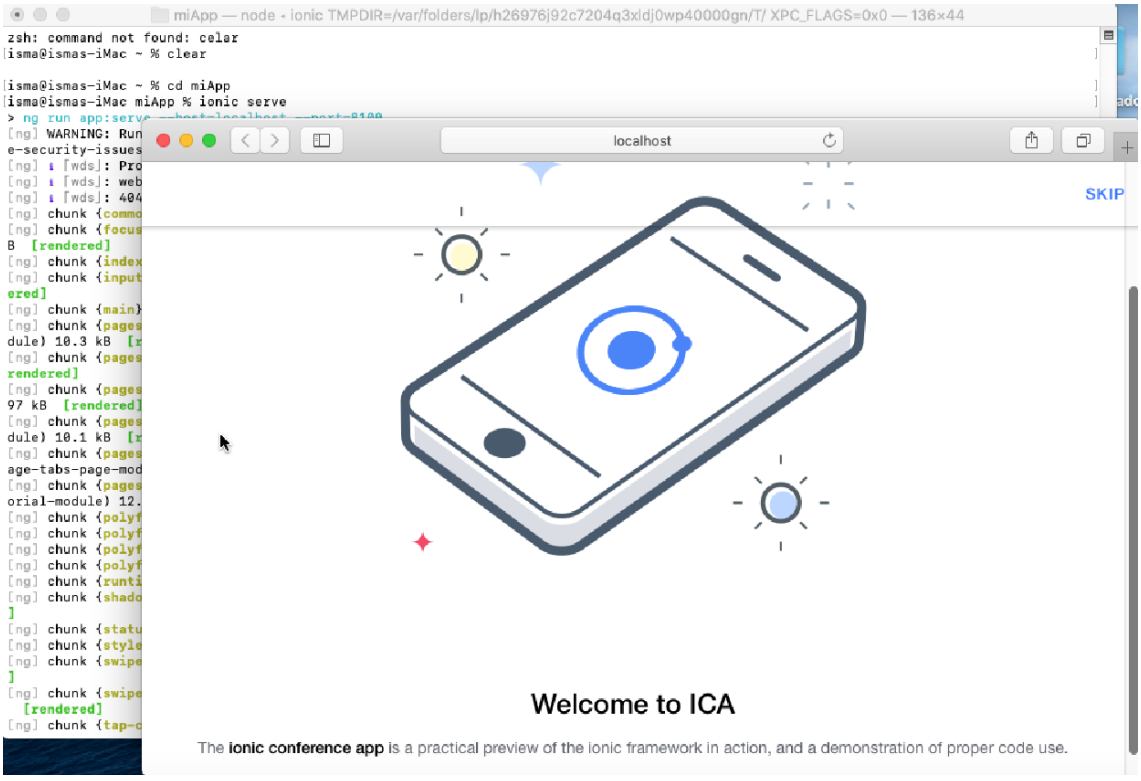
```
isma@ismas-iMac ~ % ionic start --list

Starters for @ionic/angular (--type=angular)

name          | description
--------------------------------------------------------------------------------
tabs          | A starting project with a simple tabbed interface
sidemenu      | A starting project with a side menu with navigation in the content area
blank         | A blank starter project
list          | A starting project with a list
my-first-app  | An example application that builds a camera with gallery
conference    | A kitchen-sink application that shows off all Ionic has to offer

Starters for @ionic/react (--type=react)

name          | description
--------------------------------------------------------------------------------
blank         | A blank starter project
list          | A starting project with a list
my-first-app  | An example application that builds a camera with gallery
sidemenu      | A starting project with a side menu with navigation in the content area
tabs          | A starting project with a simple tabbed interface
conference    | A kitchen-sink application that shows off all Ionic has to offer

Starters for Ionic 2/3 (--type=ionic-angular)

name      | description
--------------------------------------------------------------------------------------------------------------------
tabs      | A starting project with a simple tabbed interface
sidemenu  | A starting project with a side menu with navigation in the content area
blank     | A blank starter project
super     | A starting project complete with pre-built pages, providers and best practices for Ionic development.
tutorial  | A tutorial based project that goes along with the Ionic documentation
aws       | AWS Mobile Hub Starter

Starters for Ionic 1 (--type=ionic1)

name      | description
--------------------------------------------------------------------------------
tabs      | A starting project for Ionic using a simple tabbed interface
sidemenu  | A starting project for Ionic using a side menu with navigation in the content area
blank     | A blank starter project for Ionic
maps      | An Ionic starter project using Google Maps and a side menu

isma@ismas-iMac ~ %
```

10 Plantillas de Ionic

Con el comando ionic start procedemos a crear nuestro proyecto. Para la prueba de concepto he seleccionado la plantilla **conference** con Angular ya que es la plantilla que más funcionalidad implementa e incluye en su ciclo de vida test unitarios con karma y comprobaciones código con TSLint.

11 Creación de la app con Ionic

Una vez creada nuestra aplicación ya podemos ejecutarla en local como web app



12 Ejecución de webApp en local

Para la creación de las diferentes plataformas móvil Ionic se apoya en los comandos de Apache Cordova.

7.3.1 Creación de aplicación para IOS

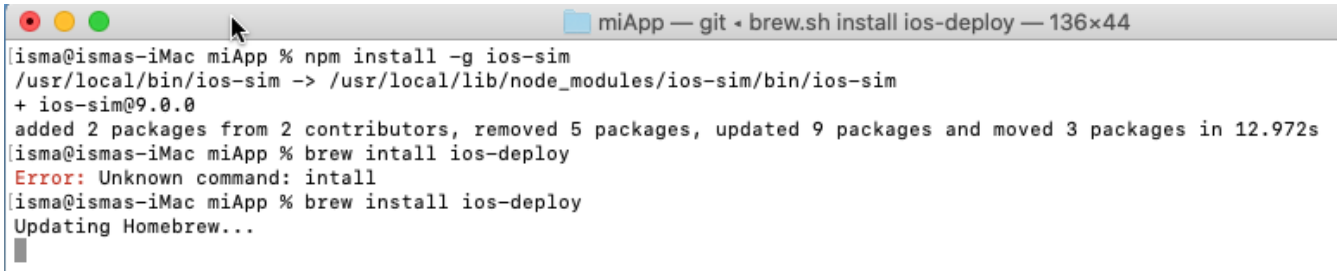
Requisitos previos:

Para crear una aplicación IOS con Ionic, como pase previo, debemos disponer de un sistema operativo MacOs con la instalación de Xcode[37] con sus herramientas de ejecución de línea de comandos.

```
xcode-select --install
```

Importante, para poder firmar nuestras aplicaciones, en XCode debemos incluir nuestra cuenta de desarrollador.

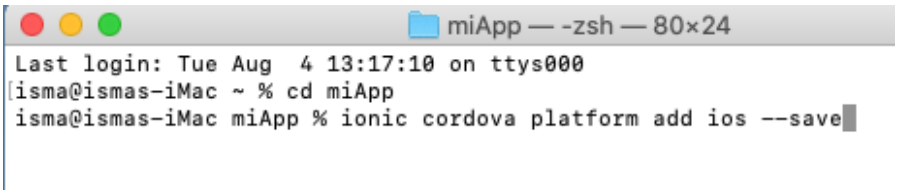
Posteriormente debemos configurar Apache Cordova para que pueda realizar las instalaciones tanto en simulador como en dispositivos directamente.



13 preparación de Apache Cordova para IOS

Configuración del proyecto:

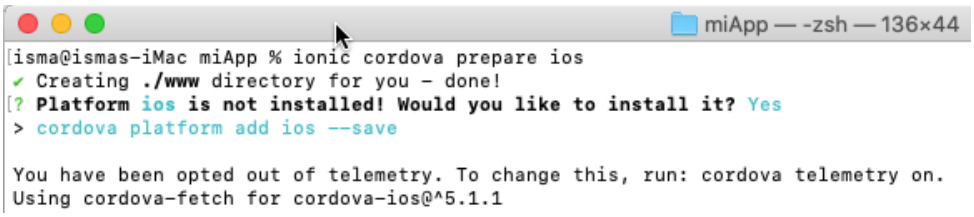
1) Creamos la plataforma de iOS en nuestro proyecto



14 Añadir plataforma IOS

2) Preparamos nuestro proyecto

[37] IDE de desarrollo de aplicaciones móviles para sistema operativo IOS



15 Preparar Proyecto IOS

3) Para compilar nuestra aplicación utilizaremos el comando build indicando la plataforma, en este caso IOS, si queremos generar la reléase para el store de Apple hay que indicar el argumento --prod.

16 Generación de app para Store IOS

4) También podemos correr nuestra aplicación en un simulador o en un dispositivo con el comando run

17 Ejecución de la app sobre un dispositivo

7.3.2 Creación de aplicación para Android

Requisitos previos:

Debemos tener instalado en nuestro equipo Android Studio con la SDK de Android[38] incluida. Una vez instalado hay que crear las siguientes variables de entorno:

- ANDROID_SDK_ROOT: path de la instalación de la SDK de Android

[38] Entorno de trabajo para el desarrollo de aplicaciones móviles para Sistema operativo Android

- ANDROID_HOME: path de la instalación de Android
- JAVA_HOME: Path de instalación de la jdk 8 de JAVA
- GRANDLE_HOME: path de instalación de grandle

Una vez definidas hay que incluirlas en la variable de entorno PATH. En el caso de MacOS se realiza con export

```
isma@ismas-iMac miApp %
isma@ismas-iMac miApp % export PATH="GRADLE_HOME:JAVA_HOME:$ANDROID_HOME:/usr/local/bin:/usr/bin:/bin:
/usr/sbin:/sbin:/Library/Apple/usr/bin"
```

18 variable PATH de entorno Android

Configuración del proyecto:

Análogamente a lo realizado con IOS, demos añadir la plataforma, en este caso Android, realizar el prepare, el build y podremos tanto generar el entregable para el Store de Android como ejecutarlo en simulador o en un dispositivo físico.

```
isma@ismas-iMac miApp % ionic cordova platform add android
> cordova platform add android

isma@ismas-iMac miApp % ionic cordova build android
> ng run app:ionic-cordova-build --platform=android

isma@ismas-iMac miApp % ionic cordova build android --prod
> ng run app:ionic-cordova-build:production --platform=android

isma@ismas-iMac miApp % ionic cordova run android
> ng run app:ionic-cordova-build --platform=android
```

19 comandos Ionic para Android

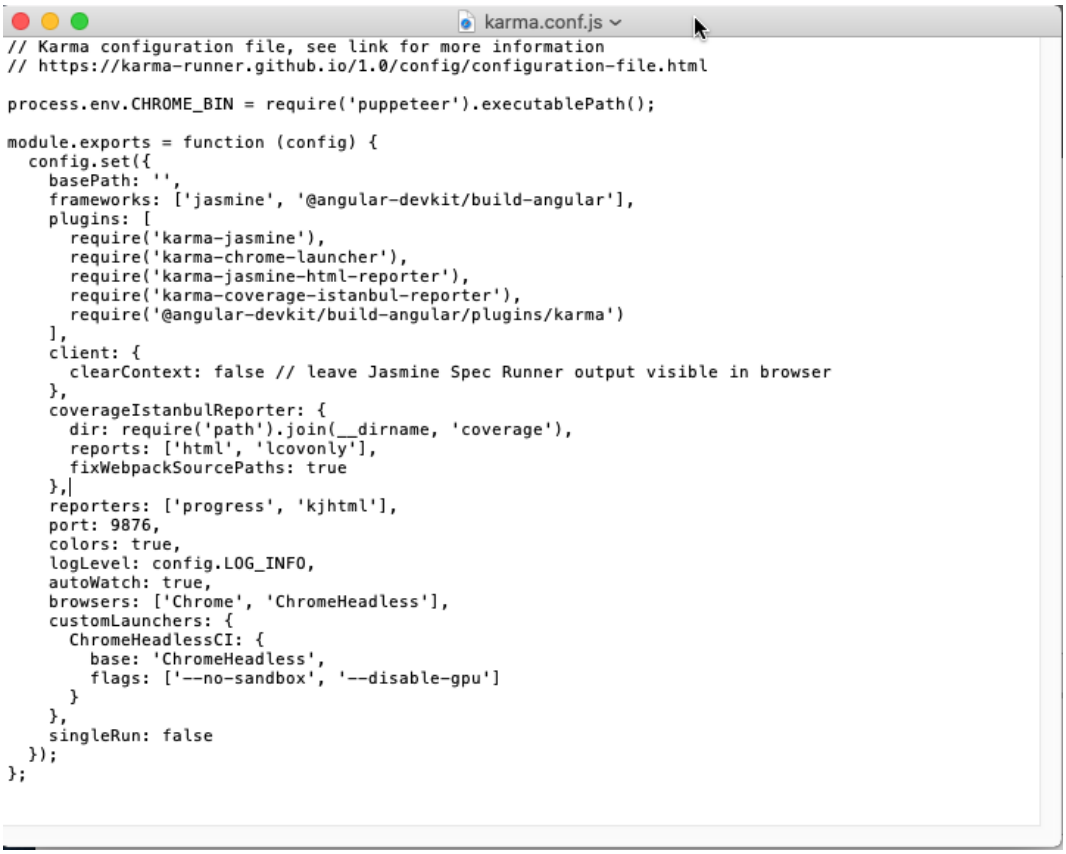
7.3.3 Definición de Test.

La creación y ejecución de test unitarios esta intrínsecamente ligada con el lenguaje de programación utilizado en nuestra aplicación, en este caso será JavaScript apoyado en el framework Angular.

Para realizar test unitarios y de integración Angular incluye **Jasmine test framework** y ejecuta los test **con karma test runner**.

- **Jasmine test Framework**[39] es framework Open Source con licencia MIT bastante extendido para pruebas unitarias en el desarrollo JavaScript, permitiendo automatizar tanto pruebas síncronas como asíncronas. Carece de dependencias externas y dispone de una sintaxis fácil. Se ejecutar sobre un navegador y proporciona los resultados en formato HTML.
- **Karma test runner**[40] es una herramienta que nos permite automatizar las ejecuciones de los test codificados en Jasmine desde la línea de comandos y de manera sencilla.

La configuración de la ejecución de test vendrá daba en el fichero karma.conf.js en la carpeta src de nuestro proyecto.

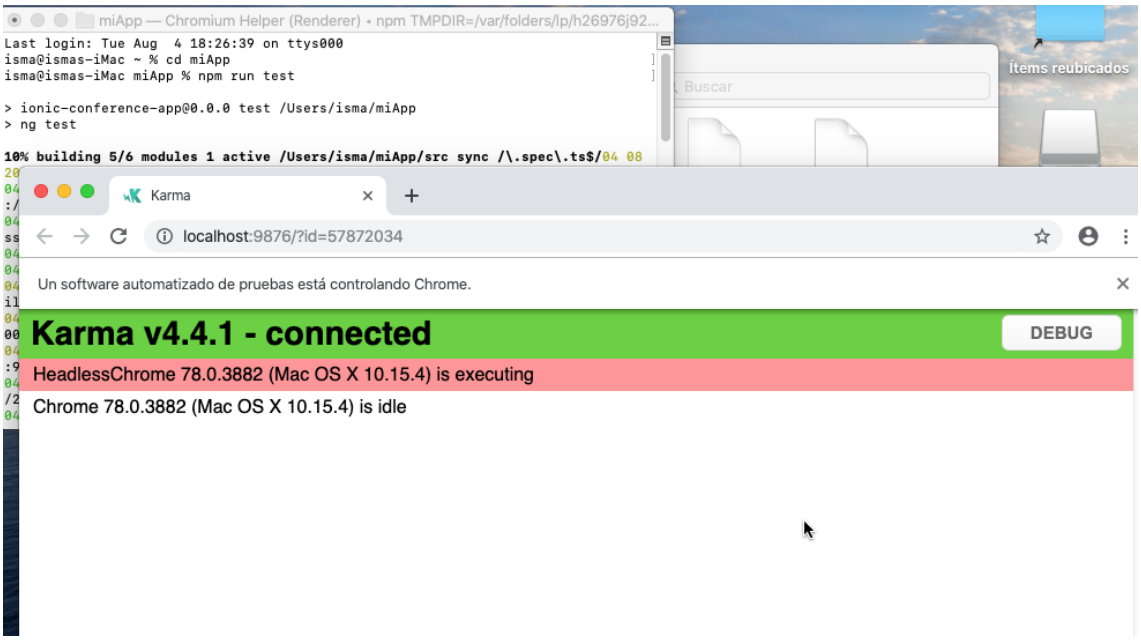


20 Configuración de Karma runner

[39] Más información en https://jasmine.github.io/
[40] Más información en https://karma-runner.github.io/latest/index.html

Los ficheros de test se localizan en la carpeta src/app de nuestros proyectos y vienen definidos por la extensión .spect.ts.

Para ejecutar los tests desde nuestro proyecto ionic con angular vamos a utilizar npm run test. Ionic ya ha configurado en la creación de nuestro proyecto para que npm ejecute los tests llamando directamente a ng test de Angular.



21 Ejecución de test con karma

Para extraer el informe de cobertura de código se utiliza *npm run test --no-watch --code-coverage*

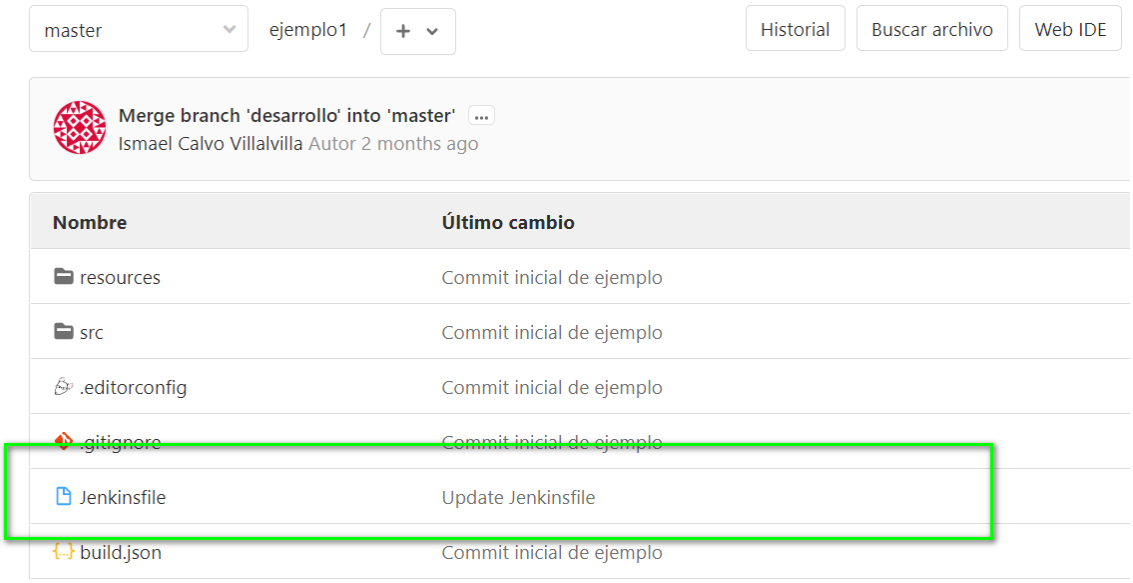
7.4 Job Jenkins

7.4.1 Definición de Jenkins Pipeline

Jenkins Pipeline[41] es un conjunto de plugins o complementos que nos va a permitir definir, relacionar y ordenar las tareas de CI/CD, se realiza mediante script codificado en un lenguaje DSL (Domain-specific Language). En el script viene definido donde queremos que se ejecuten las tareas (en nuestro caso en un esclavo Mac), las diferentes fases de CI/CD y las tareas a realizar en cada fase.

Estos scripts se guardan en modo texto, habitualmente con el nombre de JenkinsFile y se suelen incorporar como parte de nuestro proyecto de desarrollo.

En nuestro caso, el fichero va a ir incluido en el directorio raíz de nuestros proyectos. Posteriormente, desde Jenkins se indicará en el job el fichero que se debe ejecutar para nuestro proyecto.



22 Localización de fichero Jenkinsfile

[41] Mas información en https://www.jenkins.io/doc/book/pipeline/

7.4.2 Elección del tipo de Job

Partiendo de la solución planteada como metodología de trabajo para el repositorio de versiones de código, el tipo de Job Jenkins que considero que se ajusta a nuestras necesidades es Multibrach Pipeline, Este tipo de job permite seleccionar un repositorio basado en git y "descubre, gestiona y ejecuta automáticamente" las ramas que dispongan del fichero jenkinsfile definido en el Job. Por cada rama crea un job específico para su fichero Jenkins.

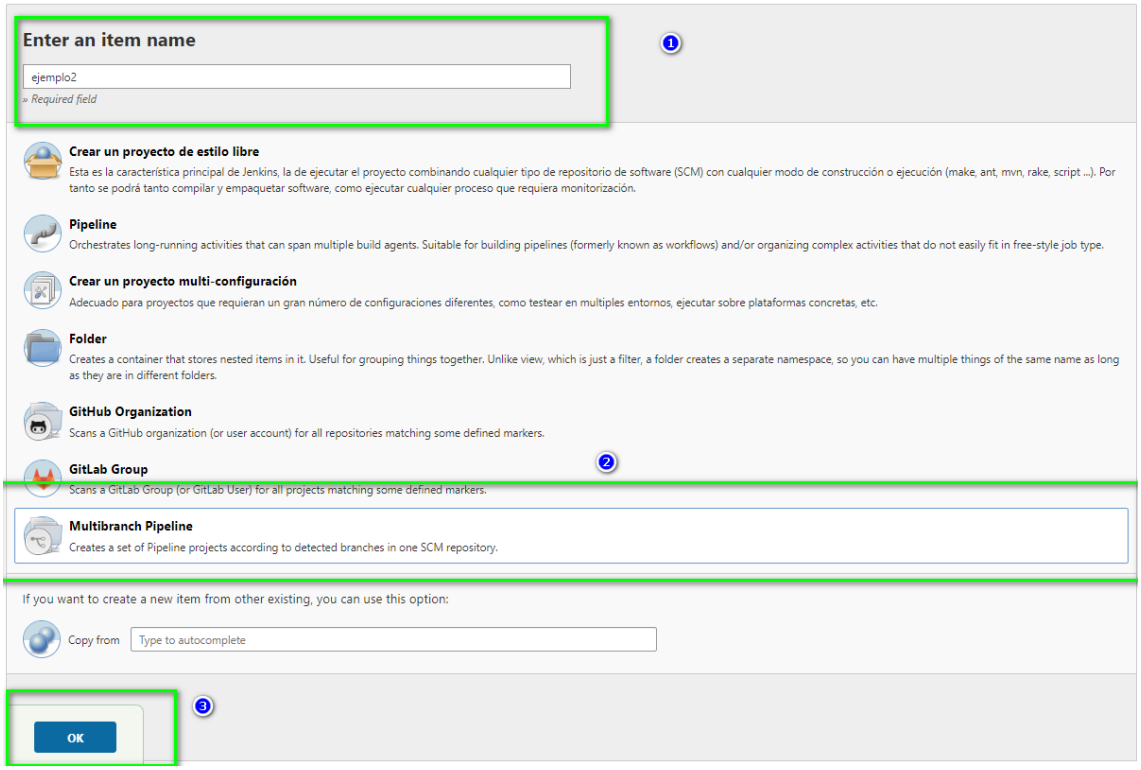
De esta forma se evitan tareas manuales sobre las diferentes ramas de nuestro proyecto, como crear nuevas tareas, asignarles el fichero JenkinsFile, programar individualmente cada rama y destruir los job una vez que desaparece la rama.

Para dar de alta nuestro Job debemos:

1. Crear nueva tarea, desde la ventana principal:

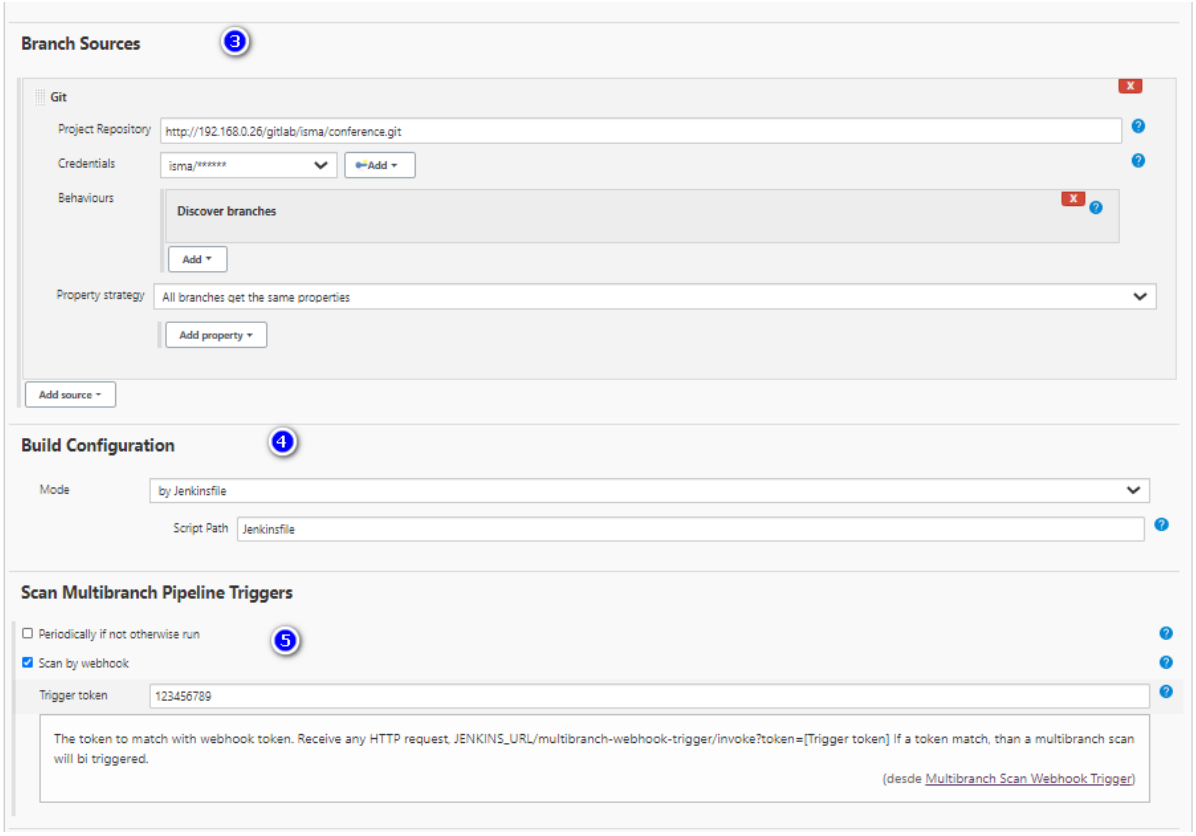
23 Nueva tarea Jenkins

2. Designamos el nombre de nuestra tarea, seleccionamos "Multibranch Pipeline y pulsamos Ok.



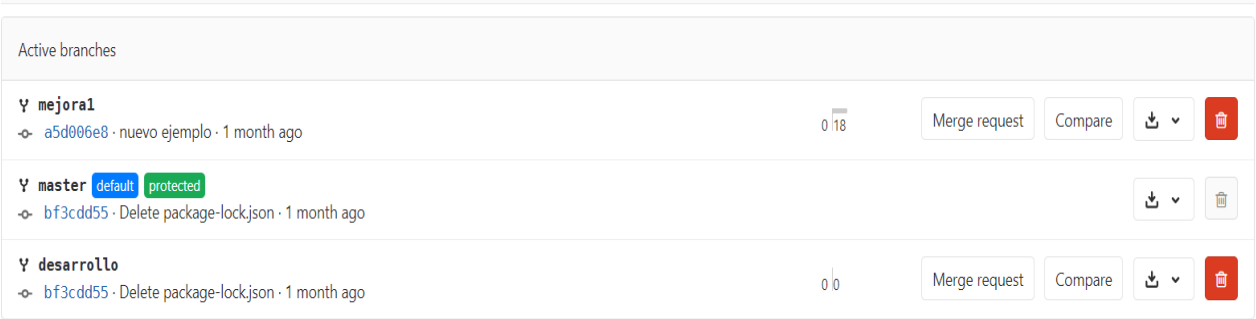
24 Alta de Multibrach Pipeline

3. Definimos el repositorio en la sección “Branch Sources” y seleccionamos el tipo de repositorio, en nuestro caso git, incorporando la información:
    - Project repository: url de acceso a nuestro repositorio.
    - Credentials: Credenciales de acceso al repositorio.
4. En la sección “Build Configuration” indicaremos el nombre del fichero con el script que debe ejecutar Jenkins.
5. En la “Scan Multibranch Pipeline Triggers” podemos definir cuando se realizará las comprobaciones sobre el repositorio de versiones. Hay dos opciones:
    - Mediante programación periódica, “Periodically if not otherwise run”
    - Que el repositorio de versiones avise a Jenkins de cambios mediante una llamada al API de Jenkins mediante un “webhoot”



25 definición Multibrach Pipeline

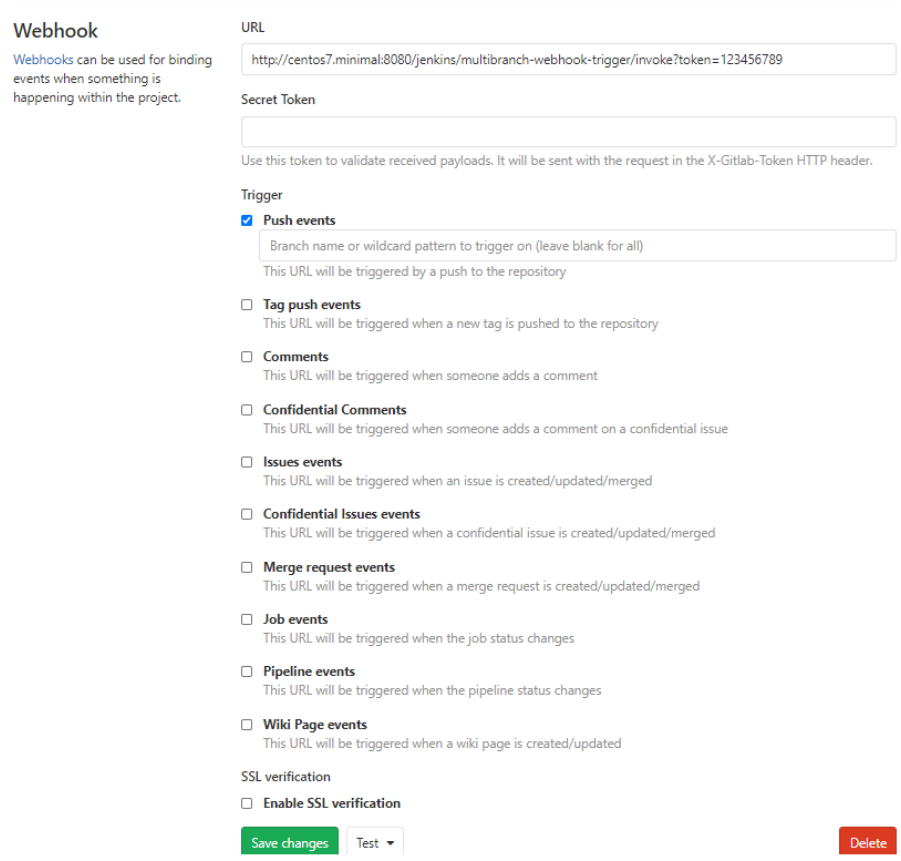
Una vez configurada la tarea o job, Jenkins escanea nuestro repositorio descubriendo cada una de las ramas y ejecutando su fichero de script correspondiente. En nuestro ejemplo del proyecto "conference" desponemos de tres ramas: master, desarrollo y mejora1 por lo que Jenkins va a escaneará y creará, colgando de nuestra tarea Multibrach "conference" los tres pipelines. También se ejecutarán.



26 Ramas del proyecto Git

27 Creación y ejecución automática de Pipelines por ramas

En nuestro caso, vamos a configurar nuestra tarea de Jenkins mediante webhook en el momento que se realice un push sobre cualquier rama. Para ello, en la consola de administración gitlab, entramos en la opción Settings→ Webhooks y creamos uno nuevo. Debemos indicar la url de Jenkins y el token configurados en el job.



28 Configuración de webhook en gitLab

7.4.3 Definición de las tareas CI/CD

En este punto vamos a definir las tareas que en sí CI/CD que se codificarán en el fichero jenkinfile que incluimos en nuestro proyecto.

29 tareas Jenkins

Primero indicamos que la ejecución la queremos realizar siempre en el nodo Mac y las variables de entorno para la ejecución en dicha maquina:

```
pipeline {
  agent  { label 'macOsServer'}
  environment {
      ANDROID_HOME="/Users/isma/Library/Android/sdk"
      ANDROID_SDK_ROOT  ="/Users/isma/Library/Android/sdk"
      JAVA_HOME="/Library/Java/JavaVirtualMachines/adoptopenjdk-8.jdk/Contents/Home/"
      GRADLE_HOME="/usr/local/Cellar/gradle/6.5.1"


      PATH = "GRADLE_HOME:JAVA_HOME:$ANDROID_HOME:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/Apple/usr/bin"
  }
```

30 Definición de nodo y propiedades del sistema

A continuación, se describen las siguientes tareas que se realizan dentro de la sección "stages"

7.4.3.1 Preparar Dependencias

Lo primero que debemos realizar con nuestro código es crear proyecto y preparar las dependencias que se han configurado para que NodeJS lo gestione en el fichero package.json. Para ello se ejecuta el siguiente paso

```
stage('preprarar dependencias'){
    steps {
        sh 'npm install'
    }
}
```

31 Preparación de entorno

7.4.3.2 Test

En el siguiente paso se ejecutan las comprobaciones de lint y los test de karma.

```
stage('test'){
    steps {
        sh 'npm run lint'
        sh 'npm run test'
    }
}
```

32 ejecución de test

7.4.3.3 Revisión de código estático con sonarQube.

El siguiente paso es la revisión de código estático mediante sonarQube, extraemos la información del fichero de configuran de NodeJS. Debemos definir:

- Nombre del proyecto.
- La clave.
- Versión de la aplicación.
- Path donde están el código fuente.
- Ficheros de exclusión.
- Localización de los reportes de test.

Una vez finalizado el análisis se comprueba si cumple con los umbrales de calidad definidos.

```
stage("SonarQube analysis") {
    steps {
      withSonarQubeEnv('sonarqube') {
          script{
            def packageJSON = readJSON file: 'package.json'
            def packageJSONVersion = packageJSON.version
            def packageJSONName = packageJSON.name
            def   scannerHome = tool 'sonar-scanner';
            def sonar = "${scannerHome}/bin/sonar-scanner"
            sonar +=  " -Dsonar.projectKey=" + packageJSONName
            sonar +=  " -Dsonar.projectName=" + packageJSONName
            sonar +=  " -Dsonar.projectVersion=" + packageJSONVersion
            sonar +=  " -Dsonar.sources=src"
            sonar +=  " -Dsonar.sourceEncoding=UTF-8"
            sonar +=  " -Dsonar.exclusions=**/node_modules/**,**/*.spec.ts"
            sonar +=  " -Dsonar.tests=src"
            sonar +=  " -Dsonar.test.inclusions=**/*.spec.ts"
            sonar +=  " -Dsonar.ts.tslintconfigpath=tslint.json"
            sonar +=  " -Dsonar.ts.lcov.reportpath=test-results/coverage/coverage.lcov"
            sh sonar
          }

      }
    }
  }
  stage("Quality Gate") {
    steps {
      timeout(time: 1, unit: 'HOURS') {
        waitForQualityGate abortPipeline: true
      }
    }
  }
```

33 Análisis de SonarQube.

7.4.3.4 Creación de plataformas y aplicaciones

Si las tareas de CI/CD ha resultado satisfactorias procedemos a realizar los pasos de creación de las aplicaciones en los diferentes entornos.

```
stage('creamos plataformas') {

    steps {

        sh 'ionic cordova platform add android'
        sh 'ionic cordova platform add ios'

    }
}
```

34 Creación de plataformas

Posteriormente se construyen las apps con el comando build, se define la rama "master" de nuestro repositorio de versiones como la rama reléase, de este modo, la construcción para esta rama será diferente.

Para la reléase de Android vamos a utilizar dos ficheros, uno que contiene la firma en un keystore y otro que contiene las claves de firma. Estos ficheros están incluidos

en el directorio raíz del proyecto, aunque hay más posibilidades de obtenerlos igual de válidas.

```
stage('construcción android') {
  steps {
      script {
          if  (env.BRANCH_NAME.equalsIgnoreCase('master')){
          //preparamos antes los ficheros de firma
            sh 'mv miApp-mobileapps.keystore platforms/android/'
            sh 'mv release-signing.properties platforms/android/'
            sh 'ionic cordova build android --release'

          }
          else{
            sh 'ionic cordova build android'

          }

      }

  }
```

35 Tareas de construcción Android

Para la construcción release de IOS es necesario dar de alta antes en la plataforma para desarrolladores la app, de dicha plataforma nos descargaremos el fichero plist que deberemos colocar en el path platforms/ios/%NombreDeLaApp%. Al igual que con Android, dispondremos del fichero plist en el directorio raíz de muestro código.

```
stage('construcción ios') {
 steps {
   script {
          if  (env.BRANCH_NAME.equalsIgnoreCase('master')){
          //preparamos antes el fichero de release
            sh 'mv IonicConferenceAppIsma-Release.plist platforms/ios/IonicConferenceAppIsma/'

            sh 'ionic cordova build ios --release'

          }
          else{
            sh 'ionic cordova build ios'

          }
      }
 }
```

36 Construcción IOS

7.4.3.5 Acciones post script

Jenkins nos permite definir tareas una vez que el script se ha ejecutado, permitiendo realizar unas u otras dependiendo del estado de la ejecución. En nuestro caso se realizan las siguientes tareas:

- Si el pipeline se ejecuta correctamente, "success", procedemos a publicar en Jenkins los artefactos generados (apk Android, web responsive) completando así la **CD**
- En todos los casos, "always", publicamos el informe de test unitarios y limpiamos el repositorio.

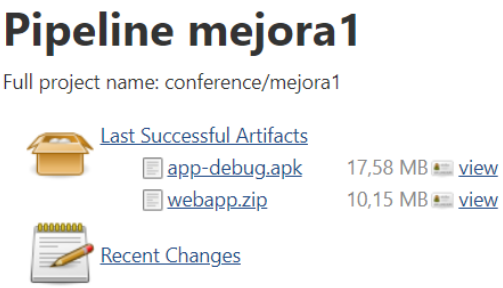
```
post {

    success{
        archiveArtifacts artifacts: 'platforms/android/app/build/outputs/apk/**/*.apk', fingerprint: true
        sh 'ls -la'
        zip zipFile: 'webapp.zip', archive: false, dir: 'www'
        archiveArtifacts artifacts: 'webapp.zip', fingerprint: true

    }
    always {
      junit allowEmptyResults: true, testResults: 'testresults/*.xml'
      echo 'no borra'
     }
}
```

37 tareas post build de Jenkins

7.4.4 Resultado de la ejecución

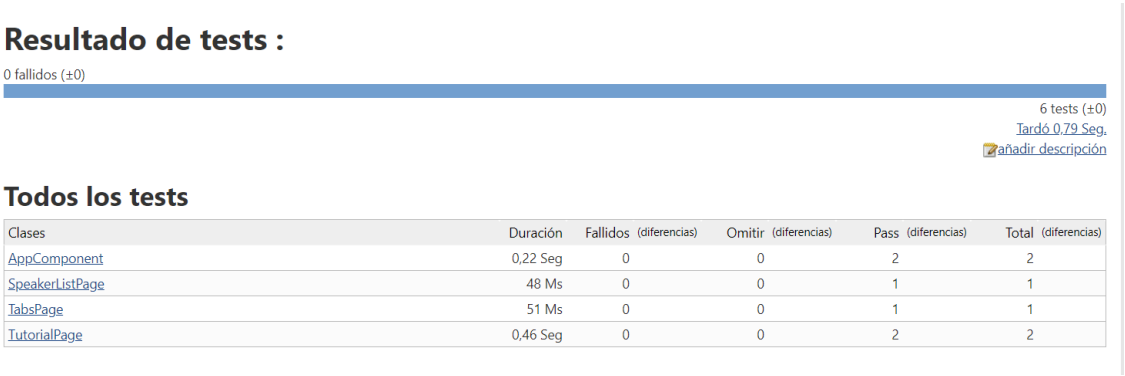
Tras la ejecución de nuestro pipeline, Jenkins nos va a proporcionar la siguiente información:

- Artefactos generados para su descarga. En este caso no se realiza la parte de despliegue continuo y debemos descargar los artefactos generados para su posterior instalación



38 Artefactos generados

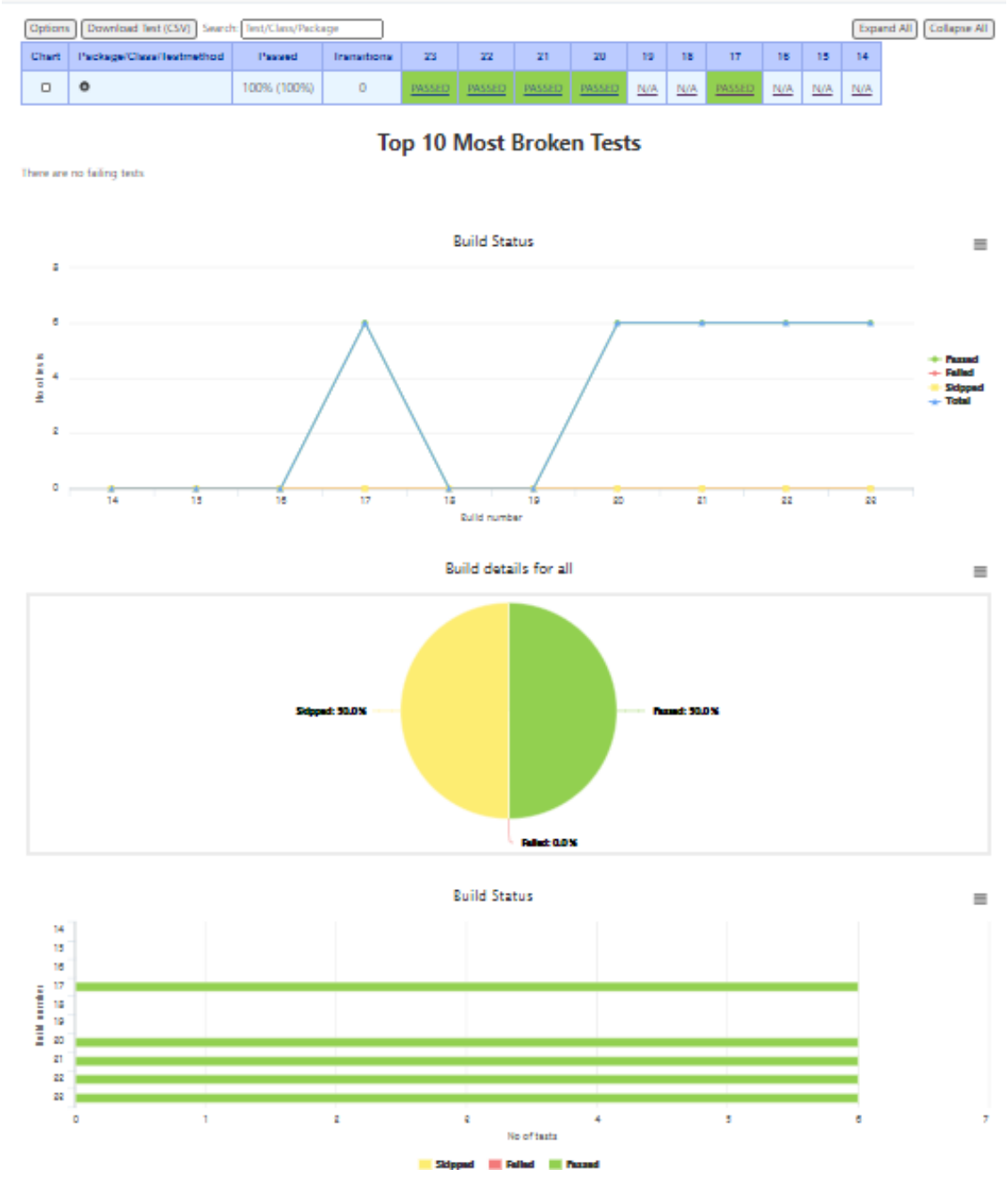
- Información de la ejecución de test. Nos indica por cada fichero de test la siguiente información



39 Resultado de test

Pinchado en la opción “Test Result Analyzer” obtenemos un historio de los test unitarios realizados y su resultado.

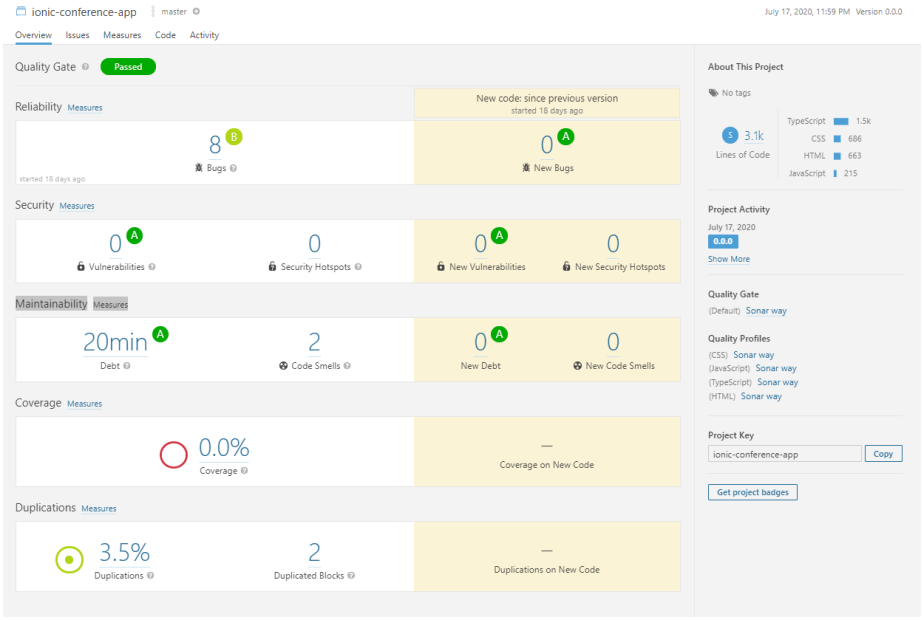
1. Debajo nos muestra un gráfico de la activad de los análisis realizados y por último nos informa de la versión de código, el umbral de calidad y los perfiles seleccionados para el proyecto (en nuestro caso será Isma tanto para java como para web).



40 Resumen de ejecuciones de Test

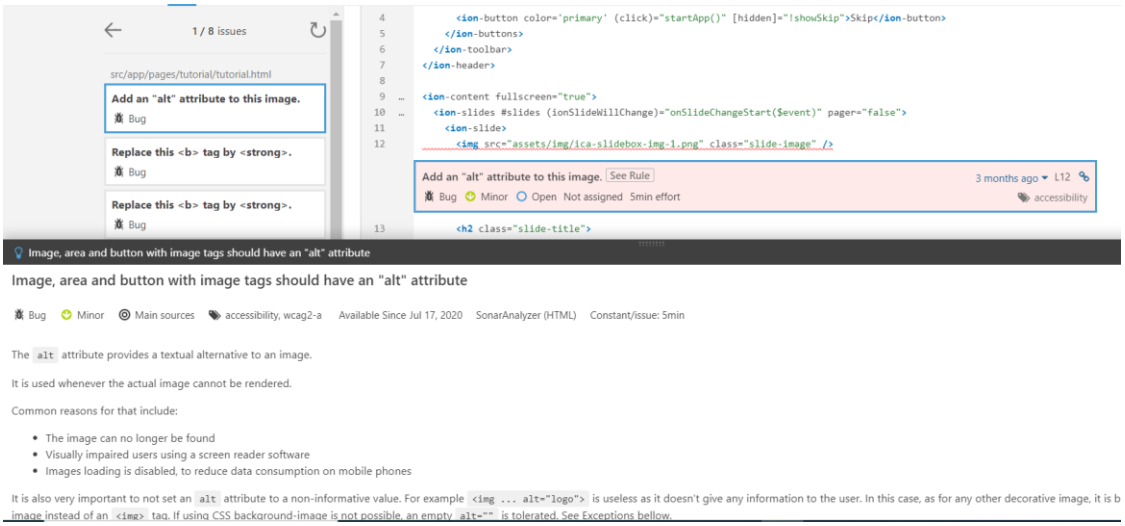
2. Análisis en sonarQube. En la siguiente ilustración se muestra el dashboard de la herramienta Sonarqube. En primer plano presenta agrupados los datos del Reliability, Security, Maintainability indicándonos la deuda técnica en un

segundo nivel, también indica la información sobre los test (cobertura de test y test unitarios realizados) y en último lugar los datos referentes a duplicidades de código. Los datos sombreados en sepia son la comparativa con los anteriores análisis. En la parte derecha del panel nos aparecen datos relativos al número de líneas de código por lenguaje y el tamaño total del código.



41 DashBoard de sonarQube

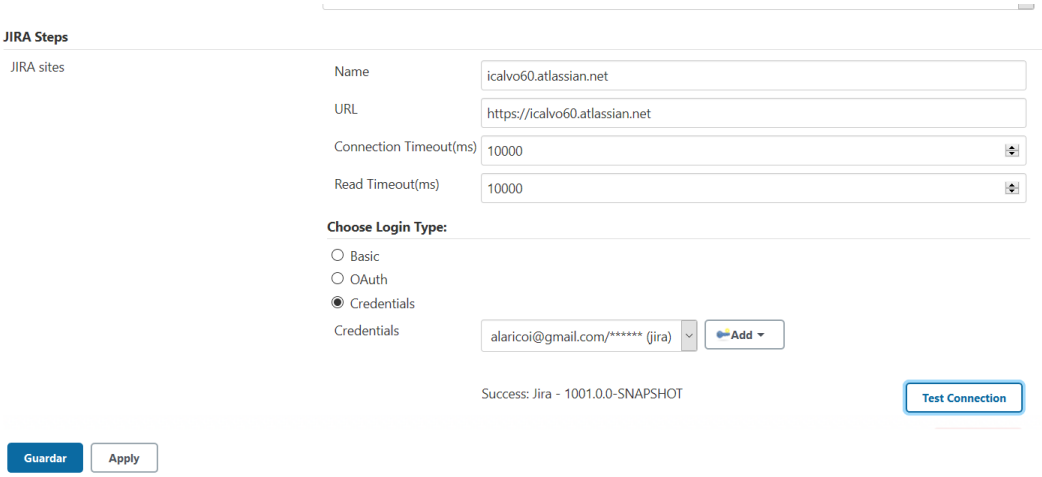
Si consultamos la pestaña “Issues” nos muestra el detalle de las evidencias detectadas ordenas, podemos consultar la situación en el código fuente, descripción de la regla y también nos permite marcarla como falso positivo, incorporar un comentario, asignar a un técnico.



42 Detalle de evidencia en SonarQube

7.4.5 Integración con Jira

Para la integración con Jira Cloud he utilizado el plugin de Jenkins "Jira Pipeline Steps" que nos permite una completa integración sobre nuestro proyecto Jira. Primero, en el panel de administración de Jenkins, Configuración de Sistema, debemos introducir los datos del servidor de Jira y sus credenciales de acceso en el apartado "Jira Steps":



43 Configuración de Jira Steps en Jenkins

Una vez configurado ya podemos añadir las llamadas a Jira que consideremos oportunas en nuestros pipelines de Jenkins. El plugin nos permite el

acceso a todo el API REST de Jira, para nuestro proyecto solo hemos integrado la apertura de incidencias y tareas, pero se pueden realizar desde tareas de administración, asignación de tareas, versionado, subida de ficheros, etc.

Lo primero que vamos a realizar es la definición de parámetros de nuestro proyecto Jira dentro del pipeline:

Jenkinsfile 7 KB

```
    def proyectoJira ='MIAP'
    def serverJira='icalvo60.atlassian.net'
    def errorJira = '10008'
    def tareaJira = '10006'

pipeline {
  agent  { label 'macOsServer'}
```

44 Parámetros Jira en JenkinsFile

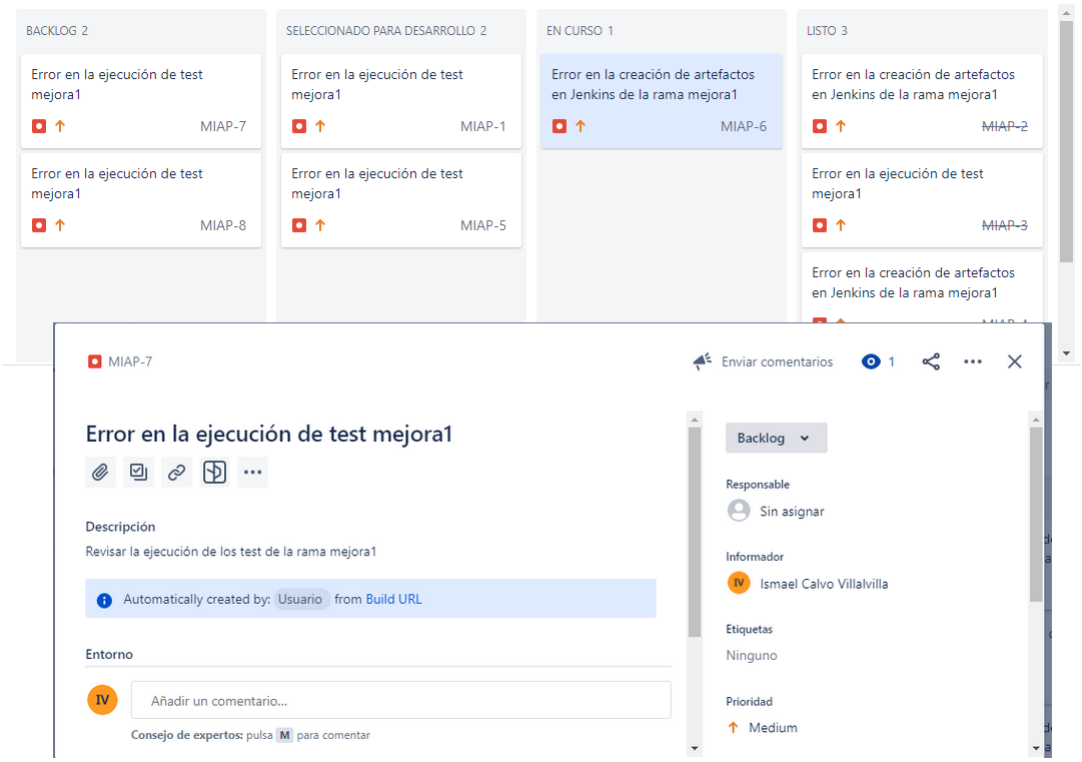
En cada paso de nuestro pipeline introducimos un bloque post con la ejecución en Repositorio de versiones proceso termina satisfactoriamente creamos un ticket de tipo tarea para que el equipo de desarrollo realice las acciones definidas para el proyecto como pueden ser el despliegue en entornos o Stores, pruebas funcionales para dispositivos, creación de versiones reléase, etc.

```
stages {
  stage('preprarar dependencias'){
    steps {
      sh 'npm install'
    }
    post{
      failure {
        script {
          testIssue = [fields: [ project: [key: proyectoJira],
                summary: 'Error al descargar dependencias ' + env.BRANCH_NAME,
                description: 'Revisar dependencias y los accesos al repositorio node',
                issuetype: [id: errorJira]]]

          response = jiraNewIssue issue: testIssue, site: serverJira

        }
      }
    }
  }
```

45 Ejemplo de Integración con Jira

En nuestro proyecto Jira se crea el ticket correspondiente sobre el tablero en la lista "BACKLOG" con los datos suministrados desde Jenkins. Para nuestro trabajo me decantado por un ticket simple pero la integración nos va a permitir desde añadir un fichero a asignar directamente a un miembro del Equipo. Otra posible integración es que en vez de crear ticket crear comentarios sobre un ticket determinado.

La apariencia en Jira, para este caso, en un tablero Kanban es la siguiente:

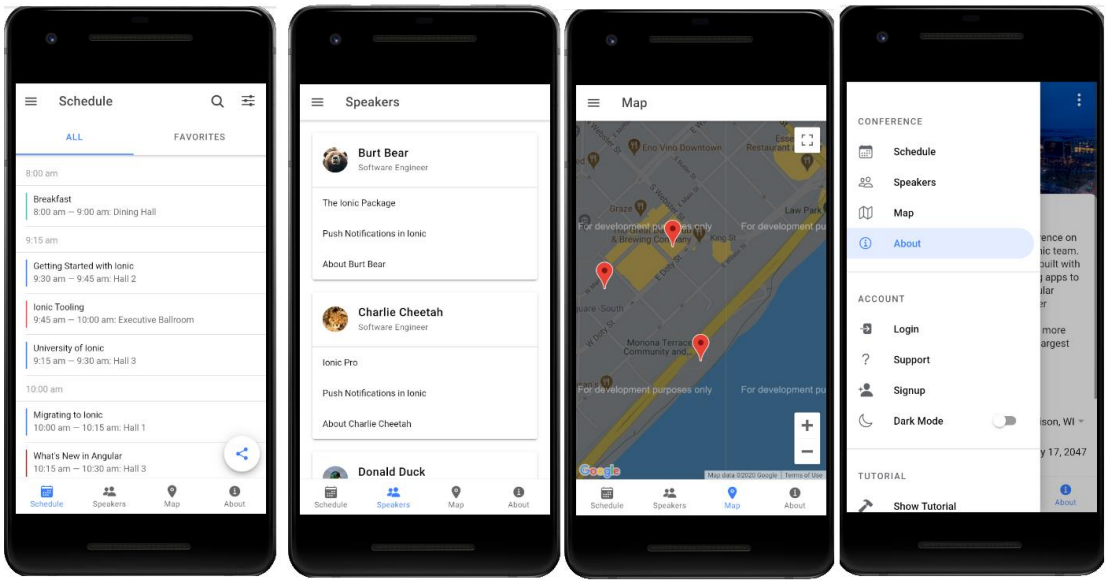


46 Ejemplo de ticket Jira

8 Caso de Uso

Para realizar la prueba de concepto de nuestro modelo, como comenté en el punto 7.3, opté por un desarrollo sobre la plantilla Conference de Ionic Framework. El nombre dado a la aplicación es miApp, se trata de una aplicación IOS, Android y webapp para navegadores.

La funcionalidad de la aplicación es una serie de ejemplos de uso de ionic como son las listas, el mostrar citas calendarios de eventos, la ejecución de búsquedas o la geolocalización.



47 Aplicación del Caso practico

Hemos simulado dos equipos de trabajo donde uno de ellos se encargará de realizar las mejoras y el otro de reparar correctivos. Partiremos de una situación inicial donde solo tenemos nuestra rama master del proyecto. En ese momento vamos a dar de alta nuestro proyecto en Jenkins según las indicaciones del punto 7.4.2. Automáticamente, por la definición que hicimos de webhook en nuestro proyecto en gitLab, Jenkins va a inspeccionar las ramas de nuestro proyecto, en este caso solo master, y procederá a ejecutar el pipeline CI/CD definido.

48 Situación inicial del Multibrach Job

Siguiendo el modelo de gitflow que hemos definido, el responsable de proyecto crea la rama desarrollo con el fin de crear tantas ramas de mejoras como sean necesarias. Como cualquier cambio en gitLab, Jenkins descubre la nueva rama y ejecuta su pipeline.

En este punto los diferentes equipos ya se ponen a realizar el desarrollo de las mejoras solicitadas por el usuario sobre la rama que le corresponda. En nuestro ejemplo hemos abierto una rama de mejora. El equipo de desarrollo realizará los cambios oportunos sobre el código y realiza periódicamente actualizaciones sobre su rama. Cada integración va a generar los entregables de todas las plataformas.

Al estar desarrollando una app hibrida la mayoría de los cambios sobre el código no están relacionados con la interacción con el dispositivo y solo influyen en la parte de tecnologías web con lo que hasta que no se obtiene un resultado final no se suele probar a generar los entregables para dispositivos, el motivo principal es que las tareas de generación y compilación de plataformas necesitan de un entorno configurado y actualizado, además para aplicaciones IOS se debe realizar sobre un Mac OS, estas tareas conllevan un tiempo relativamente grande donde el desarrollador se bloquea no pudiendo dedicar tiempo a tareas que pueden ser más productivas. Con el modelo propuesto de CI/CD nos aseguramos de que los entregables se generan correctamente con cada integración sin necesidad que el equipo de desarrollo haga esta integración manualmente Por otra parte. Con CI/CD evitamos que el equipo de desarrollo tenga que realizar tareas mecánicas que incluso deban realizar en otro dispositivo distinto al habitual de desarrollo.

En este punto Jenkins habrá generado 3 pipelines dentro del Job de Multibranchs, uno por cada rama.

49 Situación del Proyecto al comienzo de las mejoras

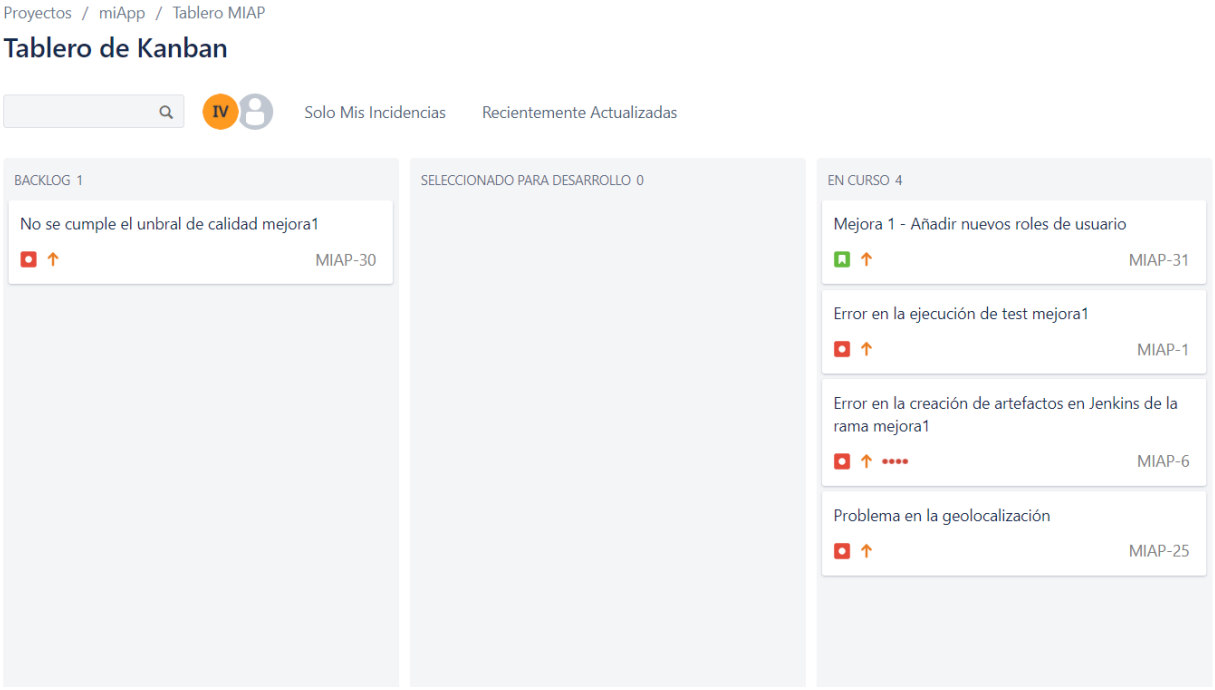
Durante el desarrollo, como es habitual en cualquier proyecto real, se reporta un bug que necesita relación inmediata y no puede esperar a incluirlo en la entrega de la mejora. En este caso, lo que realizaremos es una rama de tipo correctivo directamente desde el master, habitualmente se nombra con el código de incidencia que. en nuestro caso, nos entró por Jira Services Desk con el número 25. En este momento ya tenemos a los dos equipos trabajando sobre la misma aplicación, en distintas ramas.

50 Situación con la inclusión de un correctivo

Según vamos progresando en el desarrollo de nuestro proyecto, el proceso CI/CD que hemos definido nos ha marcado que tenemos incumplimientos del Umbral de Calidad definido en SonarQube lo que lleva a no generar los artefactos y abrir una incidencia en Jira para que el equipo tenga lo tenga en consideración. El equipo tiene constancia en fases tempranas del desarrollo de posibles problemas de seguridad o de bugs inesperados.

51 Incumplimiento en el Umbral de Calidad SonarQube

En Jira se han ido los bugs detectados por el usuario y las mejoras planteadas, pero también se van a ir registrando los problemas aflorados por el proceso de CI/CD para que el equipo trabaje sobre ellos. En la figura siguiente podemos observar cómo aparecen el correctivo "MIAP-25" y la historia de usuario "Mejora1" pero también podemos observar que el equipo tiene en curso reparar los problemas reportados por Jenkins como un problema de compilación en una integración de la rama mejora 1 (MIAP-6) o que los test de karma no se han superado (MIAP-1). En la lista de tareas por realizar aparece la nueva incidencia reportada de no superación de los Umbrales de Calidad (MIAP-30). Como se ha superado los test, el quipo puede cerrar la incidencia MIAP-1.



52 Tablero Kanban Jira con las tareas por estados

Una vez que el equipo da por corregido el bug MIAP-25, generando los artefactos sin problemas y cumpliendo con los estándares de Calidad, el jefe de proyecto procede a realizar el merge sobre la rama master para crear una nueva versión de nuestro proyecto y marcando un nuevo tag de versión, en este momento, de manera autónoma, el proceso de CI/CD nos va a generar los entregables para la instalación. En muchos casos me he encontrado con que las versiones release no las generan los equipos de desarrollo sino un equipo de paso a producción, que son los únicos que disponen de las credenciales de desarrollador de las distintas plataformas, para realizar el proceso. Deben realizar manualmente la clonación del repositorio, descarga de dependencia, creación de plataformas, compilación y generación de entregables, además, por las características de la arquitectura, suelen realizarse en equipos informáticos definidos exprofeso, distinto al habitual de trabajo. Con el sistema propuesto abstraemos de esta tarea al departamento de producción que se encontrará como los entregables ya generados con sus credenciales correspondientes.

Otro momento crítico en la generación tradicional de release es que suele ser en este momento cuando los departamentos de producción ejecutan las revisiones de calidad del producto, a estas alturas estamos en una fase muy avanzada de nuestro desarrollo y nos

podemos encontrar con un informe desfavorable que nos obligue a retrasar en gran medida el plazo de entrega, con el consiguiente perjuicio en perdida de negocio y penalizaciones. Con nuestro modelo CI/CD hemos detectado en fases tempranas del desarrollo tanto problemas de tests fallidos como incumplimientos en los umbrales de calidad.

53 Desparece el Job bug-MIAP-25

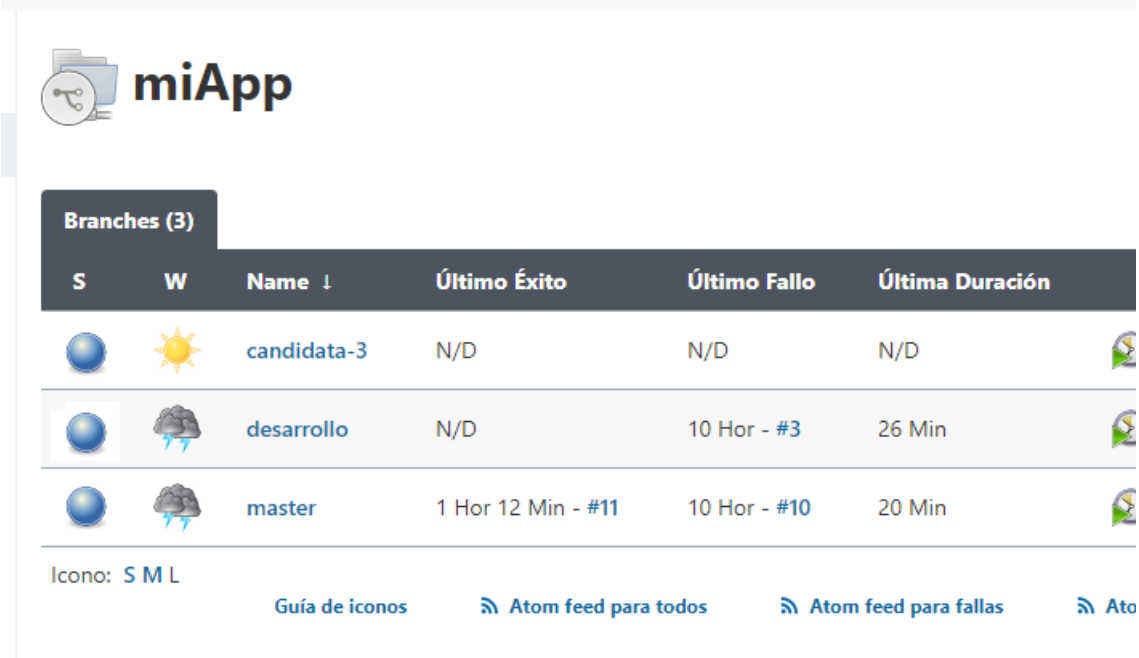
54 generación de Release

También se realiza un merge de la rama de correctivos ya coincidente con la rama master, a la rama de desarrollo. En caso de colisión siempre va a prevalecer el código de la rama

master ya que es la que es desplegada en producción. Una vez realizadas las fusiones, la rama bug-MIAP-25 desaparece y por tanto también desaparece automáticamente su job dentro del Multibrach Job de Jenkins.

Tras implementar los requisitos solicitados en la historia de usuario "Mejora 1", se ha comprobado previamente que se cumple con los criterios de calidad, el equipo de desarrollo solicita la fusión con desarrollo al jefe de proyecto. En ocasiones, los equipos de desarrollo están externalizados por lo que este paso, en modelos tradicionales, se convierte en verdaderos pasos a producción en entornos previos, con todos los inconvenientes y retrasos descritos con anterioridad. Con un modelo de CI/CD este proceso se automatiza teniendo de manera más aséptica los entregables de cada fusión.

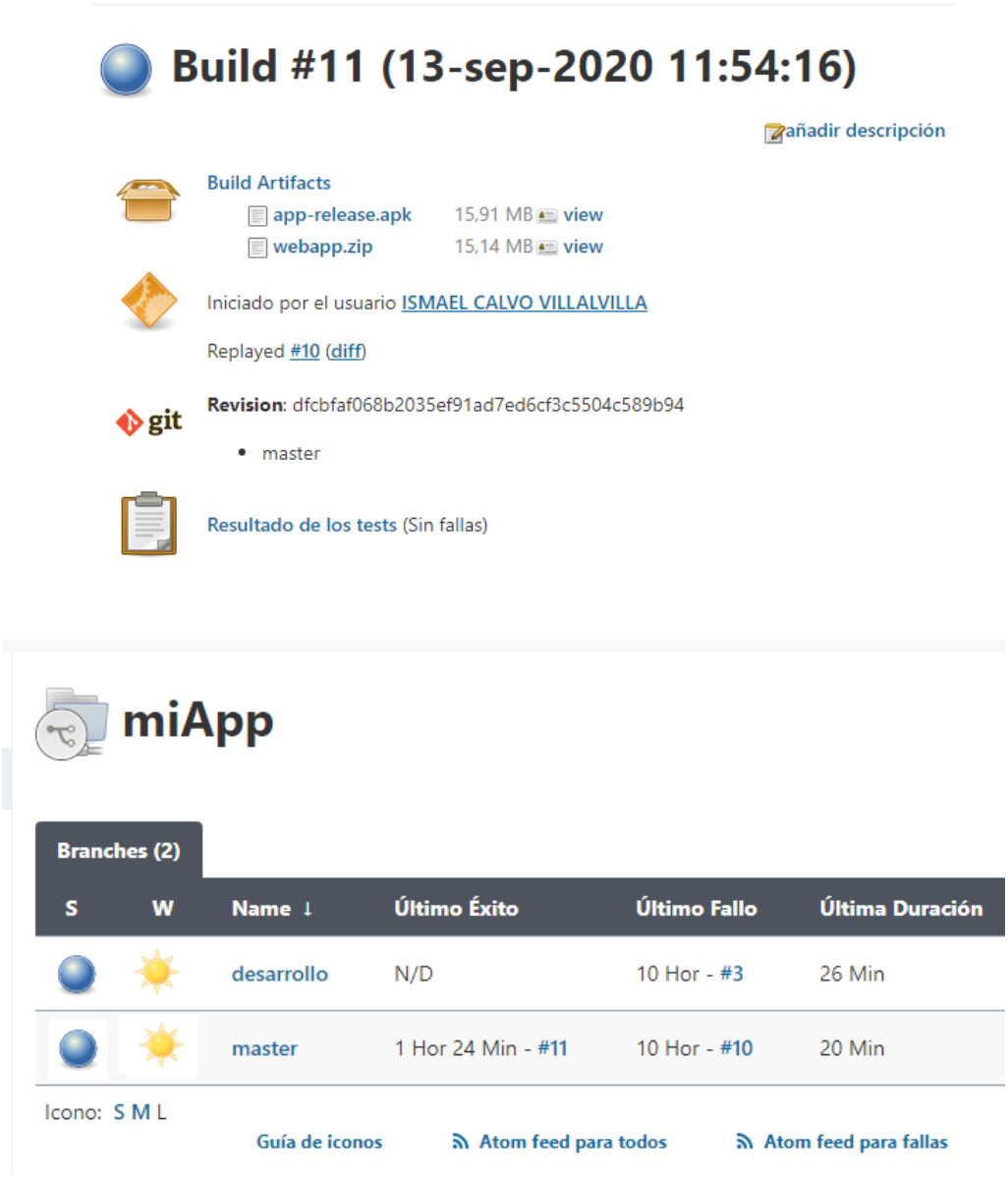
En este momento es cuando se crea una rama de release candidata donde se realizarán las pruebas de aceptación con el Cliente.



55 Creación de versión candidata

Cuando queramos generar nuestra versión final a partir de la candidata bastara con que el jefe de proyecto realice las consiguientes fusiones de ramas en el repositorio según el modelo propuesto, es decir, la rama candidata se fusiona tanto con la rama master como con

la rama de desarrollo. Nuestro modelo CI/CD, al igual que sucedió cuando arreglamos el correctivo, genera los correspondientes entregables para que proceda el departamentalmente de implantación a distribuir la nueva versión.



56 Situación tras generar la versión desde mejora

9 Conclusiones

Al comienzo del estudio nos habíamos propuesto unos objetivos con el fin de mejorar los procesos de desarrollo y mantenimiento de software a partir de un modelo CI/CD. Tras definir dicho modelo, hemos realizado un caso práctico para evaluar el grado de consecución de los objetivos planteados:

1. Mejorar los procesos integración de código entre los diferentes miembros del equipo. En el caso de uso hemos observado como el modelo CI/CD nos genera en cada integración los arquetipos de nuestra aplicación, si en una integración se produce un error de compilación o de generación el sistema es capaz de generar una incidencia para que se revise y repare, este descubrimiento del problema es inmediato en la integración por lo que acometer su solución es a priori menos complicado que si lo descubriéramos en fases más avanzadas de nuestro proyecto y tras sucesivas integraciones que arrastren el problema.
2. Detección temprana de desvíos en la calidad del producto o errores de funcionamiento. También hemos visto en el caso de uso como el sistema detectó de manera rápida los errores en los test unitarios y de integración o la degradación en la calidad en las revisiones de código realizadas con SonarQube, tenemos una visión real del estado de nuestro software, evitando problemas que afloren en fases más avanzadas del desarrollo y donde acometer la solución suele ser más costoso.
3. En el caso particular de tecnologías de desarrollo híbrido para aplicaciones móviles, la construcción de entregables para distintas plataformas se realiza mediante tareas mecánicas. En muchas organizaciones (un ejemplo real es donde desempeño mi actividad laboral) los desarrollos se contratan a terceras empresas y que no crean directamente las release, en estos casos deben solicitar la creación a un equipo interno que se encarga de la implantación y que debe generar las aplicaciones para las tiendas o Stores, utilizando las credenciales, de forma manual. El sistema propuesto permite liberar al equipo de implantación de tareas mecánicas y que no aportan valor, hemos podido observar que un ciclo

de construcción completo en el caso de estudio, con un desarrollo pequeño, tarda unos 20 minutos.

También cabe reseñar que, como ocurre en cualquier metodología o modelo de trabajo, la implicación del equipo en las practicas CI/CD es de suma importancia, sobre todo a la hora de definir el plan de pruebas y llevarlo a cabo. No es la primera vez que nos encontramos con proyectos donde el equipo no lleva a cabo la integración del código o lo hacen en contadas ocasiones, también nos podemos encontrar con que no se implementan las pruebas definidas o incluso no se definen pruebas. En estos casos, la implantación de este tipo de modelos no incidirá tan positivamente en nuestros desarrollos y se limitará a automatizar tareas de construcción, pero sin asegurar la calidad de nuestros productos.

También se puede señalar que las buenas prácticas a la hora de llevar a cabo un proyecto dependen de la metodología de desarrollo utilizada, por ejemplo, si utilizamos el TDD (Test-Driven Development) desarrollo dirigido por test está claro que en CI/CD vamos a obtener productos más fiables a los fallos ya la colección de tests que se ejecutan en cada fase es alta, al igual que las metodologías agiles inciden en la periodicidad de las ejecuciones.

Por último, debemos indicar que no se pudo realizar la prueba completa de la construcción release de IOS, el motivo es que se necesita poseer una licencia desarrollo Apple, de un coste de 99 dólares anuales por lo que no se ha probado la creación de los ficheros necesarios para subir la aplicación al Apple Store si bien si se deja indicado que ficheros se requieren y como realizarlo.

10 Futuras líneas de trabajo

Como líneas de futuras investigaciones podemos incluir:

1. Integración de pruebas en dispositivos móviles. Las pruebas que deberían integrarse son:
    - Pruebas funcionales. A parte de las ya definidas en karma en este modelo, habría que volver a realizarlas sobre dispositivo e incluir pruebas de instalación y desinstalación, ejecuciones en segundo plano, uso de local storage, pruebas de integración con el dispositivo (notificaciones, cámara, gps, …).
    - Test de rendimiento o pruebas de carga del sistema. Si nuestra app consume demasiados recursos, gasto excesivo de batería o datos de conexión, consumo excesivo de memoria o no la libera adecuadamente.

    Habría que explorar las diferentes soluciones en el mercado, productos como AWS Device Farm, Genymotion, Xamarin Test Cloud o BrowserStack nos permite el uso de granjas de dispositivos en la nube para realizar nuestras pruebas, estas soluciones son todas de pago. También existen framework Open Sources que realizan las acciones sobre simuladores como Apium, Selendroid (Selenium para Andoid).

2. Explorar la posibilidad de virtualización con contenedores Docker[42] del nodo Jenkins Mac Os X. Durante la prueba de concepto, al trabajar con varias ramas simultáneamente al mismo tiempo he observado como el rendimiento del nodo caía al ejecutarse al mismo tiempo 4 o 5 pipelines, en este caso más acentuado al utilizar una imagen de Virtual Box[43] para pc. Aunque para un entorno real se utiliza un dispositivo Mac físico nos podemos encontrar con limitaciones de ejecución. Para evitar este desbordamiento, una posible vía de continuación del

[42] Sistema de virtualización ligero, Open Source, basado en imágenes que maximiza recursos. https://www.docker.com /
[43] Oracle Virtual Box. Producto que permite virtualizar Sistemas Operativos. https://www.virtualbox.org/

proyecto sería utilizar un sistema de contenedores con una imagen Mac OS X con todo lo necesario para construir nuestras aplicaciones, en este caso, Jenkins se encargaría de levantar la imagen y ejecutar el pipeline en el contenedor, liberando la memoria una vez acabado el proceso. Además de los aspectos técnicos habría que consultar los aspectos legales de licencias Mac OS X y Xcode (el resto de las herramientas que hemos utilizado en el modelo son de libre distribución).

11 Referencias

[1] Jez Humble and David Farley, *Continuous Delivery - Reliable Software Releases Through Build, Test And Deployment Automation.* Ed. Addison-Wesley. 2010

[2] Paul M. Duvall with Steve Matyas and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Ed. Addison-Wesley. 2007

[3] *Ken Beck - Test Driven Development: By Example* Ed. Addison-Wesley 2002

[4] L. Chen, "Continuous Delivery: Huge Benefits, but Challenges Too," in *IEEE Software*, vol. 32, no. 2, pp. 50-54, Mar.-Apr. 2015.

[5] *Blog de Martin Fowler:*

- https://martinfowler.com/bliki/ContinuousDelivery.html
- https://martinfowler.com/articles/continuousIntegration.html

[6] Web oficial de Jenkins: https://jenkins.io/

[7] Web Oficial de NodeJS: https://NodeJS.org/es/

[8] Web oficial npm: https://www.npmjs.com/

[9] Web oficial de SonarQube: https://www.sonarqube.org/

[10] Web oficial de Apache Cordova: https://cordova.apache.org/

[11] Web Oficial de ionic Framework: https://ionicframework.com/

[12] Open Web Application Security Project https://owasp.org/

[13] Atlassian https://www.atlassian.com/

[14] Open Handset Alliance™ http://www.openhandsetalliance.com/

[15] Circle Internet Services, Inc. https://circleci.com/

[16] https://stackoverflow.com/

[17] Jasmine test Framework https://jasmine.github.io/

[18] Karma runner https://karma-runner.github.io/

[19] Manual de Jira Steps Plugins de Jenkins https://jenkinsci.github.io/jira-steps-plugin/index.html

[20] Web Apium, framework de test http://appium.io/

[21] http://selendroid.io/

[22] https://docs.docker.com/

[23] Herramientas para test cloud

- https://www.genymotion.com/
- https://aws.amazon.com/es/device-farm/
- https://appcenter.ms/

12 Anexo: fichero JenkinsFile.

```
  def proyectoJira ='MIAP'
  def serverJira='icalvo60.atlassian.net'
  def errorJira = '10008'
  def tareaJira = '10006'
  pipeline {
  agent{ label 'macOsServer'}
  environment {
       ANDROID_HOME="/Users/isma/Library/Android/sdk"
       ANDROID_SDK_ROOT ="/Users/isma/Library/Android/sdk"
       JAVA_HOME="/Library/Java/JavaVirtualMachines/adoptopenjdk-8.jdk/Contents/Home/"
       GRADLE_HOME="/usr/local/Cellar/gradle/6.5.1"
       PATH =
"GRADLE_HOME:JAVA_HOME:$ANDROID_HOME:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library
/Apple/usr/bin"

   }
  stages {

           stage('preprarar dependencias'){
                    steps {
        sh 'npm install'

      }
      post{
        failure {
          script {
            testIssue = [fields: [ project: [key: proyectoJira],
                summary: 'Error al descargar dependencias ' + env.BRANCH_NAME,
                description: 'Revisar dependencias y los accesos al repositorio NodeJS',
                issuetype: [id: errorJira]]]

            response = jiraNewIssue issue: testIssue, site: serverJira

                        }

        }
      }

}
    stage('lint'){
                    steps {
        sh 'npm run lint'
      }
      post{
        failure {
          script {
            testIssue = [fields: [ project: [key: proyectoJira],
              summary: 'Error en la ejecución de lint ' + env.BRANCH_NAME,
              description: 'Revisar la ejecución de lint y los errores del log',
              issuetype: [id: errorJira]]]

            response = jiraNewIssue issue: testIssue, site: serverJira

                        }

        }
      }
                }
           stage('test'){
                    steps {
                              sh 'npm run test'
      }
      post{
        failure {
```

```
          script {
            testIssue = [fields: [ project: [key: proyectoJira],
              summary: 'Error en la ejecución de test ' + env.BRANCH_NAME,
              description: 'Revisar la ejecución de los test de la rama ' +
env.BRANCH_NAME,
              issuetype: [id: errorJira]]]

            response = jiraNewIssue issue: testIssue, site: serverJira

                        }

        }
     }
           }

           stage("SonarQube analysis") {

     steps {
        withSonarQubeEnv('sonarqube') {
          script{
            def packageJSON = readJSON file: 'package.json'
            def packageJSONVersion = packageJSON.version
            def packageJSONName = packageJSON.name
            def   scannerHome = tool 'sonar-scanner-mac';
            def sonar = "${scannerHome}/bin/sonar-scanner"
             sonar +=  " -Dsonar.projectKey=" + packageJSONName
             sonar +=  " -Dsonar.projectName=" + packageJSONName
             sonar +=  " -Dsonar.projectVersion=" + packageJSONVersion
             sonar +=  " -Dsonar.sources=src"
             sonar +=  " -Dsonar.sourceEncoding=UTF-8"
             sonar +=  " -Dsonar.exclusions=**/NodeJS_modules/**,**/*.spec.ts"
             sonar +=  " -Dsonar.tests=src"
             sonar +=  " -Dsonar.test.inclusions=**/*.spec.ts"
             sonar +=  " -Dsonar.ts.tslintconfigpath=tslint.json"
             sonar +=  " -Dsonar.ts.lcov.reportpath=test-results/coverage/coverage.lcov"
            sh sonar
          }

        }
      }
      post {
        failure {
          script {
            testIssue = [fields: [ project: [key: proyectoJira],
              summary: 'Error en la llamada a sonarQube Jenkins. Rama' + env.BRANCH_NAME,
                description: 'Revisar la llamada a sonarQube en JenkinsFile. '
                            + 'Comprobar que el servicio no está caído y esta accesible',
                issuetype: [id: errorJira]]]

            response = jiraNewIssue issue: testIssue, site: serverJira
                        }
       }
      }
   }
   stage("Quality Gate") {
     steps {
       timeout(time: 1, unit: 'HOURS') {
         waitForQualityGate abortPipeline: true
       }
     }
      post {
        // Si no cumple el umbral de calidad se aborta la ejecución y creamos el ticket
Jira
        failure {
         script {
           withSonarQubeEnv('sonarqube') {
              testIssue = [fields: [ project: [key: proyectoJira],
               summary: 'No se cumple el umbral de calidad ' + env.BRANCH_NAME,
               description: 'Revisar resultados de SonarQube ' + SONAR_HOST_URL ,
```

```
                issuetype: [id: errorJira]]]

              response = jiraNewIssue issue: testIssue, site: serverJira
            }
                        }
          }
        }
      }
     stage('construcción web') {
                    steps {
                      sh 'ionic build --prod'
                    }
       post{
        failure {
                        script {
            testIssue = [fields: [ project: [key: proyectoJira],
              summary: 'Error en la creación de web en Jenkins de la rama ' +
env.BRANCH_NAME,
              description: 'Error en la creación de web en Jenkins.',
              issuetype: [id: errorJira]]]

             response = jiraNewIssue issue: testIssue, site: serverJira
           }
         }
                      }
            }
            stage('creamos plataformas') {
                    steps {
                      sh 'ionic cordova platform add android'
                              sh 'ionic cordova platform add ios'
                    }
                    post {
         failure {
                        script {
             testIssue = [fields: [ project: [key: proyectoJira],
                 summary: 'Error en la creación de Plataformas de la rama ' +
env.BRANCH_NAME,
                 description: 'Revisar las depedncias de ionic y cordova',
                 issuetype: [id: errorJira]]]

            response = jiraNewIssue issue: testIssue, site: serverJira
                          }
         }
                    }
            }
            stage('construcción android') {
                    steps {
                        script {
                            if  (env.BRANCH_NAME.equalsIgnoreCase('master')){
                            //preparamos antes los ficheros de firma
                              sh 'mv miApp-mobileapps.keystore platforms/android/'
                              sh 'mv release-signing.properties platforms/android/'
                              sh 'ionic cordova build android --release'

                            }
                            else{
                              sh 'ionic cordova build android'

                            }

                        }

                    }
     post {
       failure {
                        script {
             testIssue = [fields: [ project: [key: proyectoJira],
               summary: 'Error en la creación de Android en Jenkins de la rama ' +
env.BRANCH_NAME,
```

```
              description: 'Error en la creación de Android en Jenkins.',
              issuetype: [id: errorJira]]]

            response = jiraNewIssue issue: testIssue, site: serverJira
                        }
        }
      }
    }
           stage('construcción ios') {
            steps {
                    script {
                          if  (env.BRANCH_NAME.equalsIgnoreCase('master')){
                          //preparamos antes el fichero de release
                            sh 'mv IonicConferenceAppIsma-Release.plist
platforms/ios/IonicConferenceAppIsma/'
                            sh 'ionic cordova build ios --release'
                          }
                          else{
                            sh 'ionic cordova build ios'
                          }

                      }
                  }
      post {
        failure {
                        script {
           testIssue = [fields: [ project: [key: proyectoJira],
             summary: 'Error en la creación de app IOS en Jenkins de la rama ' +
env.BRANCH_NAME,
              description: 'Error en la creaciónapp IOS s en jenkins Jenkins.',
              issuetype: [id: errorJira]]]

            response = jiraNewIssue issue: testIssue, site: serverJira
                        }
       }
                    }
    }
  }
   post {
           success{
                            archiveArtifacts artifacts:
'platforms/android/app/build/outputs/apk/**/*.apk', fingerprint: true
                       zip zipFile: 'webapp.zip', archive: false, dir: 'www'
             archiveArtifacts artifacts: 'webapp.zip', fingerprint: true
       script {
        testIssue = [fields: [ project: [key: proyectoJira],
          summary: 'Se han creado los artefactos en Jenkins de la rama ' + env.BRANCH_NAME
,
          description: 'Realizar las acciones correspondientes de despliegue en los
diferentes dispositivos',
          issuetype: [id: tareaJira]]]

        response = jiraNewIssue issue: testIssue, site: serverJira
               }
            junit allowEmptyResults: true, testResults: 'testresults/*.xml'
        deleteDir();
                   }
                   failure {
                    // deleteDir();
       echo 'no borra si fallo'
                   }
         }
}
```